ФГБОУ ВО "Московский государственный университет им. М.В.Ломоносова"
Механико-математический факультет
Кафедра математической теории интеллектуальных систем

На правах рукописи

Плетнев Александр Андреевич

Моделирование динамических баз данных.

Специальность 01.01.09 — дискретная математика и
математическая кибернетика

Диссертация

на соискание ученой степени
кандидата физико-математических наук

Научный руководитель:
доктор физико-математических наук,
профессор Э.Э.Гасанов

Москва 2016

Оглавление

Введение

1 Общая характеристика работы

Актуальность темы

Диссертация является исследованием в области дискретной математики и математической кибернетики и посвящена изучению динамических баз данных. База данных (БД) — формализованное представление информации, удобное для хранения и поиска данных в нем. Понятие БД возникло в 60-е годы 20 века и связано с развитием вычислительной техники и информатики. Тематика теории БД связана с поиском удобного представления, компактного хранения, быстрого поиска и других свойств данных. Исследование в области баз данных широко известны из работ: Кодд (E.F.Codd) (реляционная модель данных) [3], В.Н.Решетников [4], Думи (A.Dumey) [5], А.П.Ершов (методы хеширования) [6], Бентли (J.L.Bentley) [7], Кнут (D.E.Knuth) [14], Ульман (J.D.Ullman) [9], Хаджеруп (Hagerup T.) и Каммер (Kammer F.) [10], Ружиц (Ruzic M.) [11], Шеймос (M.I.Shamos) и другие (сложность алгоритмов обработки данных) [12], Э.Э.Гасанов (информационно-графовая модель данных, сложность алгоритмов поиска) [1] и многих других.

В диссертации изучено направление БД, связанное с решением одной из задач информационного поиска (ЗИП), а именно динамическая задача поиска идентичных объектов (ДЗПИО). ЗИП – один из наиболее часто встречающихся на практике видов задач. Самым распространенным примером задачи поиска, встречающейся в любой базе данных, является задача поиска по идентификатору. Суть задачи состоит в том, что любой объект в базе данных имеет свой уникальный идентификатор. Идентификатором может быть уникальное имя или уникальное значение, например, автомобильный номер. Задача состоит в том, чтобы по заданному в запросе идентификатору найти в базе данных объект с этим идентификатором (если такой объект в базе имеется). Если множество идентификаторов может изменяться со временем, то эту задачу называют ДЗПИО.

Эффективное решение ДЗПИО — это актуальная проблема и на сегодняшний день. Современные ЭВМ позволяют использовать параллельные процессы для решения подобного рода задач. Исследования в области параллельного вычисления широ-

ко известны из работ: Гуибас (Leo J. Guibas) [17], Аржоманди (E.A. Arjomandi) [18], Чин (F.Y. Chin) [19], Беркмэн (O. Berkman) [20] и многих других.

Основная сложность параллельного вычисления заключается в считывании и записывании данных в одну и ту же область памяти. Различают несколько типов систем для параллельной обработки данных. Они различаются способами обращения к общей памяти:

- одновременное чтение, одновременная запись (ОЧОЗ). В этой системе разрешается одновременное чтение и одновременная запись разными процессами в общую память;
- одновременное чтение, эсклюзивная запись (ОЧЭЗ). Эта система разрешает одновременное чтение из общей памяти, но при этом одновременная запись разными процессами в общую память запрещена;
- эсклюзивное чтение, эсклюзивная запись (ЭЧЭЗ). В данной системе запрещается писать и читать из общей памяти разными процессам.

Подробно эти системы описаны в работах Гафни (E. Gafni) [23], Снир (Snir M.) [24]. Применительно к задаче поиска идентичного объекта было предложено множество различных алгоритмов, использующих параллельные вычисления. Они основываются на таких известных структурах данных, как сбалансированное бинарное дерево, красно-черное дерево, 2—3 дерево. В основном, цель всех исследований в этой области заключается в добавлении нескольких записей к структуре данных за наименьшее время. Более детально о полученных результатах можно прочитать в работах: Комер (Comer D.) [16], Паркс (Parks H.) [21] и Ванг (Wang B-F) [22].

Цель работы

Основной целью работы является построение математической модели динамических баз данных, кроме того решаются следующие задачи.

- Доказать применимость предлагаемой модели для решения задач, возникающих в динамических базах данных.
- Разработать бесконечно распараллеливаемые структуры данных для решения ДЗПИО для любого потока запросов.
- Предъявить бесконечно распараллеливаемую структуру данных для решения ДЗПИО с логарифмической сложностью для любого потока запросов.
- Получить оценки параметров модели, при которых возможна обработка произвольных потоков запросов.

Методы исследования

В работе используются методы теории автоматов, теории сложности управляющих систем и теории графов.

Научная новизна

В работе представлена модель динамических баз данных. В рамках этой модели можно описывать алгоритмы, использующие параллельные процессы и оценивать их сложность.

В диссертации предложена бесконечно распараллеливаемая ОЧЭЗ структура данных, решающая динамическую задачу поиска идентичного объекта для любого потока запросов с логарифмической сложностью.

Получены минимально возможные значения параметров модели, при которых база данных может обрабатывать произвольный поток запросов.

Теоретическая и практическая значимость

Работа имеет теоретический.

Введенная модель позволяет сравнивать между собой различные алгоритмы для решения ДЗПИО, в том числе известные классические алгоритмы и алгоритмы, использующие параллельные процессы. Также полученные в работе результаты являются новыми и представляют интерес для специалистов в области дискретной математики и теории алгоритмов.

Кроме этого, представленный в работе алгоритм решения ДЗПИО для любого потока запросов с логарифмической сложностью может быть использован в прикладных задачах.

Апробация работы

Результаты диссертации докладывались на следующих научных семинарах и всероссийских и международных конференциях.

(1) Семинар "Теория автоматов"под руководством академика, профессора, д.ф-м.н. В.Б. Кудрявцева (2014-2015 гг., неоднократно);

(2) Семинар "Вопросы сложности алгоритмов поиска"под руководством проф., д.ф-м.н. Э.Э.Гасанова (2009-2015 гг., неоднократно);

(3) Международная конференция студентов, аспирантов и молодых ученых "Ломоносов"(7-11 апреля 2014, 13-17 апреля 2015, Москва, МГУ);

(4) X Международный семинар "Дискретная математика и ее приложения"(1-6 февраля 2010, Москва, МГУ);

(5) X Международная конференция "Интеллектуальные системы и компьютерные науки"(21-26 ноября 2011, Москва, МГУ);

(6) XI Международный семинар "Дискретная математика и ее приложения посвященный 80-летию со дня рождения академика О.Б. Лупанова (18-23 июня 2012, Москва, МГУ);

(7) Республиканская научная конференции с участием зарубежных ученых "Современные методы математической физики и их приложения"(15–17 апреля 2015, Ташкент, Национальный университет Узбекистана им. М. Улугбека).

Публикации

По теме диссертации опубликовано десять печатных работ, из них шесть работ в журналах ВАК РФ. Работ, написанных в соавторстве, нет.

Структура и объем работы

Диссертация состоит из введения, трех глав и заключения. Объем диссертации 131 страница. Список литературы содержит 34 наименования.

2 Краткое содержание работы

В введении описана предметная область и история вопроса, даны основные используемые определения, сформулированы результаты диссертации.

В первой главе вводится модель динамических баз данных для решения задач информационного поиска. Полученная модель применяется для решения ДЗПИО, а как следствие, доказывается, что конечный объект (автомат) может поддерживать бесконечную структуру (информационный граф). В конце первой главы модель обобщается на случай потоковых запросов к базе данных. Структуры данных, полученные в рамках этой модели, называем потоковый динамический информационный граф (ПДИГ).

Во второй главе доказываются верхние оценки на ПДИГ. Первая оценка заключается в том, что существует ПДИГ, который решает ДЗПИО для любого потока запросов. Кроме этого, в этой главе доказывается, что существует ПДИГ, решающий ДЗПИО для любого потока запросов с логарифмической сложностью. Так же в конце второй главы приводится пример минимального по степени ветвления ПДИГ с радиусом видимости один, решающий ДЗПИО для любого потока запросов.

В третьей главе доказываются нижние оценки на ПДИГ. Автор показал, что не существует ПДИГ со степенью ветвления один, решающий ДЗПИО для любого потока запросов. Также доказывается, что не существует конечного, селекторного ПДИГ со степенью ветвления два и радиусом видимости один, который решает эту же задачу.

Пусть X — *множество запросов*, Y — *множество записей* (объектов поиска), ρ – *бинарное отношение на* $X \times Y$, называемое отношением поиска. Тройку $I = \langle X, V, \rho \rangle$, где V — некоторое подмножество множества Y, в дальнейшем называемой библиотекой, будем называть *задачей информационного поиска* (ЗИП). Будем считать, что ЗИП $I = \langle X, V, \rho \rangle$ содержательно состоит в перечислении для произвольного взятого запроса $x \in X$ всех тех и только тех записей $y \in V$, что $x \rho y$.

В формальном определении понятия ИГ используются 4 множества: множество запросов X, множество записей Y, множество F одноместных предикатов (заданных на множестве X), множество G одноместных переключателей (заданных на множестве X). *Предикат*— это функция, множество значений которой есть $\{0, 1\}$. *Переключатель* — это функция, множество значений которых является начальным отрезком натурального ряда.

Понятие ИГ определяется следующим образом. Берется конечная многополюсная ориентированная сеть. В ней выбирается некоторый полюс, который называется *корнем*. Остальные полюса называются *листьями* и им приписываются записи из Y, причем разным листьям могут быть приписаны одинаковые записи. Некоторые вершины сети (в том числе могут быть и полюса) называются *переключательными* и им приписываются переключатели из G. Ребра, исходящие из каждой из переключательной вершин, нумеруются подряд, начиная с 1, и называются переключательными ребрами. Ребра, не являющиеся переключательными, называются *предикатными* и им приписываются предикаты из множества F. Таким образом, нагруженную многополюсную ориентированную сеть называем ИГ над базовым множеством $\mathcal{F} = \langle F, G \rangle$, где $F = \{f_j, j \in J\}$, $G = \{g_k, k \in K\}$, J и K — множества индексов.

Введем множество функций преобразования индексов $\mathcal{C}$,

$$\mathcal{C} = \{c_m : J^{d_{1,m}} \times K^{d_{2,m}} \times Y^{d_{3,m}} \to J^{a_m} \times K^{b_m} \times Y^{R_m}, \\ m \in M, M \subseteq \mathbb{N};\ d_{1,m},\ d_{2,m},\ d_{3,m} \in \mathbb{N}; \\ a_m, b_m, R_m \in \{0, 1\} \text{ и } a_m + b_m + R_m = 1\}. \quad (1)$$

Введем три счетных множества переменных $\mathcal{P}_J, \mathcal{P}_K, \mathcal{P}_L$:

- $\mathcal{P}_J = \{p_J^j\}$, $j \in \mathbb{N}$, p_J^j принимают значения из J;
- $\mathcal{P}_K = \{p_K^k\}$, $k \in \mathbb{N}$, p_K^k принимают значения из K;
- $\mathcal{P}_L = \{p_L^l\}$, $l \in \mathbb{N}$, p_L^l принимают значения из Y.

Рассмотрим произвольный ИГ U. Он может содержать предикатные, переключательные и листовые вершины. Обозначим $F(U)$ — множество индексов предикатов, $G(U)$ — множество индексов переключателей и $Y(U)$ — множество записей, входящих в ИГ U. Заменим нагрузку всех входящих в U вершин и ребер по следующему правилу.

- Каждый предикат с индексом $j \in F(U)$ заменим на некоторую переменную из $\mathcal{P}_J$. Причем эта замена задается инъективной функцией $\Omega_F : F(U) \to \mathcal{P}_J$. Это означает, что предикаты с различными индексами $j \in F(U)$ заменим на различные переменные из $\mathcal{P}_J$, а с одинаковыми индексами $j \in F(U)$ заменим на одинаковые переменные из $\mathcal{P}_J$.

- Каждый переключатель с индексом $k \in G(U)$ заменим на переменную из $\mathcal{P}_K$. Причем эта замена задается инъективной функцией $\Omega_G : G(U) \to \mathcal{P}_K$.

- Каждую запись (приписанную листовой вершине) $y \in Y$ заменим на переменную из $\mathcal{P}_L$. Причем эта замена задается инъективной функцией $\Omega_Y : Y(U) \to \mathcal{P}_L$.

Рассмотрим множество $\mathcal{P}(U) = \Omega_F(F(U)) \cup \Omega_G(G(U)) \cup \Omega_Y(Y(U))$, то есть $\mathcal{P}(U)$ — множество переменных, которые получились в результате замены нагрузки всех вершин и ребер из U.

Рассмотрим функцию

$$\Omega : F(U) \cup G(U) \cup Y(U) \to \mathcal{P}(U),$$

где $\Omega = \Omega_F$ на множестве $F(U)$ ($\Omega \mid_{F(U)} = \Omega_F$), $\Omega \mid_{G(U)} = \Omega_G$ и $\Omega \mid_{Y(U)} = \Omega_Y$.

Очевидно из определения, что Ω — биективная функция, поэтому существует функция Ω^{-1}. Обозначим $I = \Omega^{-1}$ и будем называть *интерпретацией*, возникшей в результате замены индексов на переменные.

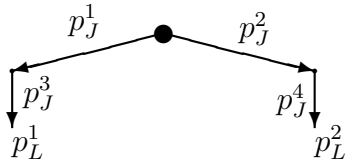


Рис. 1: Пример простого шаблона

После этого сопоставления (замены нагрузки вершин и ребер на переменные в ИГ U) получим нагруженный граф. Выделим в нем множество вершин $V_{\mathcal{T}}$, которые назовем вершинами прикрепления, и занумеруем их числами от 1 до $|V_{\mathcal{T}}|$. Полученный граф назовем *простым шаблоном* $\mathcal{T}$. При этом будем писать $\mathcal{T} = \mathcal{T}(U, I, V_{\mathcal{T}})$, когда хотим подчеркнуть, что $\mathcal{T}$ получен из ИГ U и I, где I — интерпретация, возникшая

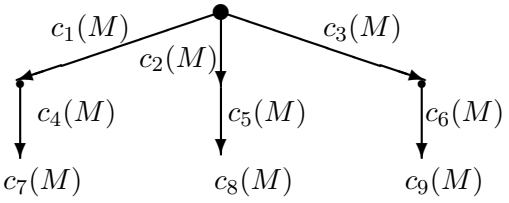


Рис. 2: Пример шаблона $\mathcal{T}_2$

в результате замены индексов на переменные, $V_{\mathcal{T}}$ — упорядоченное множество вершин прикрепления. Пример простого шаблона изображен на рисунке 1.1 (жирным кружком обозначена единственная вершина прикрепления).

Рассмотрим ИГ U', выделим в нем множество вершин $V_{U'}$, которые назовем вершинами прикрепления, и занумеруем их числами от 1 до $|V_{U'}|$ (вершины прикрепления образуют упорядоченное множество). Будем говорить, что ИГ U' и простой шаблон $\mathcal{T}$ *согласованы*, если выполнены следующие условия:

- U' и $\mathcal{T}$ совпадают как графы, и если в ИГ U' встречаются одинаковые индексы предикатов и записи, то в соответствующих местах шаблона $\mathcal{T}$ находятся одинаковые переменные из $\mathcal{P}_J$ и $\mathcal{P}_L$;

- i—ая вершина прикрепления ИГ U' совпадает с i—ой вершиной прикрепления простого шаблона $\mathcal{T}$, $i = 1, \ldots, |V_{U'}|$ и $|V_{U'}| = |V_{\mathcal{T}}|$.

То есть $\mathcal{T} = \mathcal{T}(U', I, V_{\mathcal{T}})$ для некоторой интерпретации I, и будем писать $\mathcal{T} = \mathcal{T}(U', I, V_{U'})$, когда хотим подчеркнуть, что ИГ U' и простой шаблон $\mathcal{T}$ согласованы.

Рассмотрим простой шаблон $\mathcal{T}_1$ и заменим в нем каждую переменную, на формулу над множество функций $\mathcal{C}$ и каким-либо множеством переменных $M \subseteq \mathcal{P}_J \cup \mathcal{P}_L$. В результате, полученный граф назовем *шаблоном* $\mathcal{T}_2$. При этом будем писать $\mathcal{T}_2 = \mathcal{T}_2(\mathcal{T}_1, M, V_{\mathcal{T}_1})$, когда хотим подчеркнуть, что $\mathcal{T}_2$ получен из простого шаблона $\mathcal{T}_1$, множества переменных M и упорядоченного множеств вершин прикрепления $V_{\mathcal{T}_1}$.

На рисунках запись $R_i(M)$ будет означать формулу над над множество функций $\mathcal{C}$ и множеством переменных из M.

Пример шаблона приведен на рисунке 1.2 (жирным кружком обозначена единственная вершина прикрепления).

Рассмотрим простой шаблон $\mathcal{T}_1 = \mathcal{T}_1(U, I, V_{\mathcal{T}_1})$. Обозначим множество входящих в него переменных $M(\mathcal{T}_1)$. Пусть $p_L^0 \in \mathcal{P}_L \setminus M(\mathcal{T}_1)$.

Преобразованием R назовем тройку

$$R = (\mathcal{T}_1(U, I, V_{\mathcal{T}_1}), \mathcal{T}_2(\mathcal{T}, M(\mathcal{T}_1) \cup \{p_L^0\}, V_{\mathcal{T}}), \xi),$$

где $\mathcal{T}_1$ — простой шаблон, $\mathcal{T}_2$ — шаблон (полученный из простого шаблона $\mathcal{T}$) в формулах которого встречаются только переменные из множества $M(\mathcal{T}_1) \cup p_L^0$, $V_{\mathcal{T}}$ упорядоченное множество вершин прикрепления простого шаблона $\mathcal{T}$, ξ — биективная

функция сопоставления вершин прикрепления ($\xi : V_{\mathcal{T}_1} \to V_{\mathcal{T}}$). Левой частью преобразования R будем называть простой шаблон $\mathcal{T}_1$.

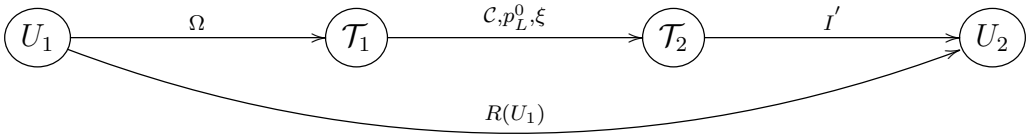


Рис. 3: Преобразование ИГ U_1 в ИГ U_2

Пусть ИГ U_1 и простой шаблон $\mathcal{T}_1$ согласованы, то есть $\mathcal{T}_1 = \mathcal{T}_1(U_1, I, V_{U_1})$ для некоторой интерпретации I, и пусть $I^{'}$ – интерпретация множества переменных $M(\mathcal{T}_1) \cup p_L^0$, причем $I^{'} = I$ на множестве переменных $M(\mathcal{T}_1)$. Тогда применением преобразования $R = (\mathcal{T}_1(U, I, V_{U_1}), \mathcal{T}_2(\mathcal{T}, M(\mathcal{T}_1) \cup p_L^0, V_{\mathcal{T}}), \xi)$ к ИГ U_1 назовем ИГ U_2, получающийся из шаблона $\mathcal{T}_2(\mathcal{T}, M(\mathcal{T}_1) \cup p_L^0, V_{\mathcal{T}})$ подстановкой вместо каждой формулы значения данной формулы в интерпретации $I^{'}$. Заметим, что упорядоченное множество вершин прикрепления V_{U_2} в ИГ U_2 однозначно соответствует упорядоченному множеству вершин прикрепления V_{U_1} в ИГ U_1, так как ξ — биективная функция, и простой шаблон $\mathcal{T}_1$ согласован с ИГ U_1.

Результат применения преобразования R к ИГ U_1 будем обозначать $R(U_1) = U_2$. Преобразование ИГ U_1 в ИГ U_2 изображено на рисунке 1.3.

Следующие понятия будут введены для определения *кода информационного графа на запросе*. Код информационного графа на запросе будет использоваться для описания передвижения автомата по вершинам ИГ.

Рассмотрим ИГ U, $N(U)$ — множество входящих в него вершин. Пусть $\beta, \beta^{'} \in N(U)$, тогда путем $\pi(\beta, \beta^{'})$ назовем последовательность вершин и ребер ИГ U, которая начинается в вершине β, заканчивается в вершине $\beta^{'}$, и в этой последовательности два любых соседних элемента инциденты. Длиной пути $l(\pi(\beta, \beta^{'}))$ назовем количество ребер, участвующих в нем. Пусть $\Pi(\beta, \beta^{'})$ — множество всех путей из β в $\beta^{'}$. Тогда расстоянием между вершинами $\beta, \beta^{'} \in N(U)$ назовем минимальную из всех путей длину $l(\pi(\beta, \beta^{'})), \pi(\beta, \beta^{'}) \in \Pi(\beta, \beta^{'})$, и обозначим $(\beta, \beta^{'}) = \min\{l(\pi(\beta, \beta^{'})) : \pi(\beta, \beta^{'}) \in \Pi(\beta, \beta^{'})\}$. Эксцентриситетом вершины $\beta \in N(U)$ назовем число $\varepsilon(\beta) = \max\limits_{\beta^{'} \in N(U)} (\beta, \beta^{'})$. *Радиусом* ИГ U назовем число $r(U) = \min\limits_{\beta \in N(U)} \varepsilon(\beta)$.

Через $\mathcal{G}(N, R)$, $N, R \in \mathbb{N}$ обозначим класс ИГ радиуса не больше R таких, что количество ребер инцидентных любой вершине графа не превосходит N.

Рассмотрим ИГ U, $E(U)$ — множество входящих в него ребер. Ребро инцидентное вершинам $\beta_1, \beta_2 \in N(U)$ будем обозначать $e(\beta_1, \beta_2)$.

Рассмотрим ИГ U_2 и его подграф (подмножество ребер и инцидентных им вершин) U_1 (пишем $U_1 \subseteq U_2$). Множество $\Sigma(U_1, U_2) = \{\beta : \beta \in N(U_1)$ и $\exists \beta_1 \in N(U_2 \backslash U_1), \exists\, \beta_2 \in N(U_1),\ e(\beta_1, \beta) \in E(U_2),\ e(\beta_2, \beta) \in E(U_1)\}$ назовем множеством *граничных вершин информационных графов* U_1 *и* U_2.

Пусть количество различных типов предикатов и переключателей конечно и равно K_T. Сопоставим им в биективном соответствие натуральные числа от 1 до K_T. Пусть $U_1, U_2 \in \mathcal{G}(N, \infty)$ и $U_1 \subseteq U_2$ ($N < \infty$). Назовем *кодом вершины на запросе* $x \in X$ *относительно пары* (U_1, U_2) пятерку $(k_1, k_2, k_3, k_4, k_5)$, где $k_1 = 0$, если вершина предикатная; $k_1 = 1$, если она переключательная; $k_2 = 0$, если вершина корень; $k_2 = 1$, если вершина листовая; $k_2 = 2$ в остальных случаях; $k_3 \in \{1, \ldots, N+1\}$ и в случае переключательной вершины принимает значение соответствующего ей переключателя на запросе $x \in X$, если это значение принадлежит $\{1,\ldots,\mathrm{N}\}$, и $k_3 = N+1$ во всех остальных случаях; $k_4 \in \{0, 1\}$, $k_4 = 1$, если вершина принадлежит множеству граничных вершин $\Sigma(U_1, U_2)$ и $k_4 = 0$, если не принадлежит; $k_5 \in \{0, 1, \ldots, K_T\}$ принимает значение 0 если вершина предикатная или номер типа соответствующего ей переключателя иначе.

Кодом ребра на запросе $x \in X$ *типа поиск относительно пары* (U_1, U_2) назовем тройку (k_1^r, k_2^r, k_3^r), где $k_1^r = 0$, если ребро предикатное; $k_1^r = 1$, если ребро переключательное; $k_2^r \in \{0, 1, \ldots, N\}$; $k_3^r \in \{0, 1, \ldots, K_T\}$ принимает значение 0 если ребро переключательное или тип предиката соответствующее этому ребру иначе. У предикатного ребра k_2^r равен значению переключателя на запросе $x \in X$. Таким образом предикатным ребрам приписан код $(0, 0, t)$ или $(0, 1, t)$, где t - тип предиката, а каждому переключательному ребру приписан код $(1, i, 0)$, где $i \in \{1, \ldots, N\}$ номер переключательного ребра.

В общем случае множества запросов X и множество записей Y могут быть разные. Поэтому для того, чтобы определить код вершин и ребер ИГ на запросах типа вставка и удаление введем дополнительную функцию $\eta : Y \to X^l$, $l \in \mathbb{N}$, $l < \infty$, $\eta = (\eta_1, \ldots, \eta_l)$.

Кодом вершины на запросе $y \in Y$ *типа вставка, удаление относительно пары* (U_1, U_2) будет l пятерок кодов вершин, которые соответствуют запросам $\eta_1(y), \ldots, \eta_l(y)$ типа поиск. *Кодом ребра на запросе* $x \in Y$ *типа вставка, удаление относительно пары* (U_1, U_2) будет значение l троек кодов ребер, которые соответствуют запросам $\eta_1(y), \ldots, \eta_l(y)$ типа поиск.

Рассмотрим ИГ U, в нем вершины и ребра имеют нагрузку. Граф $K(U)$ полученный из ИГ U удалением нагрузок вершин и ребер, назовем *каркасом* ИГ U.

Кодом ИГ U_1 *относительно ИГ* U_2 *на запросе*, назовем нагруженный каркас $K(U_1)$, где нагрузка каждой вершины $K(U_1)$ это код данной вершин на данном запросе относительно пары (U_1, U_2), а на нагрузка каждого ребра $K(U_1)$ это код данного ребра на данном запросе относительно пары (U_1, U_2). Через $\mathcal{K}(N, R)$ обозначим множество кодов ИГ U_1 относительно ИГ U_2, где U_1 пробегает класс ИГ $\mathcal{G}(N, R)$, U_2 пробегает класс ИГ $\mathcal{G}(N, R+1)$ и $U_1 \subseteq U_2$. Понятно, что $|\mathcal{K}(N, R)| < \infty$.

Пусть $N, R \in \mathbb{N}$, $\mathcal{P}$ — некоторое конечное множество преобразований, шаблоны которых порождены ИГ из $\mathcal{G}(N, R)$, и $P_0 \in \mathcal{P}$, где P_0 — тождественное преобразование такое, что $P_0(U) = U$ для произвольного ИГ U из $\mathcal{G}(N, R)$.

Пусть $N(U_1)$ множество вершин ИГ U_1. Рассмотрим множество вершин $\mathfrak{B} \subseteq$

$N(U_1)$ и обозначим через $U(\mathfrak{B}, U_1)$ — ИГ, полученный из ИГ U_1 как подграф на вершинах $\mathfrak{B}$. Под подграфом на вершинах $\mathfrak{B}$ здесь понимается множество вершин $\mathfrak{B}$ и множество всех инцидентных им ребер.

Пусть $\mathcal{A}$ — конечный автомат. Будем считать, что автомат $\mathcal{A}$ перемещается по вершинам ИГ $U_2 \in \mathcal{G}(N, \infty)$ и в каждый момент времени обозревает окрестность U_1 текущей вершины радиуса R, т.е. будем считать, что входным символом автомата $\mathcal{A}$ является код обозреваемой окрестности относительно U_2. Понятно, что эти коды будут принадлежать $\mathcal{K}(N, R)$ и значит входным алфавитом автомата $\mathcal{A}$ является конечное множество $\mathcal{K}(N, R)$. Результатом этого автомата $\mathcal{A}$ на обозреваемой окрестности текущей вершины будет некоторое преобразование этой окрестности и перемещение в некоторую вершину преобразованной окрестности. Тем самым выходным символом автомата $\mathcal{A}$ будет пятерка $b = (\mathfrak{B}, \pi, R, \beta, e)$, где

- $\mathfrak{B} \subseteq N(U_1)$ определяет множество вершин обозреваемой окрестности к которому будет применено преобразование;
- π — функция нумерации граничных вершин $U(\mathfrak{B}, U_1)$ и U_2 (содержательно это множество граничных вершин $U(\mathfrak{B}, U_1)$ и U_1 объединенное со множеством вершин в коде которых $k_2 = 1$ и одновременно принадлежащих $U(\mathfrak{B}, U_1)$). Пронумерованные граничные вершины назовем вершинами прикрепления;
- R — преобразование из конечного множества $\mathcal{P}$ применяемое к ИГ $U(\mathfrak{B}, U_1)$, причем ИГ $U(\mathfrak{B}, U_1)$ с заданной функцией π нумерацией вершин прикрепления согласован с левой частью преобразования R;
- $\beta \in N(R(U(\mathfrak{B}, U_1))) \cup \{U_1 \setminus U(\mathfrak{B}, U_1)\}$ — следующая текущая вершина автомата $\mathcal{A}$;
- $e \in \{0, 1, 2\}$ отвечает за продолжение и выдачу ответа автоматом $\mathcal{A}$. Автомат еще не нашел ответ и продолжает функционирование ($e = 0$), либо завершает функционирование и возвращает информацию о том, что искомая запись найдена ($e = 1$), либо завершает функционирование с информацией, что искомой записи нет ($e = 2$).

Множество всех таких выходных символов b образует выходной алфавит B автомата $\mathcal{A}$.

Пусть U — ИГ над базовым множеством $\langle F, G\rangle$ из класса $\mathcal{G}(N, \infty)$, тогда пару $(\mathcal{A}, U)$ назовем динамическим информационным графом (ДИГ) типа (N, R) над $\mathcal{F}{=}\langle F, G, \mathcal{P}, \eta\rangle$ и обозначим $\mathcal{D} = (\mathcal{A}, U)$.

Опишем теперь функционирование ДИГ. Определим функционирование ДИГ $\mathcal{D} = (\mathcal{A}, U)$ типа (N, R) над $\mathcal{F}{=}\langle F, G, \mathcal{P}, \eta\rangle$ на запросе. В начальный момент текущей вершиной объявляется корень ИГ U. Рассматривается ИГ U_1 как подграф $U(i)$, с центром в текущей вершине и радиуса R. На вход автомата $\mathcal{A}$ подается код ИГ U_1

относительно ИГ U на запросе. Пусть $b = (\mathfrak{B}, \pi, R, \beta, e) \in B$ выходная буква автомата $\mathcal{A}$, тогда ИГ U меняется по следующему правилу. В ИГ $U(i)$ ИГ $U(\mathfrak{B}, U_1)$ заменяется на ИГ $U^{'}$, полученной в результате применения преобразования R к ИГ $U(\mathfrak{B}, U_1)$ $(R(U(\mathfrak{B}, U_1)) = U^{'})$ и "прикрепляется" к ИГ $U(i)$ посредством функции π и функции сопоставления вершин прикрепления ξ из преобразования R. После этого текущее положение автомата $\mathcal{A}$ меняется на β и в зависимости от значения последней компоненты выходной буквы b ДИГ $\mathcal{D}_M$ либо продолжает функционирование $(e = 0)$, либо завершает функционирование и возвращает информацию о том, что искомая запись найдена (вставлена или удалена)$(e = 1)$, либо завершает функционирование с информацией, что записи нет $(e = 2)$.

Сложность и объем ДИГ $\mathcal{D} = (\mathcal{A}, U)$ типа (N, R) над базовым множеством $\mathcal{F}{=}\langle F, G, \mathcal{P}, \eta\rangle$ определяется для фиксированного ИГ U. Объемом ДИГ $\mathcal{D}$ будем называть объем ИГ U (количество ребер в графе) и обозначим $Q(\mathcal{D})$.

Определим сложность ДИГ $\mathcal{D}$ на запросе $x \in X$ типа вставка (удаление, поиск) и обозначим их соответственно $T_i(\mathcal{D}, x)$ $(T_d(\mathcal{D}, x)$, $T_s(\mathcal{D}, x))$. Для этого введем сложность выходного действия автомата $\mathcal{A}$.

Пусть $Z : \mathcal{P} \to \mathbb{N}$, входным параметром, которой является $R \in \mathcal{P}$, а значением сложность выполненного преобразования. Другими словами с помощью L вводится сложность каждого элемента из множества преобразований $\mathcal{P}$.

Последовательность преобразований, выполненных автоматом $\mathcal{A}$, в ходе функционирования ДИГ $\mathcal{D}$ на запросе $x \in X$ типа вставка обозначим $f_{\mathcal{A}}(x, 1)$ (типа удаление $f_{\mathcal{A}}(x, 2)$, типа поиск $f_{\mathcal{A}}(x, 3)$).

Сложностью ДИГ $\mathcal{D}$ на запросе $x \in X$ типа вставка (удаление) назовем

- $T_i(\mathcal{D}, x) = \sum\limits_{R \in f_{\mathcal{A}}(x,1)} Z(R)$ ($T_d(\mathcal{D}, x) = \sum\limits_{R \in f_{\mathcal{A}}(x,2)} Z(R)$, $T_s(\mathcal{D}, x) = \sum\limits_{R \in f_{\mathcal{A}}(x,3)} Z(R)$).

Сложностью ДИГ $\mathcal{D}$ назовем $\max\{T_i(\mathcal{D}), T_d(\mathcal{D}), T_s(\mathcal{D})\}$ и обозначим $T(\mathcal{D})$.

Перейдем к постановке задачи и ее решению с помощью ДИГ. Пусть $V \subset Y \subseteq \mathbb{R}$, $|V| < \infty$, $X = Y$. Скажем, что ДИГ $\mathcal{D}$ решает ЗПИО, если ответ на произвольный запрос $x \in X$ типа поиска равен $\{x\}$, если $x \in V$, и пуст в противном случае, и если на произвольном запросе типа вставки (удаления) записи $y \in Y$ результирующий ДИГ $\mathcal{D}$ выдает ответ $\{x\}$ на произвольный запрос $x \in X$ типа поиск, если $x \in V \cup \{y\}$ $(x \in V \setminus \{y\})$, и пуст в противном случае.

Пусть $J = \{(a) : a \in \mathbb{R}\}$, $K = \{(a, b) : a, b \in \mathbb{R}\}$.

$$G_{id} = \left\{ g_{a,b}(x) = \begin{cases} 1, & \text{если } x \le a; \\ 2, & \text{если, } a < x \le b; \\ 3, & \text{иначе.} \end{cases}, (a,b) \in K \right\}; \quad (2)$$

$$F_{id} = \left\{ f_{=,a}(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{если } x \ne a. \end{cases}, a \in J \right\}; \quad (3)$$

$$\mathcal{F}^{id} = \langle F_{id} \cup G_{id} \rangle. \quad (4)$$

,

В качестве функции η_{id} всегда рассматриваем тождественную функцию $\eta_{id} : Y \to X$ (в ЗПИО $Y = X$), поэтому ее будем опускать в формулировках в дальнейшем.

Теорема 1. *Существует базовое множество преобразований $\mathcal{R}_{id}$ и существует ДИГ $\mathcal{D}_{id} = (\mathcal{A}_{id}, U_{id})$ типа $(5, 3)$ над базовым множеством $\langle F_{id}, G_{id}, \mathcal{R}_{id}, \eta_{id} \rangle$, который решает ЗПИО, причем*

$$T(D_{id}) \le (\lambda_{max} + 1)([\log_2 |V|] + 1);$$
$$Q(D_{id}) \le 3|V|,$$

где $\lambda_{max} = \max\{Z(R) : R \in \mathcal{P}_{id}\}$.

В конце первой главы модель обобщатся для решения задач с использованием параллельных процессов. Обобщенная модель носит название *потоковый динамический информационный граф* (ПДИГ).

Обобщим понятие запроса к базе данных на случай вставки, удаления и пустого действия (запрос не требует действий для него). Для этого рассмотрим алфавит $A = \{\text{П, В, У, } \Lambda\}$. Обобщенным запросом назовем пару $(a, x), (a, x) \in \tilde{X}$, где $\tilde{X} = A \times X$. В данной работе рассматривается задача поиска идентичного объекта поэтому множество запросов и множество записей одинаково и равно X. При этом второй аргумент обобщенного запроса (буква из алфавита A) будет означать необходимое действие, которое нужно осуществить для запроса: П — поиск, В — вставка, У — удаление, Λ — ничего не делать. Далее везде под словом запрос будем понимать обобщенный запрос.

Потоком запросов H, $H : \mathbb{N} \to \tilde{X}$, назовем последовательность запросов, поступающих к базе данных через равные промежутки времени — такты. Другими словами поток запросов H означает, что к базе данных в i-ый такт времени будет подан запрос $H(i)$. Обозначим множество всех потоков через $\mathcal{H}$.

Рассмотрим функцию W, $W : \mathcal{H} \to A^\infty$ (через A^∞ обозначено множество всех сверхслов в алфавите A) сопоставляющую потоку запросов H сверхслово $W(H)$ из

алфавита A, причем i-ая буква сверхслова $W(H)$ (обозначим ее через $W(H)(i)$) такова, что $H(i) = (W(H)(i), x_i)$.

Для обработки потоков запросов введем понятие *потоковый динамический информационный граф* (ПДИГ) и обозначим его $\mathcal{D}_\Pi = (\mathcal{A}, U)$. Пусть дано бесконечное множество копий автомата $\mathcal{A}$. Будем считать, что несколько копий автомата $\mathcal{A}$ могут одновременно обслуживать несколько запросов. Например пусть имеется поток запросов на поиск $H = (\Pi, x_1), (\Pi, x_2), (\Pi, x_3), \ldots$, тогда при наличии 3 и более копий автомата $\mathcal{A}$ ПДИГ сможет параллельно обслужить три запроса x_1, x_2, x_3 на поиск, не дожидаясь завершения каждого ДИГ в отдельности. Если в потоке запросов H встречаются запросы на изменение базы данных (вставка и удаление), то при параллельной обработки этих запросов возможны конфликты преобразований (ИГ общий для всех автоматов).

Определим *функционирование ПДИГ* $\mathcal{D}_\Pi, \mathcal{D}_\Pi = (\mathcal{A}, U)$, *типа* (N, R) *над* $\mathcal{F}, \mathcal{F} = \langle F, \emptyset, \mathcal{P}, \eta_{id} \rangle$, *для потока запросов* H *из* $\mathcal{H}$. Считаем, что время применения любого преобразования R из $\mathcal{P}$, ровно 1 такт. Тем самым автомат $\mathcal{A}$ за один такт делает ровно одно преобразование над ИГ U. Заметим, что мы всегда можем выбрать единицу измерения времени (такт), что это будет выполнено, например, как максимум из времен всех преобразований. Так же мы предполагаем, что такт запроса совпадает с тактом работы автомата.

После каждого такта работы ПДИГ — ИГ U может изменяться, поэтому будем рассматривать $U = U(t)$ $(t \in \mathbb{N} \cup \{0\})$. В начальный момент функционирования $U = U(0)$.

Рассмотрим запрос $H(i), H(i) = (a_i, x_i)$, который поступил для обработки к ПДИГ $\mathcal{D}_\Pi, \mathcal{D}_\Pi = (\mathcal{A}, U)$, $U = U(i)$ в i-ый такт, $i \geq 1$. ПДИГ действует в зависимости от типа запроса a_i. Если $a_i = \Lambda$ (ничего не делать), то ПДИГ ничего не делает для этого запроса и $U(i+1) = U(i)$, иначе ПДИГ функционирует как ДИГ для данного запроса.

Теперь определим понятие *сложности* ПДИГ. Для любого i из $\mathbb{N}$ через $T(\mathcal{D}_\Pi, H(i))$ обозначим количество тактов необходимое для обработки запроса $H(i)$ динамическим информационным графом $\mathcal{D}_\Pi$.

В второй главе доказываются верхние оценки на ПДИГ. Сначала опишем формальную постановку задачи. Пусть H — поток запросов, $H : \mathbb{N} \to X$, где $X, X = \{\Pi, \text{В}, \text{У}, \Lambda\} \times \mathbb{N}$ — множество запросов. Буква в запросе означает его действие: поиск (П), вставка (В), удаление (У) или пустой запрос (Λ).

Множество всех потоков запросов обозначим через $\mathcal{H}$. Рассмотрим функцию множества записей $V : \{\mathbb{N} \cup \{0\}\} \times \mathcal{H} \to 2^X$, которая удовлетворяет следующим условиям:

$$V(i,H) = \begin{cases} \emptyset, & \text{если } i = 0; \\ V(i-1,H), & \text{если } i > 0 \text{ и } H(i) \text{ запрос на поиск или пустой запрос;} \\ V(i-1,H) \cup \{x\}, & \text{если } i > 0 \text{ и } H(i) = (\text{В}, x); \\ V(i-1,H) \setminus \{x\}, & \text{если } i > 0 \text{ и } H(i) = (\text{У}, x). \end{cases}$$

Функция V составляет каждому такту i и потоку запросов H — множество записей, которые образуют базу данных после завершения функционирования ПДИГ для всех запросов до такта i включительно, если обработка любого запроса происходила бы мгновенно (за 1 такт).

Пусть дана функция $L : \mathbb{N} \cup \{0\} \to \mathbb{N}$. Будем говорить, что ПДИГ решает ДЗПИО (*логическую ДЗПИО*) со сложностью L, если для любого потока H и любого такта i выполняются следующие условия

- если $H(i) = (\text{П}, x)$, то результатом функционирования ПДИГ должен быть $\{x\}$ (да), если $x \in V(i,H)$ и пустое множество (нет) в противном случае. При этом количество тактов, необходимое на выдачу ответа, должно не превосходить $L(|V(i,H)|)$;

- если $H(i) = (\text{В}, x)$, то для любого запроса на поиск (П, z), поступившего в такт $i+1$, результат функционирования ПДИГ должен быть $\{z\}$(да), если $z \in V(i+1,H) = V(i,H) \cup \{x\}$, и пустое множество(нет) в противном случае;

- если $H(i) = (\text{У}, x)$, то для любого запроса на поиск (П,z), поступившего в такт $i+1$, результат функционирования ПДИГ должен быть $\{z\}$(да), если $z \in V(i+1,H) = V(i,H) \setminus \{x\}$, и пустое множество(нет) в противном случае.

Введем еще два понятие как *конечный* и *селекторный ПДИГ*. Будем говорить, что *ПДИГ конечный*, если для произвольного потока запросов H и произвольного такта времени i существует такая константа C,$C < \infty$, что для потока запросов $H^{'}, H^{'}(1) = H(1), \ldots H^{'}(i) = H(i), H^{'}(i+1) = \Lambda, \ldots, H^{'}(i+C) = \Lambda$, автомат обслуживающий запрос $H(i)$ завершит свое функционирование.

Неформально говоря, ПДИГ будем называть конечным, если автомат для любого потока запроса и любого такта, функционирует конечное время. ПДИГ, который не является конечным, будем называть бесконечным.

Введем понятие *селекторный ПДИГ*. Рассмотрим произвольную функцию $\phi \in \{\phi_1, \ldots, \phi_n\}$, которую может применить автомат в своем преобразовании. Автомат, зная тип предиката, знает также, сколько у него аргументов. Функция ϕ на вход получает аргументы и типы предикатов , типы вершин, а так же запись, которую обслуживает автомат. После этого эта функция может создать новый предикат или запись. Естественно, функция ϕ может создать только предикат, принадлежащий

множеству предикатов F, над которым рассматривается ИГ. Оперируя этими функциями, автомат может в новой окрестности создавать новые предикаты из F.

Преобразование будем называть селекторным, если оно не может создать новый аргумент, кроме тех, что есть во входной окрестности или запроса, который обслуживает автомат.

Пусть

$$f_a(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{иначе} \end{cases},$$

$$F_{list} = \{f_a(x) : a \in \mathbb{R}\}.$$

Через $]x[$ будем обозначать целую часть сверху от x (наименьшее целое число, которое не меньше x). Справедлива следующая теорема.

Теорема 2. *Существует множество преобразований $\mathcal{R}$ и существует конечный, селекторный ПДИГ типа (2,2) над базовым множеством $\langle F_{list}, \mathcal{R}\rangle$, который решает ДЗПИО для любого потока запросов H из $\mathcal{H}$, со сложностью $L, L(n) =]n/2[+1$.*

Пусть

$$f_a(x) = f_a^h(x) = \begin{cases} 1, & \text{если } x \le a; \\ 0, & \text{иначе} \end{cases}; f_{=,a}(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{иначе} \end{cases};$$

$$f_a^{\Pi}(x) = f_a^{\text{В}}(x) = f_a^{\text{У}}(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{иначе} \end{cases}; f^h(x) \equiv 0;$$

$$F_{log} = \left\{f_a(x), f_a^h(x), f_{=,a}(x), f_a^{\Pi}, f_a^{\text{В}}, f_a^{\text{У}} : a \in \mathbb{R}\right\} \cup \{f^h\}.$$

Справедлива следующая теорема.

Теорема 3. *Существует множество преобразований $\mathcal{R}$ и существует конечный, селекторный ПДИГ типа (8,4) над базовым множеством $\langle F_{log}, \mathcal{R}\rangle$, который решает ДЗПИО для любого потока запросов $H, H \in \mathcal{H}$, со сложностью $L, L(n) = \left]\frac{\log_2 \max(n,2)}{3}\right[+ 1$.*

Рассмотрим множество предикатов $F(T) = \{f^0, f^1, f_a^t(x) | t \in T, a \in \mathbb{N}\}$, где $|T| < \infty$ и

$\forall t \in T, a \in \mathbb{N}$:

$$f_a^t(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{иначе} \end{cases}, f^0(x) \equiv 0, f^1(x) \equiv 1.$$

Пусть

$$T_0 = \{\text{П}, \text{В}_1, \text{В}_2, \text{У}\},$$

то есть $F(T_0) = \{f^0, f^1, f_a^{\text{П}}, f_a^{\text{В}_1}, f_a^{\text{В}_2}, f_a^{\text{У}} : a \in \mathbb{N}\}$.

Справедлива следующая теорема.

Теорема 4. *Существует множество преобразований $\mathcal{R}$ и существует бесконечный, селекторный ПДИГ типа (1,1) над базовым множеством $\langle F(T_0), \mathcal{R}\rangle$, который решает ДЗПИО для любого потока запросов H из $\mathcal{H}$ со сложность $L, L \equiv 3$.*

Пусть

$$T_1 = \{\text{да}, \text{нет}\},$$

то есть $F(T_1) = \{f^0, f^1, f_a^{\text{да}}, f_a^{\text{нет}} : a \in \mathbb{N}\}$.

Теорема 5. *Существует множество преобразований $\mathcal{R}$ и существует конечный, не селекторный ПДИГ типа (1,1) над базовым множеством $\langle F(T_1), \mathcal{R}\rangle$, решающий логическую ДЗПИО со сложностью $L, L \equiv 2$.*

Как будет доказано ниже, не существует конечного ПДИГ типа (1,1), решающего ДЗПИО. В предыдущей теореме конечный ПДИГ типа (1,1) решает логическую ДЗПИО.

Следующая теорема утверждает, что существует конечный не селекторный ПДИГ типа (2,1), который решает логическую ДЗПИО.

Пусть

$$T = \{\text{а}, \text{н}\} \times \{-2, -1, 0, 1, 2\} \times \{\text{П}_{\leftarrow}, \text{П}_{\rightarrow}, \text{В}_{\leftarrow}, \text{В}_{\rightarrow}, \text{У}_{\leftarrow}, \text{У}_{\rightarrow}, \text{н}\},$$

$$F_2(T) = \{f_{l,r,a}^t(x) : t \in T, l, r, a, x \in \mathbb{N}\}, \text{ где}$$

$f_{l,r,a}^t(x) = 1$, если $x = a$ и 0 иначе.

Теорема 6. *Существует множество преобразований $\mathcal{R}$ и существует конечный ПДИГ типа (2,1) над базовым множеством $\langle F_2(T), \mathcal{R}\rangle$, решающий ДЗПИО со сложностью $L, L(n) =]n/2[+5$.*

Как видно из предыдущих теорем ПДИГ достаточно мощный аппарат для решения ДЗПИО. Поэтому найти ограничения на ПДИГ для доказательства не существования алгоритма — еше одна интересная задача для математиков. В частности, предыдущие теоремы 4 — 6 получились в результате поисков ограничений на ПДИГ.

В третьей главе доказываются нижние оценки для ПДИГ.

Теорема 7. *Для любого натурального R, не существует конечного ПДИГ типа $(1, R)$, решающего ДЗПИО для любого потока запросов.*

Теорема 8. *Для любой функции $L, L : \mathbb{N} \to \mathbb{N}$ и для любого множества $T, |T| < \infty$ не существует конечного, селекторного ПДИГ типа (2, 1) над множеством $F(T)$, который решает логическую ДЗПИО со сложностью L.*

Благодарности

Автор выражает глубокую благодарность своему научному руководителю — доктору физико-математических наук, профессору Эльяру Эльдаровичу Гасанову за постановку задачи, постоянное внимание к работе и всестороннюю поддержку, а также заведующему кафедрой академику Валерию Борисовичу Кудрявцеву и всему коллективу кафедры математической теории интеллектуальных систем за доброжелательную и творческую атмосферу.

Глава 1

Математическая модель динамических баз данных

1 Основные понятия.

Пусть X — *множество запросов*, Y — *множество записей* (объектов поиска), ρ – *бинарное отношение на* $X \times Y$, называемое отношением поиска. Тройку $I = \langle X, V, \rho \rangle$, где V — некоторое подмножество множества Y, в дальнейшем называемой библиотекой, будем называть *задачей информационного поиска* (ЗИП). Будем считать, что ЗИП $I = \langle X, V, \rho \rangle$ содержательно состоит в перечислении для произвольного взятого запроса $x \in X$ всех тех и только тех записей $y \in V$, что $x \rho y$.

В формальном определении понятия ИГ используются 4 множества: множество запросов X, множество записей Y, множество F одноместных предикатов (заданных на множестве X), множество G одноместных переключателей (заданных на множестве X). *Предикат*— это функция, множество значений которой есть $\{0, 1\}$. *Переключатель* — это функция, множество значений которых является начальным отрезком натурального ряда.

Понятие ИГ определяется следующим образом. Берется конечная многополюсная ориентированная сеть. В ней выбирается некоторый полюс, который называется *корнем*. Остальные полюса называются *листьями* и им приписываются записи из Y, причем разным листьям могут быть приписаны одинаковые записи. Некоторые вершины сети (в том числе могут быть и полюса) называются *переключательными* и им приписываются переключатели из G. Ребра, исходящие из каждой из переключательной вершин, нумеруются подряд, начиная с 1, и называются переключательными ребрами. Ребра, не являющиеся переключательными, называются *предикатными* и им приписываются предикаты из множества F. Таким образом, нагруженную многополюсную ориентированную сеть называем ИГ над базовым множеством $\mathcal{F} = \langle F, G \rangle$, где $F = \{f_j, j \in J\}$, $G = \{g_k, k \in K\}$, J и K — множества индексов. Более подробное описание ИГ приведено в [1].

Введем множество функций преобразования индексов $\mathcal{C}$,

$$\mathcal{C} = \{c_m : J^{d_{1,m}} \times K^{d_{2,m}} \times Y^{d_{3,m}} \to J^{a_m} \times K^{b_m} \times Y^{R_m}, \\ m \in M, M \subseteq \mathbb{N};\ d_{1,m},\ d_{2,m},\ d_{3,m} \in \mathbb{N}; \\ a_m, b_m, R_m \in \{0,1\} \text{ и } a_m + b_m + R_m = 1\}. \quad (1.1)$$

Введем три счетных множества переменных $\mathcal{P}_J, \mathcal{P}_K, \mathcal{P}_L$:

- $\mathcal{P}_J = \{p_J^j\}$, $j \in \mathbb{N}$, p_J^j принимают значения из J;
- $\mathcal{P}_K = \{p_K^k\}$, $k \in \mathbb{N}$, p_K^k принимают значения из K;
- $\mathcal{P}_L = \{p_L^l\}$, $l \in \mathbb{N}$, p_L^l принимают значения из Y.

Рассмотрим произвольный ИГ U. Он может содержать предикатные, переключательные и листовые вершины. Обозначим $F(U)$ — множество индексов предикатов, $G(U)$ — множество индексов переключателей и $Y(U)$ — множество записей, входящих в ИГ U. Заменим нагрузку всех входящих в U вершин и ребер по следующему правилу.

- Каждый предикат с индексом $j \in F(U)$ заменим на некоторую переменную из $\mathcal{P}_J$. Причем эта замена задается инъективной функцией $\Omega_F : F(U) \to \mathcal{P}_J$. Это означает, что предикаты с различными индексами $j \in F(U)$ заменим на различные переменные из $\mathcal{P}_J$, а с одинаковыми индексами $j \in F(U)$ заменим на одинаковые переменные из $\mathcal{P}_J$.
- Каждый переключатель с индексом $k \in G(U)$ заменим на переменную из $\mathcal{P}_K$. Причем эта замена задается инъективной функцией $\Omega_G : G(U) \to \mathcal{P}_K$.
- Каждую запись (приписанную листовой вершине) $y \in Y$ заменим на переменную из $\mathcal{P}_L$. Причем эта замена задается инъективной функцией $\Omega_Y : Y(U) \to \mathcal{P}_L$.

Рассмотрим множество $\mathcal{P}(U) = \Omega_F(F(U)) \cup \Omega_G(G(U)) \cup \Omega_Y(Y(U))$, то есть $\mathcal{P}(U)$ — множество переменных, которые получились в результате замены нагрузки всех вершин и ребер из U.

Рассмотрим функцию

$$\Omega : F(U) \cup G(U) \cup Y(U) \to \mathcal{P}(U),$$

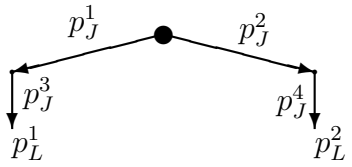


Рис. 1.1: Пример простого шаблона

где $\Omega = \Omega_F$ на множестве $F(U)$ ($\Omega \mid_{F(U)} = \Omega_F$), $\Omega \mid_{G(U)} = \Omega_G$ и $\Omega \mid_{Y(U)} = \Omega_Y$.

Очевидно из определения, что Ω — биективная функция, поэтому существует функция Ω^{-1}. Обозначим $I = \Omega^{-1}$ и будем называть *интерпретацией*, возникшей в результате замены индексов на переменные.

После этого сопоставления (замены нагрузки вершин и ребер на переменные в ИГ U) получим нагруженный граф. Выделим в нем множество вершин $V_{\mathcal{T}}$, которые назовем вершинами прикрепления, и занумеруем их числами от 1 до $|V_{\mathcal{T}}|$. Полученный граф назовем *простым шаблоном* $\mathcal{T}$. При этом будем писать $\mathcal{T} = \mathcal{T}(U, I, V_{\mathcal{T}})$, когда хотим подчеркнуть, что $\mathcal{T}$ получен из ИГ U и I, где I — интерпретация, возникшая в результате замены индексов на переменные, $V_{\mathcal{T}}$ — упорядоченное множество вершин прикрепления. Пример простого шаблона изображен на рисунке 1.1 (жирным кружком обозначена единственная вершина прикрепления).

Рассмотрим ИГ U', выделим в нем множество вершин $V_{U'}$, которые назовем вершинами прикрепления, и занумеруем их числами от 1 до $|V_{U'}|$ (вершины прикрепления образуют упорядоченное множество). Будем говорить, что ИГ U' и простой шаблон $\mathcal{T}$ *согласованы*, если выполнены следующие условия:

- U' и $\mathcal{T}$ совпадают как графы, и если в ИГ U' встречаются одинаковые индексы предикатов и записи, то в соответствующих местах шаблона $\mathcal{T}$ находятся одинаковые переменные из $\mathcal{P}_J$ и $\mathcal{P}_L$;

- i—ая вершина прикрепления ИГ U' совпадает с i—ой вершиной прикрепления простого шаблона $\mathcal{T}$, $i = 1, \ldots, |V_{U'}|$ и $|V_{U'}| = |V_{\mathcal{T}}|$.

То есть $\mathcal{T} = \mathcal{T}(U', I, V_{\mathcal{T}})$ для некоторой интерпретации I, и будем писать $\mathcal{T} = \mathcal{T}(U', I, V_{U'})$, когда хотим подчеркнуть, что ИГ U' и простой шаблон $\mathcal{T}$ согласованы.

Рассмотрим простой шаблон $\mathcal{T}_1$ и заменим в нем каждую переменную, на формулу над множество функций $\mathcal{C}$ и каким-либо множеством переменных $M \subseteq \mathcal{P}_J \cup \mathcal{P}_L$. В результате, полученный граф назовем *шаблоном* $\mathcal{T}_2$. При этом будем писать $\mathcal{T}_2 = \mathcal{T}_2(\mathcal{T}_1, M, V_{\mathcal{T}_1})$, когда хотим подчеркнуть, что $\mathcal{T}_2$ получен из простого шаблона $\mathcal{T}_1$, множества переменных M и упорядоченного множеств вершин прикрепления $V_{\mathcal{T}_1}$.

На рисунках запись $R_i(M)$ будет означать формулу над над множество функций $\mathcal{C}$ и множеством переменных из M.

Пример шаблона приведен на рисунке 1.2 (жирным кружком обозначена единственная вершина прикрепления).

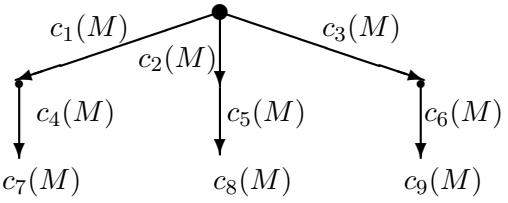


Рис. 1.2: Пример шаблона $\mathcal{T}_2$

Рассмотрим простой шаблон $\mathcal{T}_1 = \mathcal{T}_1(U, I, V_{\mathcal{T}_1})$. Обозначим множество входящих в него переменных $M(\mathcal{T}_1)$. Пусть $p_L^0 \in \mathcal{P}_L \setminus M(\mathcal{T}_1)$.

Преобразованием R назовем тройку

$$R = (\mathcal{T}_1(U, I, V_{\mathcal{T}_1}), \mathcal{T}_2(\mathcal{T}, M(\mathcal{T}_1) \cup \{p_L^0\}, V_{\mathcal{T}}), \xi),$$

где $\mathcal{T}_1$ — простой шаблон, $\mathcal{T}_2$ — шаблон (полученный из простого шаблона $\mathcal{T}$) в формулах которого встречаются только переменные из множества $M(\mathcal{T}_1) \cup p_L^0$, $V_{\mathcal{T}}$ упорядоченное множество вершин прикрепления простого шаблона $\mathcal{T}$, ξ — биективная функция сопоставления вершин прикрепления ($\xi : V_{\mathcal{T}_1} \to V_{\mathcal{T}}$). Левой частью преобразования R будем называть простой шаблон $\mathcal{T}_1$.

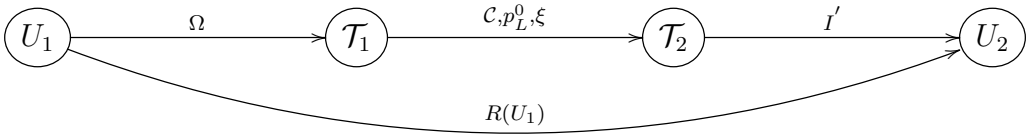


Рис. 1.3: Преобразование ИГ U_1 в ИГ U_2

Пусть ИГ U_1 и простой шаблон $\mathcal{T}_1$ согласованы, т.е. $\mathcal{T}_1 = \mathcal{T}_1(U_1, I, V_{U_1})$ для некоторой интерпретации I, и пусть I' – интерпретация множества переменных $M(\mathcal{T}_1) \cup p_L^0$, причем $I' = I$ на множестве переменных $M(\mathcal{T}_1)$. Тогда применением преобразования $R = (\mathcal{T}_1(U, I, V_{U_1}), \mathcal{T}_2(\mathcal{T}, M(\mathcal{T}_1) \cup p_L^0, V_{\mathcal{T}}), \xi)$ к ИГ U_1 назовем ИГ U_2, получающийся из шаблона $\mathcal{T}_2(\mathcal{T}, M(\mathcal{T}_1) \cup p_L^0, V_{\mathcal{T}})$ подстановкой вместо каждой формулы значения данной формулы в интерпретации I'. Заметим, что упорядоченное множество вершин прикрепления V_{U_2} в ИГ U_2 однозначно соответствует упорядоченному множеству вершин прикрепления V_{U_1} в ИГ U_1, так как ξ — биективная функция, и простой шаблон $\mathcal{T}_1$ согласован с ИГ U_1.

Результат применения преобразования R к ИГ U_1 будем обозначать $R(U_1) = U_2$. Преобразование ИГ U_1 в ИГ U_2 изображено на рисунке 1.3.

Чтобы легче было понять смысл введенных преобразований, приведем пример. Прежде всего напомним известный алгоритмом 2–3 дерева. Этот алгоритм используется для решения задачи поиска идентичных объектов. В него входят перестроения графов. Одно из перестроений и будет рассмотрено в качестве примера.

Известно множество структур данных позволяющих за разумное время решать динамическую задачу поиска идентичных объектов, в частности структура 2—3 дерева, которая позволяет решать эту задачу за логарифмическое время и линейную память (от объема библиотеки). Описание 2—3 дерева возьмем из [15].

2—3 деревом называется дерево, в котором каждая вершина, не являющаяся листом, имеет двух или трех сыновей, а длины всех путей из корня в листья одинаковы. Заметим, что дерево, состоящее из единственного узла является 2—3 деревом.

Рассмотрим линейно упорядоченное множество. Его можно представить с помощью 2—3 дерева следующим образом: элементы множества приписываются листьям слева направо по порядку. В каждой внутренней вершине v хранятся две величины: $L[v]$ — максимальный элемент в левом поддереве узла v и $M[v]$ — максимальный элемент в среднем (если две дуги, тогда в правом) поддереве v.

Пример 2—3 дерева для множества $\{1, 2, 3, 4, 5, 7\}$ (см. рис. 1.4).

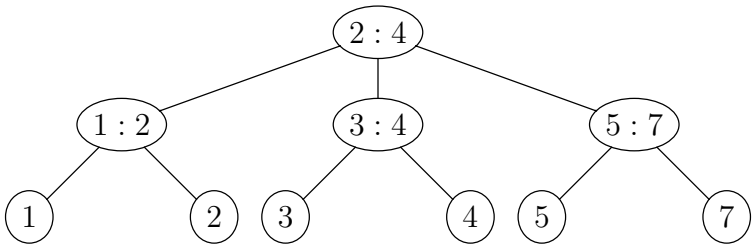


Рис. 1.4: Пример 2—3 дерева

Опишем алгоритм **A** функционирования 2—3 дерева:

A_0**: Поиск** y**.**

Поиск элемента y в 2—3 дереве происходит путем последовательного сравнения по следующему алгоритму: если $y \leq L[v]$, то переходим к левому сыну v, иначе если у v два сына, или $v \leq M[v]$, то переходим к среднему сыну v, иначе переходим к правому сыну v.

A_1**: Вставка** y**.**

$A_{1.0}$: если дерево состоит из единственного узла l с меткой y_0, образуем новый корень l'. Образуем новый узел v с меткой y. $L(v) = \min\{y_0, y\}, M(v) = \max\{y_0, y\}$ и делаем y_0, y сыновьями корня l', причем l будет левым сыном, если $y_0 < y$, и правым в противном случае.

$A_{1.1}$: если дерево содержит более одного узла, то нужно найти место для нового листа l, который будет содержать y. Для этого ищут элемент y в дереве (A_0). Если дерево содержит более одного элемента, то поиск y окончится в узле f, имеющим двух или трех сыновей, которые являются листами. Образуем новый лист l с меткой y.

$A_{1.2}$: если у f два сына с метками y_1 и y_2, то делаем l сыном узла f таким образом, чтобы сохранилась упорядоченность меток слева направо.

$A_{1.3}$: если у f три сына, делаем l сыном узла f, сохраняя упорядоченность меток. Затем, чтобы включить узел f и его четырех сыновей в дерево, выполняем шаг $A_{1.4}$ (рекурсивный).

$A_{1.4}$: образуем новый узел g. Два левых сына оставим сыновьями f, а два правых переделаем в сыновей узла g. Затем сделаем g братом узла f, сделав его сыном отца узла f. Если отец узла f уже имел трех сыновей, то надо рекурсивно повторять $A_{1.4}$ до тех пор, пока у всех узлов в дереве останется не более трех сыновей. Если у корня окажется четыре сына, образуем новый корень с двумя новыми сыновьями, каждый из которых будет иметь в качестве двух своих сыновей двух из четырех сыновей старого корня.

A_2**: Удаление** y**.**

Элемент y можно удалить из 2—3 дерева, по существу способом обратным к вставке. Пусть y — метка листа l.

$A_{2.0}$: если l — корень, удаляем его (y был единственным элементов в дереве).

$A_{2.1}$: если l — сын узла, имеющего трех сыновей, удаляем его.

$A_{2.2}$: если l — сын узла f, имеющего двух сыновей (s брат l), то возможно 2 случая ($A_{2.2.1}, A_{2.2.2}$).

$A_{2.2.1}$: если f — корень, то удаляем l и f и делаем корнем второго сына s.

$A_{2.2.2}$: если f — не корень и допустим f имеет брата g слева от себя (справа аналогично).

$A_{2.3}$: если у g два сына, делаем узел s самым правым сыном узла g, удаляем l и рекурсивно вызываем A_2, чтобы удалить f.

$A_{2.4}$: если у g три сына, то самого правого сына делаем левым сыном узла f и удаляем l.

Пусть база данных из примера приведенного выше и мы хотим добавить к ней число 6. В результате должно получиться следующее дерево (см. рис. 1.5). Как видно

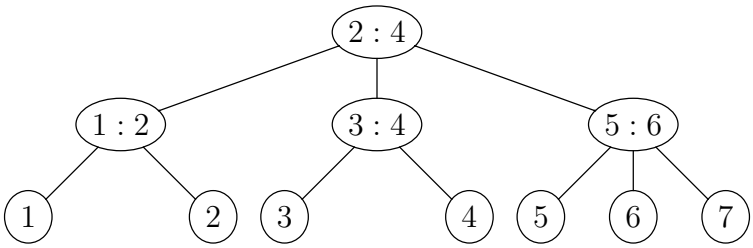


Рис. 1.5: К 2–3 дереву добавили 6

мы применили к части дерева некоторое преобразование.

Такое преобразование можно описать формально, используя в качестве структуры ИГ и шаблонный язык. Для этого опишем 2—3 дерево на языке ИГ.

Пусть $J = \{(a) : a \in \mathbb{R}\}$, $K = \{(a, b) : a, b \in \mathbb{R}\}$.

$$G_{id} = \left\{ g_{a,b}(x) = \begin{cases} 1, & \text{если } x \le a; \\ 2, & \text{если, } a < x \le b; \\ 3, & \text{иначе.} \end{cases}, (a,b) \in K \right\}; \tag{1.2}$$

$$F_{id} = \left\{ f_{=,a}(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{если } x \ne a. \end{cases}, a \in J \right\}; \tag{1.3}$$

$$\mathcal{F}^{id} = \langle F_{id} \cup G_{id} \rangle. \tag{1.4}$$

Таким образом каждому предикату $f_{=,a} \in F$ сопоставлен индекс $(a) \in J$, а каждому переключателю $g_{a,b} \in G$ сопоставлен индекс $(a,b) \in K$.

ИГ U_{id} над базовым множеством $\mathcal{F}^{id}$ строится по аналогии с 2—3 деревом следующим образом. В вершинах в которых стоят числа $A : B$ назовем переключательными и припишем им переключатели $g_{A,B} \in G$. К концевым вершинам проведем соответствующие предикатные ребра $f_{=,y} \in F$ (см. рис. 1.6).

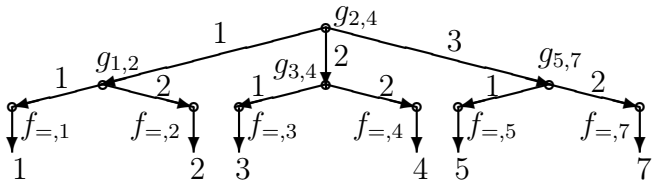


Рис. 1.6: ИГ построенный по аналогии с 2—3 деревом

Приведем пример как формально описывается преобразование описанное выше (добавление 6 к 2—3 дереву в виде ИГ).

Пусть $Z = \{z_1, z_2, \ldots, z_k\}$ — множество переменных, принимающих значения из $\mathbb{R}$. Пусть $\widehat{Z} = (z_{(1)}, z_{(2)}, \ldots, z_{(k)})$ — вариационный ряд соответствующий множеству Z, т.е. $\widehat{Z}$ это вектор, компоненты которого есть упорядоченные по неубыванию значения переменных множества Z.

Обозначим $\mu_i(Z) = z_{(i)}$, т.е. $\mu_i(Z)$ — $i-$ый элемент вариационного ряда, соответствующего Z. Введем следующие обозначения для функций:

$$c^{i,j}(Z) = (\mu_i(Z), \mu_j(Z)); c^i(Z) = (\mu_i(Z)); c^i_L(Z) = \mu_i(Z), i, j \in \mathbb{N} \tag{1.5}$$

Вершина выделенная жирным (‘’•‘’) на всех рисунках в этом примере будет соответствовать вершине прикрепления.

Рассмотрим ИГ U_1 (рис. 1.7).

Нужно найти такое преобразование R_e, чтобы $R_e(U_1) = U_2$ (рис. 1.8), в результате добавления новой записи y_3 к ИГ U_1, где $(y_{(1)}, y_{(2)}, y_{(3)})$ — вариационный ряд, соответствующий множеству $\{y_1, y_2, y_3\}$.

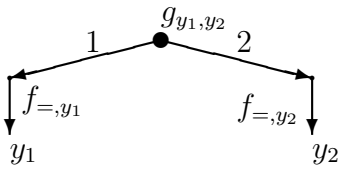


Рис. 1.7: ИГ $U_1, (y_1 < y_2)$

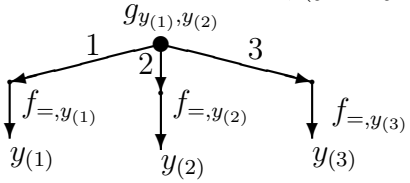


Рис. 1.8: ИГ U_2

Предъявим простой шаблон $\mathcal{T}_1(U_1, I, V_{U_1})$, согласованный с U_1 (рис. 1.9), а соответствующая этому простому шаблону интерпретация I: $I(p^1_K) = (y_1, y_2) = g_{y_1,y_2}; I(p^1_J) = (y_1) = f_{=,y_1}; I(p^2_J) = (y_2) = f_{=,y_2}; I(p^1_L) = y_1; I(p^2_L) = y_2$.

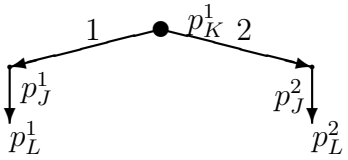


Рис. 1.9: Простой шаблон $\mathcal{T}_1$

Следуя введенным обозначениям, $M(\mathcal{T}_1) = \{p^1_J, p^2_J, p^1_K, p^1_L, p^2_L\}$ и $p^0_L \notin M(\mathcal{T}_1)$. Рассмотрим интерпретацию I' множества переменных $M \cup p^0_L$, причем

- $I' = I$ на множестве переменных M;
- $I'(p^0_L) = y_3$ (запрос на добавление).

Предъявим теперь шаблон $\mathcal{T}_2(\mathcal{T}, M_L, V_\mathcal{T})$ (рис. 1.10), $M_L = \{p^0_L, p^1_L, p^2_L\}$.

Исходя из определения формул преобразования, получаем:

$I'(c^{1,2}(M_L)) = (y_{(1)}, y_{(2)}) = g_{y_{(1)},y_{(2)}}$; $I'(c^1(M_L)) = (y_{(1)}) = f_{=,y_{(1)}}$; $I'(c^2(M_L)) = (y_{(2)}) = f_{=,y_{(2)}}$; $I'(c^3(M_L)) = (y_{(3)}) = f_{=,y_{(3)}}$; $I'(c^1_L(M_L)) = y_{(1)}$; $I'(c^2_L(M_L)) = y_{(2)}$; $I'(c^3_L(M_L)) = y_{(3)}$.

Если положить $y_1 = 5, y_3 = 6, y_2 = 7$, то получим преобразование, используемое в 2—3 дереве, которое было описано выше (см. рис. 1.4 и рис. 1.5).

2 Динамический информационный граф (ДИГ)

Следующие понятия будут введены для определения *кода информационного графа на запросе*. Код информационного графа на запросе будет использоваться для описания передвижения автомата по вершинам ИГ.

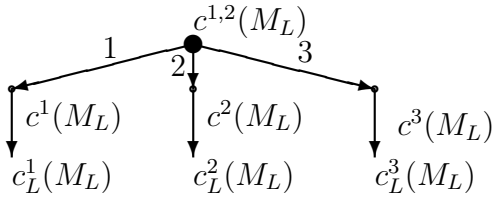


Рис. 1.10: Шаблон $\mathcal{T}_2$

Рассмотрим ИГ U, $N(U)$ — множество входящих в него вершин. Пусть $\beta, \beta^{'} \in N(U)$, тогда путем $\pi(\beta, \beta^{'})$ назовем последовательность вершин и ребер ИГ U, которая начинается в вершине β, заканчивается в вершине $\beta^{'}$, и в этой последовательности два любых соседних элемента инциденты. Длиной пути $l(\pi(\beta, \beta^{'}))$ назовем количество ребер, участвующих в нем. Пусть $\Pi(\beta, \beta^{'})$ — множество всех путей из β в $\beta^{'}$. Тогда расстоянием между вершинами $\beta, \beta^{'} \in N(U)$ назовем минимальную из всех путей длину $l(\pi(\beta, \beta^{'})), \pi(\beta, \beta^{'}) \in \Pi(\beta, \beta^{'})$, и обозначим $(\beta, \beta^{'}) = \min\{l(\pi(\beta, \beta^{'})) : \pi(\beta, \beta^{'}) \in \Pi(\beta, \beta^{'})\}$. Эксцентриситетом вершины $\beta \in N(U)$ назовем число $\varepsilon(\beta) = \max\limits_{\beta^{'} \in N(U)} (\beta, \beta^{'})$. *Радиусом* ИГ U назовем число $r(U) = \min\limits_{\beta \in N(U)} \varepsilon(\beta)$.

Через $\mathcal{G}(N, R)$, $N, R \in \mathbb{N}$ обозначим класс ИГ радиуса не больше R таких, что количество ребер инцидентных любой вершине графа не превосходит N.

Рассмотрим ИГ U, $E(U)$ — множество входящих в него ребер. Ребро инцидентное вершинам $\beta_1, \beta_2 \in N(U)$ будем обозначать $e(\beta_1, \beta_2)$.

Рассмотрим ИГ U_2 и его подграф (подмножество ребер и инцидентным им вершин) U_1 (пишем $U_1 \subseteq U_2$). Множество $\Sigma(U_1, U_2) = \{\beta : \beta \in N(U_1)$ и $\exists \beta_1 \in N(U_2 \backslash U_1), \exists\ \beta_2 \in N(U_1),\ e(\beta_1, \beta) \in E(U_2),\ e(\beta_2, \beta) \in E(U_1)\}$ назовем множеством *граничных вершин информационных графов* U_1 *и* U_2.

Пусть количество различных типов предикатов и переключателей конечно и равно K_T. Сопоставим им в биективном соответствие натуральные числа от 1 до K_T. Пусть $U_1, U_2 \in \mathcal{G}(N, \infty)$ и $U_1 \subseteq U_2$ ($N < \infty$). Назовем *кодом вершины на запросе* $x \in X$ *относительно пары* (U_1, U_2) пятерку $(k_1, k_2, k_3, k_4, k_5)$, где $k_1 = 0$, если вершина предикатная; $k_1 = 1$, если она переключательная; $k_2 = 0$, если вершина корень; $k_2 = 1$, если вершина листовая; $k_2 = 2$ в остальных случаях; $k_3 \in \{1, \ldots, N+1\}$ и в случае переключательной вершины принимает значение соответствующего ей переключателя на запросе $x \in X$, если это значение принадлежит {1,. . . ,N}, и $k_3 = N+1$ во всех остальных случаях; $k_4 \in \{0, 1\}$, $k_4 = 1$, если вершина принадлежит множеству граничных вершин $\Sigma(U_1, U_2)$ и $k_4 = 0$, если не принадлежит; $k_5 \in \{0, 1, \ldots, K_T\}$ принимает значение 0 если вершина предикатная или номер типа соответствующего ей переключателя иначе.

Кодом ребра на запросе $x \in X$ *типа поиск относительно пары* (U_1, U_2) назовем тройку (k_1^r, k_2^r, k_3^r), где $k_1^r = 0$, если ребро предикатное; $k_1^r = 1$, если ребро переключательное; $k_2^r \in \{0, 1, \ldots, N\}$; $k_3^r \in \{0, 1, \ldots, K_T\}$ принимает значение 0 если ребро переключательное или тип предиката соответствующее этому ребру иначе. У пре-

дикатного ребра k_2^r равен значению переключателя на запросе $x \in X$. Таким образом предикатным ребрам приписан код $(0,0,t)$ или $(0,1,t)$, где t - тип предиката, а каждому переключательному ребру приписан код $(1,i,0)$, где $i \in \{1,\ldots,N\}$ номер переключательного ребра.

В общем случае множества запросов X и множество записей Y могут быть разные. Поэтому для того, чтобы определить код вершин и ребер ИГ на запросах типа вставка и удаление введем дополнительную функцию $\eta : Y \to X^l$, $l \in \mathbb{N}$, $l < \infty$, $\eta = (\eta_1, \ldots, \eta_l)$.

Кодом вершины на запросе $y \in Y$ типа вставка, удаление относительно пары (U_1, U_2) будет l пятерок кодов вершин, которые соответствуют запросам $\eta_1(y), \ldots, \eta_l(y)$ типа поиск. *Кодом ребра на запросе $x \in Y$ типа вставка, удаление относительно пары* (U_1, U_2) будет значение l троек кодов ребер, которые соответствуют запросам $\eta_1(y), \ldots, \eta_l(y)$ типа поиск.

Рассмотрим ИГ U, в нем вершины и ребра имеют нагрузку. Граф $K(U)$ полученный из ИГ U удалением нагрузок вершин и ребер, назовем *каркасом* ИГ U.

Кодом ИГ U_1 относительно ИГ U_2 на запросе, назовем нагруженный каркас $K(U_1)$, где нагрузка каждой вершины $K(U_1)$ это код данной вершин на данном запросе относительно пары (U_1, U_2), а на нагрузка каждого ребра $K(U_1)$ это код данного ребра на данном запросе относительно пары (U_1, U_2). Через $\mathcal{K}(N, R)$ обозначим множество кодов ИГ U_1 относительно ИГ U_2, где U_1 пробегает класс ИГ $\mathcal{G}(N, R)$, U_2 пробегает класс ИГ $\mathcal{G}(N, R+1)$ и $U_1 \subseteq U_2$. Понятно, что $|\mathcal{K}(N, R)| < \infty$.

Пусть $N, R \in \mathbb{N}$, $\mathcal{P}$ — некоторое конечное множество преобразований, шаблоны которых порождены ИГ из $\mathcal{G}(N, R)$, и $P_0 \in \mathcal{P}$, где P_0 — тождественное преобразование такое, что $P_0(U) = U$ для произвольного ИГ U из $\mathcal{G}(N, R)$.

Пусть $N(U_1)$ множество вершин ИГ U_1. Рассмотрим множество вершин $\mathfrak{B} \subseteq N(U_1)$ и обозначим через $U(\mathfrak{B}, U_1)$ — ИГ, полученный из ИГ U_1 как подграф на вершинах $\mathfrak{B}$. Под подграфом на вершинах $\mathfrak{B}$ здесь понимается множество вершин $\mathfrak{B}$ и множество всех инцидентных им ребер.

Пусть $\mathcal{A}$ — конечный автомат (подробно об определении конечного автомата можно прочитать в [2]). Будем считать, что автомат $\mathcal{A}$ перемещается по вершинам ИГ $U_2 \in \mathcal{G}(N, \infty)$ и в каждый момент времени обозревает окрестность U_1 текущей вершины радиуса R, т.е. будем считать, что входным символом автомата $\mathcal{A}$ является код обозреваемой окрестности относительно U_2. Понятно, что эти коды будут принадлежать $\mathcal{K}(N, R)$ и значит входным алфавитом автомата $\mathcal{A}$ является конечное множество $\mathcal{K}(N, R)$. Результатом этого автомата $\mathcal{A}$ на обозреваемой окрестности текущей вершины будет некоторое преобразование этой окрестности и перемещение в некоторую вершину преобразованной окрестности. Тем самым выходным символом автомата $\mathcal{A}$ будет пятерка $b = (\mathfrak{B}, \pi, R, \beta, e)$, где

- $\mathfrak{B} \subseteq N(U_1)$ определяет множество вершин обозреваемой окрестности к которому будет применено преобразование;

- π — функция нумерации граничных вершин $U(\mathfrak{B}, U_1)$ и U_2 (содержательно это множество граничных вершин $U(\mathfrak{B}, U_1)$ и U_1 объединенное со множеством вершин в коде которых $k_2 = 1$ и одновременно принадлежащих $U(\mathfrak{B}, U_1)$). Пронумерованные граничные вершины назовем вершинами прикрепления;

- R — преобразование из конечного множества $\mathcal{P}$ применяемое к ИГ $U(\mathfrak{B}, U_1)$, причем ИГ $U(\mathfrak{B}, U_1)$ с заданной функцией π нумерацией вершин прикрепления согласован с левой частью преобразования R;

- $\beta \in N(R(U(\mathfrak{B}, U_1))) \cup \{U_1 \setminus U(\mathfrak{B}, U_1)\}$ — следующая текущая вершина автомата $\mathcal{A}$;

- $e \in \{0, 1, 2\}$ отвечает за продолжение и выдачу ответа автоматом $\mathcal{A}$. Автомат еще не нашел ответ и продолжает функционирование ($e = 0$), либо завершает функционирование и возвращает информацию о том, что искомая запись найдена ($e = 1$), либо завершает функционирование с информацией, что искомой записи нет ($e = 2$).

Множество всех таких выходных символов b образует выходной алфавит B автомата $\mathcal{A}$.

Пусть U — ИГ над базовым множеством $\langle F, G\rangle$ из класса $\mathcal{G}(N, \infty)$, тогда пару $(\mathcal{A}, U)$ назовем динамическим информационным графом (ДИГ) типа (N, R) над $\mathcal{F}=\langle F, G, \mathcal{P}, \eta\rangle$ и обозначим $\mathcal{D} = (\mathcal{A}, U)$.

Опишем теперь функционирование ДИГ. Определим функционирование ДИГ $\mathcal{D} = (\mathcal{A}, U)$ типа (N, R) над $\mathcal{F}=\langle F, G, \mathcal{P}, \eta\rangle$ на запросе. В начальный момент текущей вершиной объявляется корень ИГ U. Рассматривается ИГ U_1 как подграф $U(i)$, с центром в текущей вершине и радиуса R. На вход автомата $\mathcal{A}$ подается код ИГ U_1 относительно ИГ U на запросе. Пусть $b = (\mathfrak{B}, \pi, R, \beta, e) \in B$ выходная буква автомата $\mathcal{A}$, тогда ИГ U меняется по следующему правилу. В ИГ $U(i)$ ИГ $U(\mathfrak{B}, U_1)$ заменяется на ИГ $U^{'}$, полученной в результате применения преобразования R к ИГ $U(\mathfrak{B}, U_1)$ ($R(U(\mathfrak{B}, U_1)) = U^{'}$) и "прикрепляется" к ИГ $U(i)$ посредством функции π и функции сопоставления вершин прикрепления ξ из преобразования R. После этого текущее положение автомата $\mathcal{A}$ меняется на β и в зависимости от значения последней компоненты выходной буквы b ДИГ $\mathcal{D}$ либо продолжает функционирование ($e = 0$), либо завершает функционирование и возвращает информацию о том, что искомая запись найдена (вставлена или удалена)($e = 1$), либо завершает функционирование с информацией, что записи нет ($e = 2$).

Замечание: Операция замены ИГ $U(\mathfrak{B}, U_1) \subseteq U$ на ИГ $C(U(\mathfrak{B}, U_1)) = U^{''}$ корректная, т.е. ДИГ $\mathcal{D} = (\mathcal{A}, U^{''})$ останется типа (N, R), где ИГ $U^{''}$ это ИГ полученный из ИГ U заменой подграфа $U(\mathfrak{B}, U_1)$ на $U^{'}$. Этот факт следует из определения множества преобразований $\mathcal{P}$, а именно, что все преобразования из $\mathcal{P}$ порождены шаблонами из $\mathcal{G}(N, R)$, т.е. после преобразования получим, что $U^{''} \in \mathcal{G}(N, \infty)$.

Сложность и объем ДИГ $\mathcal{D} = (\mathcal{A}, U)$ типа (N, R) над базовым множеством $\mathcal{F}=\langle F, G, \mathcal{P}, \eta\rangle$ определяется для фиксированного ИГ U. Объемом ДИГ $\mathcal{D}$ будем называть объем ИГ U (количество ребер в графе) и обозначим $Q(\mathcal{D})$.

Определим сложность ДИГ $\mathcal{D}$ на запросе $x \in X$ типа вставка (удаление, поиск) и обозначим их соответственно $T_i(\mathcal{D}, x)$ ($T_d(\mathcal{D}, x)$, $T_s(\mathcal{D}, x)$). Для этого введем сложность выходного действия автомата $\mathcal{A}$.

Пусть $Z : \mathcal{P} \to \mathbb{N}$, входным параметром, которой является $R \in \mathcal{P}$, а значением сложность выполненного преобразования. Другими словами с помощью L вводится сложность каждого элемента из множества преобразований $\mathcal{P}$.

Последовательность преобразований, выполненных автоматом $\mathcal{A}$, в ходе функционирования ДИГ $\mathcal{D}$ на запросе $x \in X$ типа вставка обозначим $f_{\mathcal{A}}(x, 1)$ (типа удаление $f_{\mathcal{A}}(x, 2)$, типа поиск $f_{\mathcal{A}}(x, 3)$).

Сложностью ДИГ $\mathcal{D}$ на запросе $x \in X$ типа вставка (удаление) назовем

- $T_i(\mathcal{D}, x) = \sum\limits_{R \in f_{\mathcal{A}}(x,1)} Z(R)$ ($T_d(\mathcal{D}, x) = \sum\limits_{R \in f_{\mathcal{A}}(x,2)} Z(R)$, $T_s(\mathcal{D}, x) = \sum\limits_{R \in f_{\mathcal{A}}(x,3)} Z(R)$).

Сложностью ДИГ $\mathcal{D}$ назовем $\max\{T_i(\mathcal{D}), T_d(\mathcal{D}), T_s(\mathcal{D})\}$ и обозначим $T(\mathcal{D})$.

3 ДИГ, решающий задачу поиска идентичных объектов с логарифмической сложностью

Пусть $V \subset Y \subseteq \mathbb{R}$, $|V| < \infty$, $X = Y$. Скажем, что ДИГ $\mathcal{D}$ решает задачу поиска идентичных объектов (ЗПИО) $\langle X, V, =\rangle$, если ответ на произвольный запрос $x \in X$ типа поиска равен $\{x\}$, если $x \in V$, и пуст в противном случае, и если на произвольном запросе типа вставки (удаления) записи $y \in Y$ результирующий ДИГ $\mathcal{D}$ выдает ответ $\{x\}$ на произвольный запрос $x \in X$ типа поиск, если $x \in V \cup \{y\}$ ($x \in V \setminus \{y\}$), и пуст в противном случае.

В качестве базового множества будем рассматривать множество

$$\mathcal{F}_{id} = \langle F_{id}, G_{id}, \mathcal{P}_{id}, \eta_{id}\rangle, \tag{1.6}$$

где F_{id}, G_{id} определены соотношениями (1.2), (1.3). В качестве функции η_{id} рассматриваем тождественную функцию $\eta_{id} : Y \to X$ (в ЗПИО $Y = X$). Базовое множество преобразований $\mathcal{P}_{id}$ состоит из 55 преобразований, которые изображены на рис. 1.7 — 1.30, а также включает в себя тождественное преобразование $R_0 \in \mathcal{P}_{id}$.

Во всех преобразованиях из $\mathcal{P}_{id}$ шаблоны порождены ИГ из $\mathcal{G}(5, 3)$, множество индексов $J = \{(a) : a \in \mathbb{R}\}$, $K = \{(a, b) : a, b \in \mathbb{R}\}$, и во всех преобразованиях использованы формулы, реализующие функции из $\mathcal{R}_{id}$, которое определим ниже.

Введем обозначения для краткости записи формул в преобразованиях. Пусть $\mathcal{T}$ — простой шаблон и $M(\mathcal{T})$ — множество переменных, входящих в этот простой шаблон.

Обозначим $M_J(\mathcal{T}) = M(\mathcal{T}) \cap P_J$, $M_K^{'}(\mathcal{T}) = M(\mathcal{T}) \cap P_K$, $M_L^{'}(\mathcal{T}) = M(\mathcal{T}) \cap P_L$. В случае когда шаблон $\mathcal{T}$ определен однозначно, будем писать просто множества $M_J, M_K^{'}$ и $M_L^{'}$.

Если $p \in P_K$, тогда значением этой переменной будет пара из K. Через p^1 и p^2 обозначим первую и вторую компоненту переменной p, т.е. $p = (p^1, p^2)$, p^1 и p^2 принимают значения из $\mathbb{R}$. Пусть $M \subseteq P_K$, тогда $\widehat{M} = \bigcup\limits_{p \in M} p^1 \cup \bigcup\limits_{p \in M} p^2$.

Пусть $M_L = M_L^{'} \sqcup p_L^0$ ($p_L^0 \in P_L, p_L^0 \notin M_L^{'}$), $M_K = \widehat{M_K^{'}}$, $M \subseteq M_J \cup M_K \cup M_L$, тогда $\mathcal{R}_{id}$ состоит из функций $r^{i,j}(M), r^i(M), r_L^i(M)$, где $i, j \in \{1, 2, 3, 4\}$ и функции определены соотношением (1.5), а так же еще одной функции

$$r_{sp}(p_L^0, (p_J^1), (a, b)) = \begin{cases} r^{1,2}(p_J^1, b), & \text{если } p_L^0 = a; \\ r^{1,2}(a, p_J^1), & \text{если } p_L^0 = b; \ , (a, b) \in K \\ r^{1,2}(a, b), & \text{иначе.} \end{cases} \tag{1.7}$$

.

Теорема 1. *Существует ДИГ $\mathcal{D}_{id} = (\mathcal{A}_{id}, U_{id})$ типа $(5, 3)$ над базовым множеством $\mathcal{F}_{id}$, определяемым соотношением (1.6), который решает ЗПИО $\langle X, V, = \rangle$, причем*

$$T(D_{id}) \leq (\lambda_{max} + 1)([\log_2 |V|] + 1);$$
$$Q(D_{id}) \leq 3|V|,$$

где $\lambda_{max} = \max\{Z(R) : R \in \mathcal{P}_{id}\}$.

Доказательство.

Доказательство основывается на алгоритме **A** (алгоритм 2—3 дерева), описанного выше. ИГ U_{id} будет графом построенный по аналогии с 2—3 деревом (см. рис. 1.6), а автомат $\mathcal{A}_{id}$, используя преобразования из $\mathcal{P}_{id}$, в некотором смысле будет повторять алгоритм **A**.

Доказательство разобьем на три подпункта, соответствующие запросам типа поиск, вставка и удаление. Каждый подпункт описывается в виде алгоритма, который выполняет автомат (при получении запроса автомат $\mathcal{A}_{id}$ переходит в соответствующее состояние).

Сложность и объем ДИГ $\mathcal{D}_{id}$ определяется при фиксированной библиотеке V.

Поиск

ДИГ $\mathcal{D}_{id}$ функционирует как ИГ U_{id}. Сложность ИГ U_{id} на запросе не превосходит $[\log_2 |V|] + 1$. Этот факт является следствием определения сложности ИГ, а так же очевидного свойства 2—3 дерева (высота дерева $h :] \log_3 |V| [\leq h \leq [\log_2 |V|])$. Поэтому,

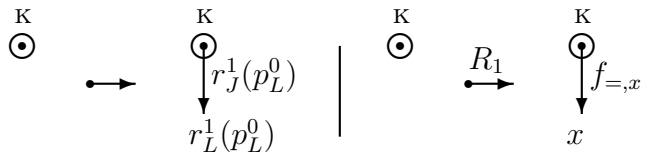


Рис. 1.11: Преобразование R_1 и его применение

чтобы найти нужную запись ИГ вычислит не более чем $[\log_2 N]$ переключателей, а так же один предикат, откуда и следует что $\widehat{T}(U_{id}) \leq [\log_2 |V|] + 1$.

Прежде чем перейти к вставке и удалению, разберем отдельно частные случаи (база данных пустая, состоит из 1,2 или 3 записей).

Во время доказательства в скобках будем указывать номер шага алгоритма **A**, проводя аналогию с этим алгоритмом.

Когда база данных пустая, ИГ U_{id} состоит из одной вершины корень. Если поступил запрос на удаление, то функционирование ДИГ завершается ошибкой. Если поступил запрос $x \in Y$ на **вставку**, то выходом автомата является преобразование $R_1 \in \mathcal{P}_{id}$ (см. рис. 1.11). На этом рисунке слева направо изображены соответственно простой шаблон $\mathcal{T}_1$, шаблон $\mathcal{T}_2$, ИГ U_1 (ИГ к которому применяем преобразование), и последним на рисунке изображен пример ИГ $U_2{=}R_1(U_1)$.

В дальнейшем все рисунки, изображающие преобразования из $\mathcal{P}_{id}$ будут содержать последовательно те же объекты (слева направо). Так же буква ''к'' находящаяся рядом с вершиной обозначает корень, вершина обведенная кругом ''$\odot$'' означает центр рассматриваемой окрестности (текущее положение автомата $\mathcal{A}_{id}$), а так же новое текущее положение автомата $\mathcal{A}_{id}$, которое будет соответствовать компоненте β выходного символа $b \in B$ автомата $\mathcal{A}_{id}$. Граничные вершины на одних и тех же преобразованиях могут отличаться (в зависимости от текущего ИГ U_{id}), самая верхняя вершина будет почти всегда граничной (за исключением начальных графов, когда окрестность радиуса 3 больше графа), а остальные будут интуитивно понятны. В неочевидных случаях, обозначим упорядоченное множество вершин прикрепления соответствующими символами ''ex_i'', $i \in \{1, \ldots, m\}$, где m количество вершин прикрепления.

Преобразование R_2 (см. рис. 1.12) предназначено для вставки $x \in Y$ ($A_{1.0}$), а преобразование R_3 (см. рис. 1.13) для удаления $x \in Y$ в ИГ U_{id}, когда количество записей в нем равняется 1 ($A_{2.0}$). В качестве примера ИГ U_2 выбран случай, когда $y < x$. В дальнейшем будем приводить пример ИГ U_2 без пояснений.

Преобразование R_e (пример преобразования ИГ, рис. 1.7 - 1.10) предназначено для вставки $x \in Y$, а преобразования R_4^L, R_4^R (см. рис. 1.14) для удаления $x \in Y$ в ИГ U_{id}, когда количество записей в нем равняется 2 ($A_{2.2.1}$).

Преобразования R_4^L, R_4^R отличаются тем, что удаляя $x \in Y$ мы должно разделить (в случае вставки можно воспользоваться вариационным рядом) одно по сути преобразование на два. Иногда на рисунке будем приводить только одно из преоб-

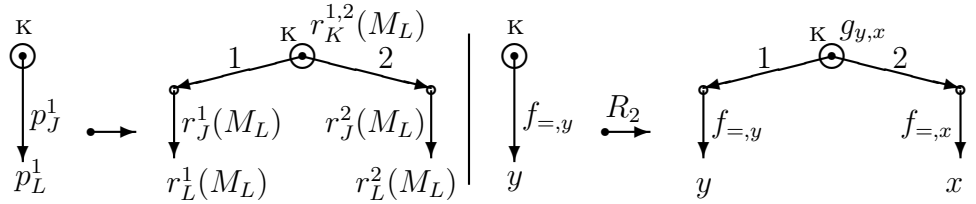


Рис. 1.12: Преобразование R_2 и его применение

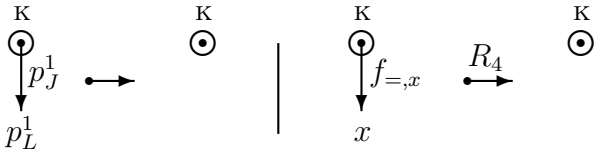


Рис. 1.13: Преобразование R_3 и его применение

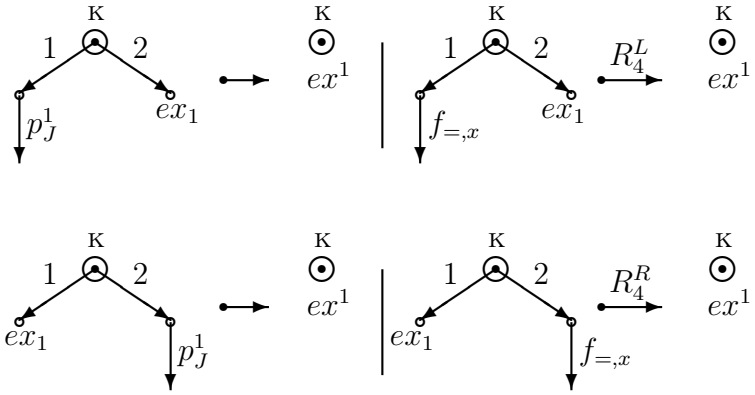


Рис. 1.14: Преобразования R_4^L, R_4^R и их применения

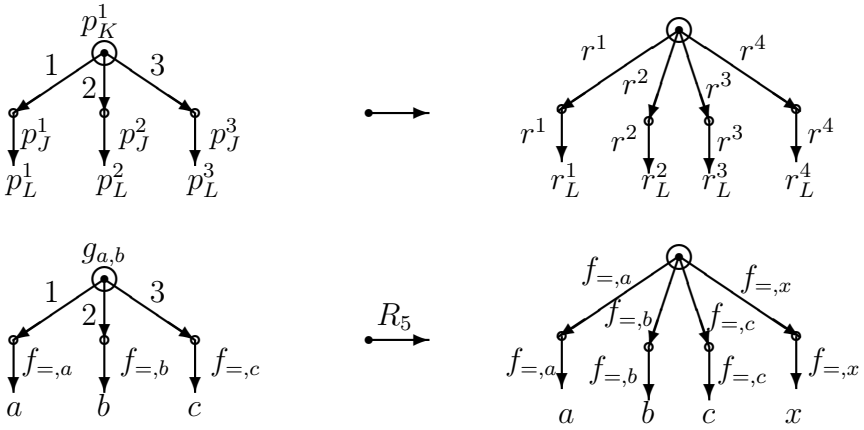


Рис. 1.15: Преобразование R_5 и его применение

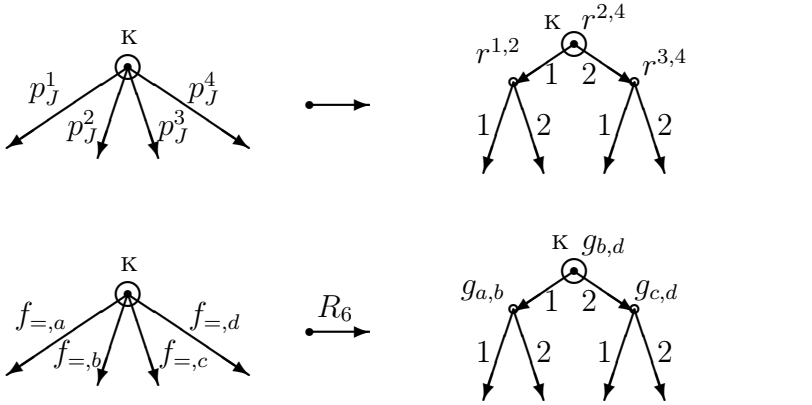


Рис. 1.16: Преобразование R_6 и его применение

разований вида C^L, C^M, C^R (буквы означают удаление левого поддерева, среднего и правого поддерева).

Так же преобразования R^L_4, R^R_4 будут использованы в случаях большого значения N, поэтому на рисунке не отображено правое и левое поддерево соответственно (при $|V| = 2$ это поддерево есть предикатное ребро и запись), вместо этого возле вершины указано, что она является вершиной прикрепления и имеет номер 1 (ex_1).

В этом и во всех последующих примерах функция ξ соответствия вершин прикрепления из применяемого преобразования будет иметь следующий вид $\xi(ex_i) = ex^i$, $i \in \{1, \ldots, m\}$, где m количество вершин прикрепления.

Для **вставки** $x \in Y$, когда ИГ U_{id} содержит 3 записи, потребуется выполнить 2 преобразования. Сначала автомат $\mathcal{A}_{id}$ применяет преобразование R_5 (см. рис. 1.15), которое необходимо как промежуточное преобразование, после чего автомат применяет преобразование R_6 (см. рис. 1.16) и завершает функционирование. Чтобы не усложнять рисунки, иллюстрирующие преобразования, все формулы, встречающиеся в шаблоне $\mathcal{T}_2$, будем описывать во время доказательства. В преобразовании R_5 все формулы зависят от M_L, в R_6 все формулы зависят от M_J.

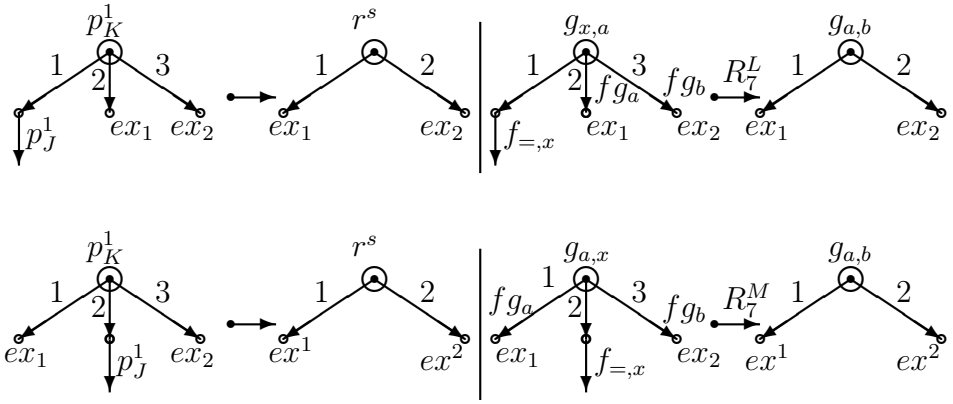


Рис. 1.17: Преобразование R_7^L,R_7^M и их применения

Некоторые преобразования отличаются только в том, что на месте предикатной вершины стоит переключательная, но для сути преобразования это не важно, а важно знать максимальный параметр у этих функций. Изображать важную информацию в примерах ИГ будем символом с информативным параметром. Например возможно ситуация, когда ex_i может быть предикатной вершиной и предикатному ребру (исходящему из этой вершины) будет приписан предикат $f_{=,b} \in F_{id}$, или быть переключательной вершиной и приписан переключатель $g_{a,b} \in G_{id}$. В обоих случаях для сути преобразования нужен только параметр b, поэтому для таких ситуаций на рисунках будем изображать символ ''fg_b'' рядом с этой вершиной (в формальном описании это два разных преобразования, так как будут использованы разные формулы). В таких ситуациях будем писать два множества переменных, от которых зависят формулы (подразумевая при этом два разных преобразования) (см. рис. 1.17).

Для **удаления** $x \in Y$, когда ИГ U_{id} содержит 3 записи, автомат $\mathcal{A}_{id}$ выполнит 1 преобразование из трех R_7^L, R_7^M или R_8^k (см. рис. 1.17, 1.18) в зависимости от входного символа и завершит функционирование. В преобразовании $R_7^L \in \mathcal{P}_{id}$ в простом шаблоне $\mathcal{T}_1$ вершина соответствующая ex_1 может быть предикатной и предикатному ребру, выходящему из нее, приписана переменная p_J^2, или быть переключательной и ей приписана переменная p_K^2. Аналогично для вершины ex_2 могут приписаны в том же смысле переменные p_J^3 или p_K^3. Это раздвоение позволит не загромождать доказательство рисунками, и под R_7^L надо понимать два разных преобразования из $\mathcal{P}_{id}$: $R_{7.1}^L$ — это R_7^L с предикатными вершинами ex_1,ex_2, $R_{7.2}^L$ — это R_7^L с переключательными вершинами ex_1,ex_2.

Формула r^s в преобразовании $R_{7.1}^L$ есть формула $r^{1,2}(p_J^1, p_J^2)$, а в преобразовании $R_{7.2}^L$ есть формула $r^{2,4}(p_K^2, p_K^3)$. В дальнейшем такие ситуации будем коротко описывать $R_7^L(r^s = r^{1,2}(p_J^1, p_J^2)$ или $r^s = r^{2,4}(p_K^2, p_K^3))$. $R_7^M(r^s = r^{1,2}(p_J^1, p_J^2)$ или $r^s = r^{2,4}(p_K^2, p_K^3))$, $R_8^k(r^s = r^{1,2}(p_J^1, p_J^2)$ или $r^s = r^{2,4}(p_K^2, p_K^3))$.

Итак рассмотрим общий случай $|V| \geq 4$ (высота ИГ $U_{id} \geq 3$).

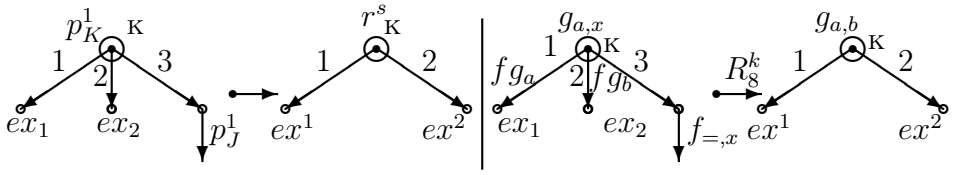


Рис. 1.18: Преобразование R^k_8 и его применение

Вставка $x \in Y$

Опишем алгоритм автомата **P**.

P_0: Автомат $\mathcal{A}_{id}$ находится в состояние поиска (A_0) в котором на выход он выдает тождественное преобразование, а новую текущую вершину выбирает согласно третьей компоненте кода текущей вершины входного символа (k_3). Дойдя до того момента, что расстояние до записи стало равно 2, останавливает процедуру поиска. Чтобы дойти до этого места автомат прошел не более чем по $[\log_2 |V|]$ ребрам графа. Автомат $\mathcal{A}_{id}$ переходит в состояние P_1.

P_1: Если автомат $\mathcal{A}_{id}$ находится в корне ИГ U_{id}, то если корень предикатная вершина и выходит из него 4 предикатных ребра, тогда выходом автомата $\mathcal{A}_{id}$ будет преобразование R_6 ($A_{1.4}$) и завершение функционирования, иначе просто завершение функционирования. Если автомат находится не корне ИГ U_{id}, то в зависимости от входа (код ИГ на запросе $x \in Y$ радиуса 3 с центром в той точке, где находится автомат) на выход он выдает соответствующее преобразование R_e ($A_{1.2}$) или R_5. Если выходом было преобразование R_e, то автомат переходит в состояние P_3 (правки параметров). Если выходом было преобразование R_5, то автомат переходит в состояние P_2.

P_2: В зависимости от входа, автомат на выход выдает одно из преобразований R_9 — R_{13} (см. рис. 1.19—1.23) ($A_{1.3}$ или $A_{1.4}$). R_9(все формулы зависят от M_J), $R_{10}(r^{1,2}(M_J), r^{3,4}(M_J)$, $r^s = r^{2,3}(p^2_K, p^2_J))$, $R_{11}(r^{1,2}, r^{3,4}, r^2, r^4$ зависят от M_J, $r^{s_1} = r^{2,4}(p^2_K, p^3_K))$, $R_{12}(r^{1,2}, r^{3,4}, r^2, r^4$ зависят от M_J, $r^{s_1} = r^2(p^2_K), r^{s_2} = r^2(p^3_K))$, $R_{13}(r^{1,2}, r^{3,4}, r^2, r^4$ зависят от M_J, $r^{s_1} = r^2(p^2_K), r^{s_2} = r^2(p^3_K))$.

Если выходом было преобразование R_9 или R_{10}, то автомат переходит в состояние P_3.

Если выходом было одно из преобразований R_{11}, R_{12} или R_{13}, то автомат переходит в состояние P_1.

P_3: В случае правки параметров структура самого ИГ U_{id} больше не измениться, могут измениться только параметры в переключателях.

Если автомат находится в корне ИГ U_{id}, то функционирование прекращается. Иначе в зависимости от входа (важно по какому ребру он перейдет к родителю текущей вершины), выходом будет одно из преобразований R^1_{14}, R^2_{14} или R^3_{14}, и переходит в состояние P_3. $R^1_{14}(r^s = r^{1,2}(r^2(r^1(p^1_K), p^0_L), r^2(p^1_K)))$, $R^2_{14}(r^{s_1} = r^{1,2}(r^2(r^2(p^1_K), p^0_L), r^1(p^1_K)))$, $R^3_{14}(r^{s_2} = r^{1,2}(p^1_K))$.

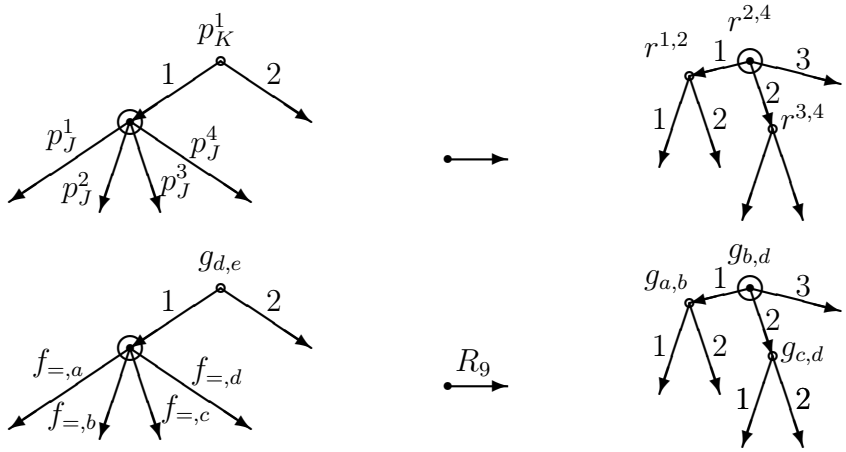


Рис. 1.19: Преобразование R_9

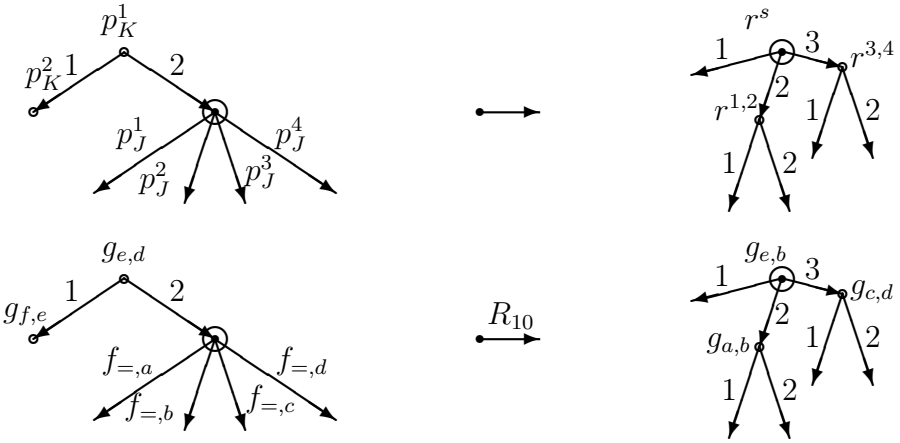


Рис. 1.20: Преобразование R_{10}

Очевидно, что сложность обратного прохода (P_1—P_3) можно оценить максимальной сложностью из всех преобразований умноженную на высоту дерева, так как после каждого преобразования текущее положение автомата уменьшает расстояние до корня как минимум на 1. Поэтому $\widehat{T}_{in}(\mathcal{D}_{id}) \leq (\lambda_{max}+1)([\log_2|V|]+1)$.

При этом несложно убедиться непосредственной проверкой, что каждый переключатель до и после преобразования сохранял свойство переключателей ИГ U_{id}: первый параметр переключателя обозначает максимальную запись в левом поддереве, а второй в среднем (правом если у переключателя двое сыновей). Поэтому полученный после завершения функционирования ДИГ $\mathcal{D}_{id} = (\mathcal{A}_{id}, U_{id})$, ДИГ $\mathcal{D}_{id} = (\mathcal{A}_{id}, U'_{id})$ будет решать ЗПИО $\langle X, V \cup x, = \rangle$.

Удаление $x \in Y$

Опишем алгоритм автомата **D**.

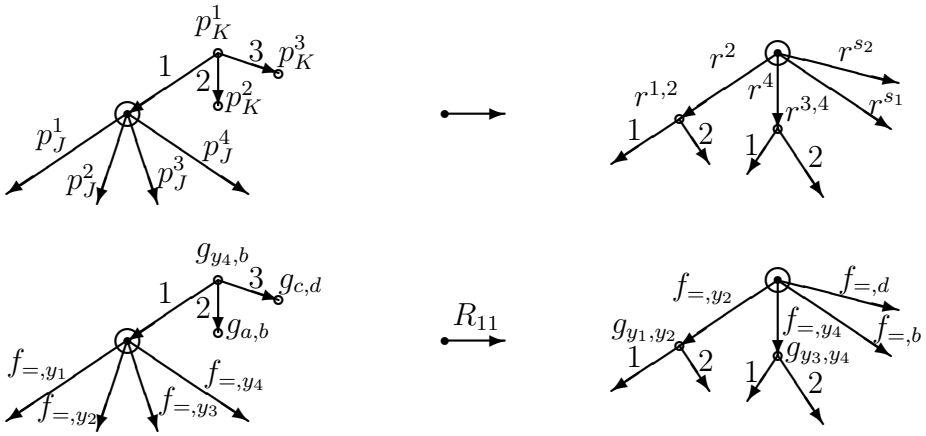


Рис. 1.21: Преобразование R_{11}

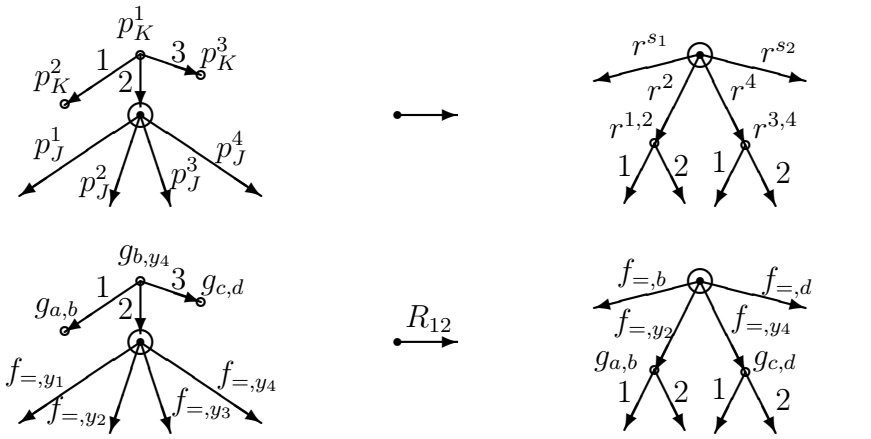


Рис. 1.22: Преобразование R_{12}

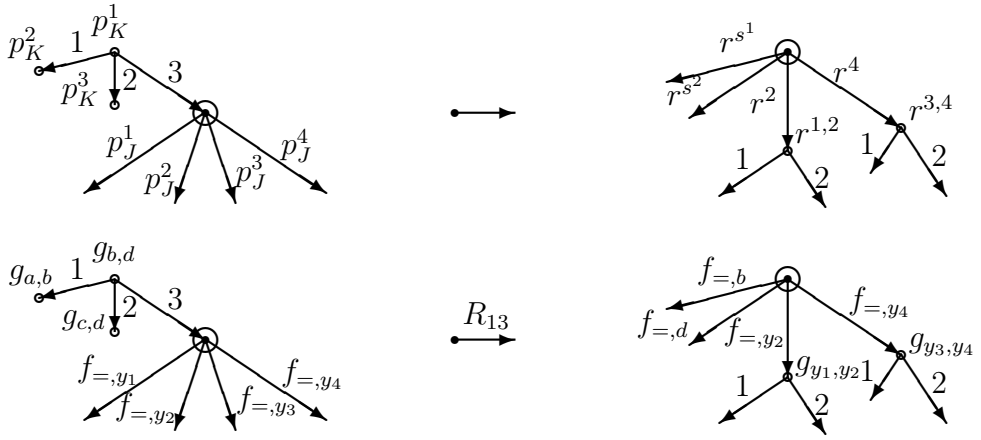


Рис. 1.23: Преобразование R_{13}

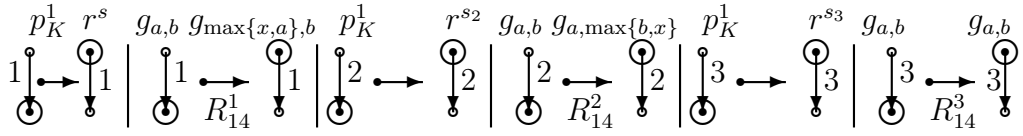


Рис. 1.24: Преобразование R_{14}^1,R_{14}^2,R_{14}^3 и их применения

D_0:

Аналогично P_0 автомат $\mathcal{A}_{id}$ переходит в состояние поиска (A_0). Дойдя до листовой вершины (до удаляемой записи x), останавливает процедуру поиска и следующее выходное действие подняться к родителю текущей вершины. Чтобы дойти до этого места автомат прошел не более чем по $[\log_2 |V|]+1$ ребрам графа. Если он не дошел до такой вершины, то функционирование завершается с ошибкой. После чего автомат переходит в состояние D_1.

D_1:

Автомат $\mathcal{A}_{id}$ поднимается к родителю текущей вершины. Если у родителя текущей вершины выходит 3 ребра ($A_{2.1}$), то выходом автомата $\mathcal{A}_{id}$ будет применить одно из преобразований R_7^L,R_7^M,R_8, иначе автомат $\mathcal{A}_{id}$ переходит в состояние D_2. $R_8(r^{s_2} = r^{1,2}(p_J^2, p_J^3)$ или $r^{s_2} = r^{2,4}(p_K^2, p_K^3)$, $r^{s_1} = r^1(p_J^3)$ или $r^{s_1} = r^2(p_K^3))$, где p_J^2, p_K^2 — соответствуют ex_1, p_J^3, p_K^3 — соответствуют ex_2 (здесь описано два разных преобразования). После этого автомат $\mathcal{A}_{id}$ переходит в состояние правки параметров D_5.

D_2:

Автомат $\mathcal{A}_{id}$ поднимается к родителю текущей вершины. Если текущая вершина корень, то применяется в зависимости от входа одно из преобразований R_4^L,R_4^R,R_7^L,R_7^M,R_8^k, после чего алгоритм заканчивает работу ($A_{2.2.1}$). Иначе переходит к D_3.

D_3:

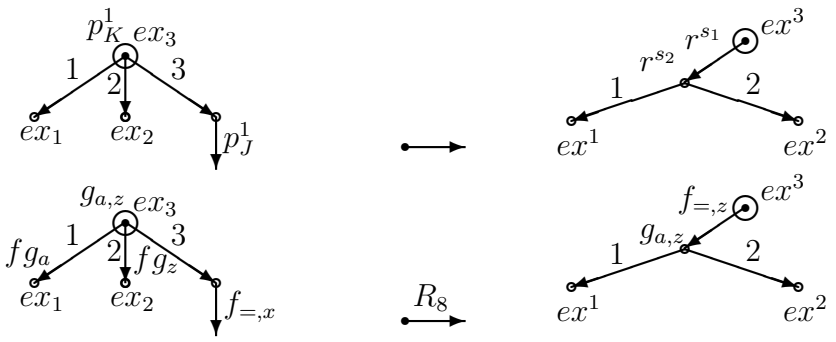


Рис. 1.25: Преобразование R_8 и его применение

Если у текущей вершины есть соседний брат с тремя сыновьями, то автомат $\mathcal{A}_{id}$ применяет одно из преобразований R^L_{14},R^R_{14},R^L_{15},R^R_{15} (см. рис. 1.26, 1.27) в зависимости от входа, иначе он переходит в состояние D_4. R^R_{14} — это 4 разных преобразования в зависимости от (i,j) (при $(i,j)=(1,2)$ предполагается, что из верхнего переключателя выходит 2 ребра). $R^R_{14}(r^{s_1}=r^{1,2}(p^2_K); r^{s_2}=r^{1,2}(p^2_J,p^2_K)$ или $r^{s_2}=r^{2,3}(p^4_K,p^2_K); r^{s_{1,2}}=r^{1,4}(p^3_K,p^2_K), r^{s_{2,3}}=r^{1,2}(p^3_K); r^s=r^2(p^2_K))$, где p^2_J,p^4_K соответствуют вершине ex_3 (два разных преобразования). R^L_{14} отличается от R^R_{14} тем, что предикатное ребро $f_{=,x}$ выходит из ex_3. В R^L_{14} нет дополнительного предикатного ребра r^s, а смысл всех формул сделать все переключатели правильными после переброски брата, удаляемой предикатной вершины (под правильным понимаем свойство параметров переключателя содержать информацию о максимальной записи в поддереве). Преобразования R^L_{15}, R^R_{15} аналогичны преобразованию R^L_{14}.

При этом дополнительное предикатное ребро участвуют в новом ИГ (см. рис. 1.26) только в случае, когда удаляемая запись была максимальна в этом (удаляемом) графе, и с помощью этого предикатного ребра хранится информация о том, на что нужно менять x во время правки параметров.

D_4:

Автомат $\mathcal{A}_{id}$ в зависимости от входа применяет одно из преобразований R^L_{16},R^R_{16},R^L_{17},R^R_{17}. После чего переходит в состояние D_1 $(A_{2.3})$. При этом в этих преобразованиях остается предикатное ребро $f_{=,x}$, чтобы в последующем можно было применить уже введенные преобразования. R^R_{16} отличается от R^L_{16} тем, что предикатное ребро p^1_J выходит из ex_3; R^L_{17} отличается от R^R_{17} тем, что предикатное ребро p^1_J выходит из ex_1. В $R^L_{16}(r^{s_1}=r^{1,2}(p^1_K);\ r^{s_{1,2}}=r^{2,3}(p^3_K,p^0_L);\ r^{s_{2,3}}=r^{1,3}(p^3_K,p^2_K))$. В $R^R_{17}(r^{s_1}=r^{1,3}(p^1_K,p^2_K);\ r^{s_{1,2}}=r^{2,1}(p^2_K);\ r^{s_{2,3}}=r^{1,4}(p^3_K,p^2_K))$. В $r^{s_{1,2}}$ из R^R_{17} нарушен порядок параметров (второй параметр меньше первого, это сделано для сохранения свойства переключателей в ИГ: содержание информации о максимальной записи в соответствующем поддереве).

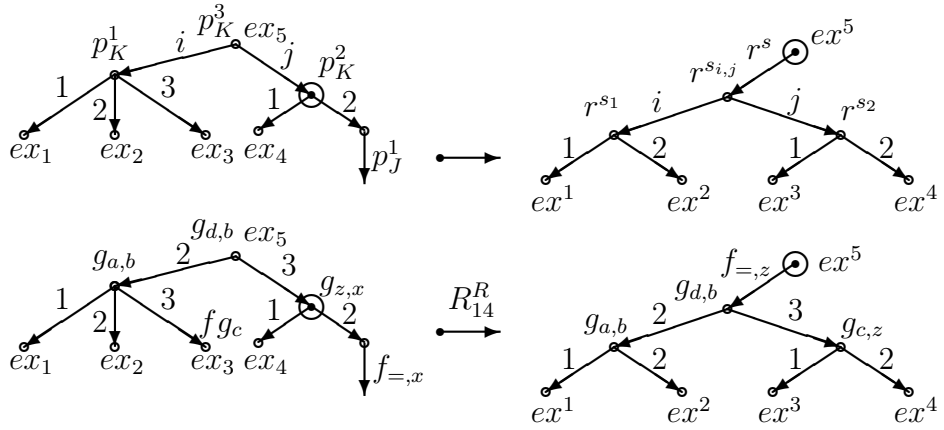


Рис. 1.26: Преобразование R_{14}^R и его применение: $(i,j) \in \{(1,2),(2,3)\}$

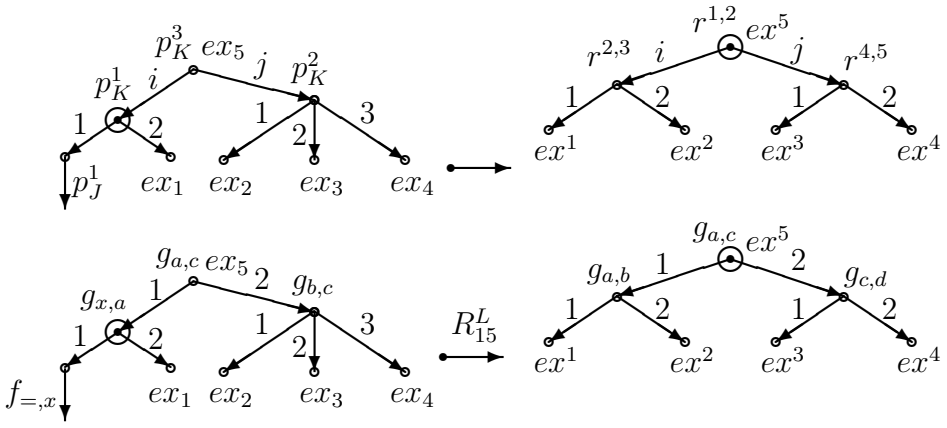


Рис. 1.27: Преобразование R_{15}^L и его применение: $(i,j) \in \{(1,2),(2,3)\}$

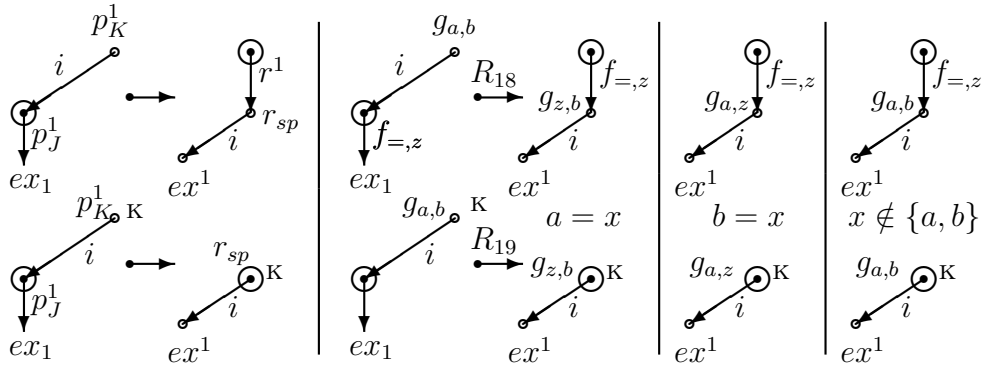


Рис. 1.28: Преобразование R_{18},R_{19} и их применения: $i \in \{1, 2, 3\}$

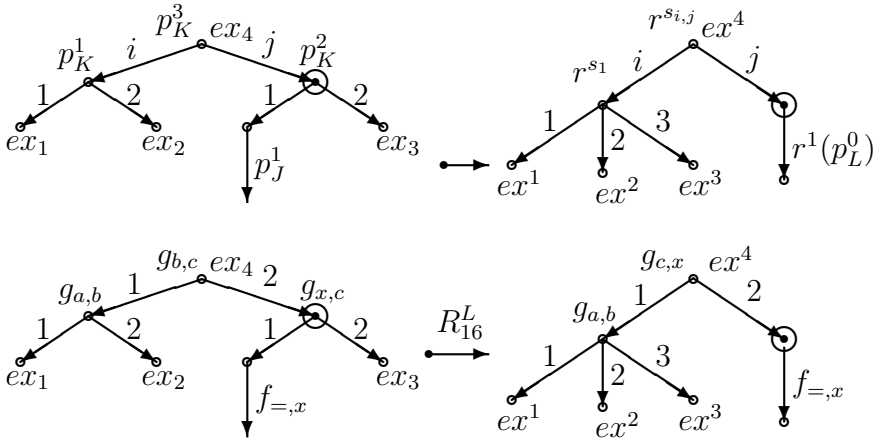


Рис. 1.29: Преобразование R_{16}^L и его применение: $(i, j) \in \{(1, 2), (2, 3)\}$

D_5:

Если родитель текущей вершины корень, то применяется преобразование R_{19} (см. рис. 1.28) и автомат $\mathcal{A}_{id}$ завершает функционирование, иначе автомат $\mathcal{A}_{id}$ переходит к состоянию D_6. На рис. 1.28 изображены все случаи ИГ U_2 в зависимости от значений формул. В R_{18} и R_{19} ($r_{sp}(p_L^0, p_J^1, p_K^1)$ определяется соотношением (1.7)).

D_6:

Автомат $\mathcal{A}_{id}$ на выходе выдает преобразование R_{18} и переходит к состоянию D_5.

Алгоритм **D** повторяет рассуждения доказательства работы алгоритма **A** для удаления [3]. За исключением D_5 и D_6, которые с помощью преобразований R_{18} и R_{19} (информация о заменяемых параметров передается в дополнительных предикатах) правильно переставляют параметры переключателей после того как удалили запись. В этом не сложно убедиться, если учесть, что если встречался параметр x, то он должен будет заменен на переменную z (максимальный оставшийся брат x) во всех предикатах ведущих по родителям удаляемой вершины к корню. Этот пара-

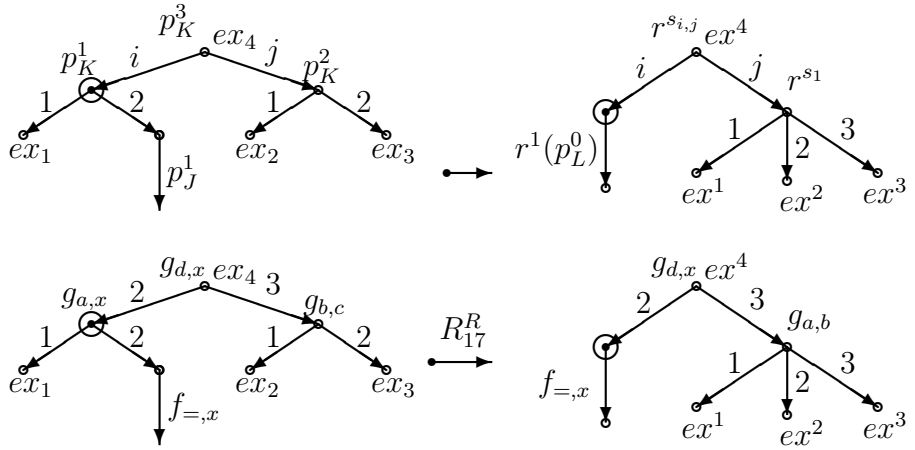


Рис. 1.30: Преобразование R_{17}^R и его применение: $(i,j) \in \{(1,2),(2,3)\}$

метр z передается с помощью дополнительного предикатного ребра, которое в итоге удаляется, когда автомат доходит до корня.

Очевидно, что сложность обратного прохода можно оценить максимальной сложностью из всех преобразований умноженную на высоту дерева. Поэтому $T(\mathcal{D}_{id}) \leq (\lambda_{max}+1)([\log_2 |V|]+1)$, откуда и следует доказательство теоремы, так как несложно заметить, что $Q(\mathcal{D}_{id}) \leq 3|V|$. ■

4 Потоковый динамический информационный граф (ПДИГ)

Обобщим понятие запроса к базе данных на случай вставки, удаления и пустого действия (запрос не требует действий для него). Для этого рассмотрим алфавит $A = \{\text{П}, \text{В}, \text{У}, \Lambda\}$. Обобщенным запросом назовем пару $(a,x), (a,x) \in \tilde{X}$, где $\tilde{X} = A \times X$. В данной работе рассматривается задача поиска идентичного объекта поэтому множество запросов и множество записей одинаково и равно X. При этом второй аргумент обобщенного запроса (буква из алфавита A) будет означать необходимое действие, которое нужно осуществить для запроса: П — поиск, В — вставка, У — удаление, Λ — ничего не делать. Далее везде под словом запрос будем понимать обобщенный запрос.

Потоком запросов H, $H : \mathbb{N} \to \tilde{X}$, назовем последовательность запросов, поступающих к базе данных через равные промежутки времени — такты. Другими словами поток запросов H означает, что к базе данных в i-ый такт времени будет подан запрос $H(i)$. Обозначим множество всех потоков через $\mathcal{H}$.

Рассмотрим функцию W, $W : \mathcal{H} \to A^\infty$ (через A^∞ обозначено множество всех сверхслов в алфавите A) сопоставляющую потоку запросов H сверхслово $W(H)$ из алфавита A, причем i-ая буква сверхслова $W(H)$ (обозначим ее через $W(H)(i)$) та-

кова, что $H(i) = (W(H)(i), x_i)$.

Для обработки потоков запросов введем понятие *потоковый динамический информационный граф* (ПДИГ) и обозначим его $\mathcal{D}_\Pi = (\mathcal{A}, U)$. Пусть дано бесконечное множество копий автомата $\mathcal{A}$. Будем считать, что несколько копий автомата $\mathcal{A}$ могут одновременно обслуживать несколько запросов. Например пусть имеется поток запросов на поиск $H = (\Pi, x_1), (\Pi, x_2), (\Pi, x_3), \ldots$, тогда при наличии 3 и более копий автомата $\mathcal{A}$ ПДИГ сможет параллельно обслужить три запроса x_1, x_2, x_3 на поиск, не дожидаясь завершения каждого ДИГ в отдельности. Если в потоке запросов H встречаются запросы на изменение базы данных (вставка и удаление), то при параллельной обработки этих запросов возможны конфликты преобразований (ИГ общий для всех автоматов).

Определим *функционирование ПДИГ* $\mathcal{D}_\Pi, \mathcal{D}_\Pi = (\mathcal{A}, U)$, *типа* (N, R) *над* $\mathcal{F}, \mathcal{F} = \langle F, \emptyset, \mathcal{P}, \eta_{id} \rangle$, *для потока запросов* H *из* $\mathcal{H}$. Считаем, что время применения любого преобразования R из $\mathcal{P}$, ровно 1 такт. Тем самым автомат $\mathcal{A}$ за один такт делает ровно одно преобразование над ИГ U. Заметим, что мы всегда можем выбрать единицу измерения времени (такт), что это будет выполнено (например как максимум из времен всех преобразований). Так же мы предполагаем, что такт запроса совпадает с тактом работы автомата.

После каждого такта работы ПДИГ — ИГ U может изменяться, поэтому будем рассматривать $U = U(t)$ $(t \in \mathbb{N} \cup \{0\})$. В начальный момент функционирования $U = U(0)$.

Рассмотрим запрос $H(i), H(i) = (a_i, x_i)$, который поступил для обработки к ПДИГ $\mathcal{D}_\Pi, \mathcal{D}_\Pi = (\mathcal{A}, U)$, $U = U(i)$ в i-ый такт, $i \geq 1$. ПДИГ действует в зависимости от типа запроса a_i. Если $a_i = \Lambda$ (ничего не делать), то ПДИГ ничего не делает для этого запроса и $U(i+1) = U(i)$, иначе ПДИГ функционирует как ДИГ для данного запроса.

Теперь определим понятие *сложности* ПДИГ. Для любого i из $\mathbb{N}$ через $T(\mathcal{D}_\Pi, H(i))$ обозначим количество тактов необходимое для обработки запроса $H(i)$ динамическим информационным графом $\mathcal{D}_\Pi$.

5 Динамическая задача поиска идентичных объектов (ДЗПИО)

Пусть H — поток запросов, $H : \mathbb{N} \to X$, где $X, X = \{\Pi, \text{В}, \text{У}, \Lambda\} \times \mathbb{N}$ — множество запросов. Буква в запросе означает его действие: поиск (П), вставка (В), удаление (У) или пустой запрос (Λ).

Множество всех потоков запросов обозначим через $\mathcal{H}$. Рассмотрим функцию множества записей $V : \{\mathbb{N} \cup \{0\}\} \times \mathcal{H} \to 2^X$, которая удовлетворяет следующим условиям:

$$V(i,H) = \begin{cases} \emptyset, & \text{если } i = 0; \\ V(i-1,H), & \text{если } i > 0 \text{ и } H(i) \text{ запрос на поиск или пустой запрос;} \\ V(i-1,H) \cup \{x\}, & \text{если } i > 0 \text{ и } H(i) = (\text{В}, x); \\ V(i-1,H) \setminus \{x\}, & \text{если } i > 0 \text{ и } H(i) = (\text{У}, x). \end{cases}$$

Функция V сопоставляет каждому такту i и потоку запросов H — множество записей, которые образуют базу данных после завершения функционирования ПДИГ для всех запросов до такта i включительно, если обработка любого запроса происходила бы мгновенно (за 1 такт).

Пусть дана функция $L : \mathbb{N} \to \mathbb{N}$. Будем говорить, что ПДИГ решает ДЗПИО (*логическую ДЗПИО*) со сложностью L, если для любого потока H и любого такта i выполняются следующие условия

- если $H(i) = (\text{П}, x)$, то результатом функционирования ПДИГ должен быть $\{x\}$ (да), если $x \in V(i,H)$ и пустое множество (нет) в противном случае. При этом количество тактов, необходимое на выдачу ответа, должно не превосходить $L(|V(i,H)|)$;

- если $H(i) = (\text{В}, x)$, то для любого запроса на поиск (П, z), поступившего в такт $i+1$, результат функционирования ПДИГ должен быть $\{z\}$(да), если $z \in V(i+1,H) = V(i,H) \cup \{x\}$, и пустое множество(нет) в противном случае;

- если $H(i) = (\text{У}, x)$, то для любого запроса на поиск (П,z), поступившего в такт $i+1$, результат функционирования ПДИГ должен быть $\{z\}$(да), если $z \in V(i+1,H) = V(i,H) \setminus \{x\}$, и пустое множество(нет) в противном случае.

Очевидно, что если ПДИГ решает ДЗПИО с выдачей ответа, то он решает ее в постановке да или нет. Также не сложно заметить, что если ПДИГ не решает задачу в постановке да или нет, то он не решает ее с выдачей ответа.

Напомним, что означает базовое множество для ПДИГ. Множество предикатов F – это множество функций, которые являются нагрузками ребер ИГ. ПДИГ базируется на конечном автомате, поэтому сами функции, приписанные ребрам, автомат увидеть не может. Автомат во время функционирования видит значение этих функций на запросе, который он обслуживает, а не сами функции. Также, помимо значений функций на запросе, автомат видит структуру подграфа ИГ. Подграф, который видит автомат, ограничен фиксированным параметром R, который означает радиус окрестности с центром в вершине, в которой находится автомат. Другими словами, если автомат находится в вершине v, то он видит все вершины, которые находятся на расстояния не большем, чем R ребер от нее. Пример радиуса видимости автомата изображен на рисунке 1.31. Естественно, сам ИГ не может иметь бесконечное

ветвление, так как конечный автомат даже не сможет увидеть окрестность радиуса 1 в этом случае. Поэтому, ПДИГ подразумевает, что ИГ имеет ограничение на ветвление (максимальное количество ребер инцидентных вершине), которые зависит от параметра N. На рисунке 1.31, N равняется 3 и достигается в вершине, в которой стоит автомат. Множество преобразований $\mathcal{R}$ означает, что автомат в каждый такт своего функционирования может применить преобразование из этого множества. Цель преобразований – это перемещение по ИГ и его локальная модификация для поддержания базы данных в актуальном состоянии.

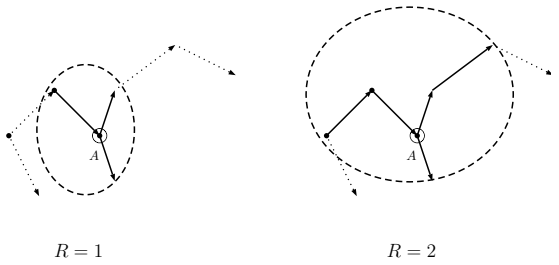


Рис. 1.31: Пример радиуса видимости автомата.

Введем понятие *конечный* ПДИГ. Будем говорить, что *ПДИГ конечный*, если для произвольного потока запросов H и произвольного такта времени i существует такая константа C,$C < \infty$, что для потока запросов $H', H'(1) = H(1), \ldots H'(i) = H(i), H'(i+1) = \Lambda, \ldots, H'(i+C) = \Lambda$, автомат обслуживающий запрос $H(i)$ завершит свое функционирование.

Неформально говоря, ПДИГ будем называть конечным, если автомат для любого потока запроса и любого такта, функционирует конечное время. ПДИГ, который не является конечным, будем называть бесконечным.

Введем понятие *селекторный ПДИГ*. Для этого напомним, что представляет из себя преобразование, которое применяет автомат. Формальное описание преобразования достаточно громоздкое, поэтому опишем его суть. Автомат, находясь в вершине ИГ, на вход получает код окрестности на запросе, который он обслуживает. Удобнее представлять код как граф вершинам и ребрам, которого присвоена конечная информация. Вершинам присвоены их тип: корень, внутренняя вершина или листовая. Ребрам присвоена пара – это тип предиката и его значение на запросе. Автомат на выход выдает инструкцию ИГ, на какую окрестность нужно заменить ту, которую он видит. При этом он может использовать конечный набор функций $\{\phi_1, \ldots, \phi_n\}$, которые могут создавать новые предикаты.

На рисунках, изображающих преобразования, вершину, в которой находится автомат, обозначаем через вершину в круге. Автоматы не знают значение запроса, которые они обслуживают. Поэтому запрос, который обслуживает автомат, будем обозначать $*$. На каждый такт своего функционирования автомат на вход получает код ИГ на запросе с центром в текущей вершине и радиусом его видимости. При

этом, если в правой части преобразования нет вершины, обведенной кругом, автомат завершает функционирование. Например, автомат на удаление всегда завершает функционирование после первого такта. Код ИГ на запросе представляет из себя следующее: каждой вершине сопоставляется число 0,1 или 2. При этом 0 соответствует корню ИГ, 1 внутренней вершине, 2 листовой. На рисунках вместо 0 будем писать букву "к" для наглядности.

Рассмотрим произвольную функцию $\phi \in \{\phi_1, \ldots, \phi_n\}$, которую может применить автомат в своем преобразовании. Автомат, зная тип предиката, знает также, сколько у него аргументов. Функция ϕ на вход получает аргументы и типы предикатов , типы вершин, а так же запись, которую обслуживает автомат. После этого эта функция может создать новый предикат или запись. Естественно, функция ϕ может создать только предикат, принадлежащий множеству предикатов F, над которым рассматривается ИГ. Оперируя этими функциями, автомат может в новой окрестности создавать новые предикаты из F.

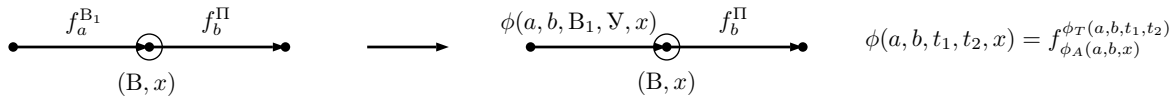


Рис. 1.32: Пример преобразования ИГ, которое использует функцию ϕ для создания нового предиката.

Пусть дано множество предикатов $F(T) = \{f^0, f^1, f_a^t(x) | t \in T, a \in \mathbb{N}\}$, где $|T| < \infty$ и $\forall t \in T, a \in \mathbb{N}$:

$$f_a^t(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{иначе} \end{cases}, f^0(x) \equiv 0, f^1(x) \equiv 1.$$

Пусть $T_0 = \{\text{П}, \text{В}_1, \text{В}_2, \text{У}\}$, то есть $F(T_0) = \{f^0, f^1, f_a^{\text{П}}, f_a^{\text{В}_1}, f_a^{\text{В}_2}, f_a^{\text{У}} : a \in \mathbb{N}\}$.

Рассмотрим ИГ над базовым множеством $F = F(T_0)$, изображенный на рисунке 1.32. Допустим, что автомат может использовать функцию $\phi(a, b, t_1, t_2, x) : \mathbb{N} \times \mathbb{N} \times T_0 \times T_0 \times \mathbb{N} \to F$ для преобразования предикатов. Рассмотрим пример функции ϕ. Пусть $\phi(a, b, t_1, t_2, x) = f^{\phi_T(a,b,t_1,t_2)}_{\phi_A(a,b,x)}$, где $\phi_T(a, b, t_1, t_2) = \text{У}$, если $a = b$ и t_2 иначе, $\phi_A(a, b, x) = a + b - x * (a + b)$. Видно, что, используя функцию ϕ, автомат создает новый предикат из F, опираясь на аргументы предикатов. При этом сами аргументы автомат не знает. Используя эту функцию, автомат может сказать ИГ сделать новую окрестность взамен той, которую он видит.

Преобразование будем называть селекторным, если оно не может создать новый аргумент, кроме тех, что есть во входной окрестности или запроса, который обслуживает автомат. Например, преобразование, описанное выше, не является селекторным, так как оно создает новый аргумент $\phi_A(a, b, x) = a + b - x * (a + b)$, который может не принадлежать множеству $\{a, b, x\}$. Преобразование $\phi(a, b, t_1, t_2, x) = f^{\phi_T(a,b,t_1,t_2)}_{\phi_A(a,b,x)}$,где

$\phi_A(a, b, x) = x$, является селекторным. Другой пример не селекторного преобразования изображен на рисунке 2.73. Пример селекторного преобразования изображен на рисунке 2.59.

ПДИГ будем называть селекторным, если преобразования, которые он использует, являются селекторными.

Глава 2

Верхние оценки ПДИГ

1 ПДИГ, допускающий параллельную обработку произвольных потоков запросов

Пусть

$$f_a(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{иначе} \end{cases}.$$

$$F = \{f_a(x) : a \in \mathbb{R}\}.$$

Введем базовое множества преобразований $\mathcal{R}$. Базовое множество $\mathcal{R}$ состоит из 12 преобразований, изображенных на рисунках 2.1, 2.3 и 2.4-2.8. Преобразования будут описаны в доказательстве теоремы 2.

В качестве базового множества для ПДИГ будем рассматривать множество

$$\mathcal{F} = \langle F, \emptyset, \mathcal{R}, \eta_{id} \rangle. \tag{2.1}$$

Через $]x[$ будем обозначать целую часть сверху от x (наименьшее целое число, которое не меньше x). Справедлива следующая теорема.

Теорема 2. *Существует конечный, селекторный ПДИГ $\mathcal{D}_{\Pi} = (\mathcal{A}, U)$ типа (2,2) над базовым множеством $\mathcal{F}$, определяемым соотношением (2.1), который решает ДЗПИО для любого потока запросов H из $\mathcal{H}$, со сложностью $L, L(n) =]n/2[+1$.*

Доказательство. Сначала опишем базовое множество преобразований

$$\mathcal{R} = \{R_1, R_2, \ldots, R_{12}\},$$

которые будет применять автомат $\mathcal{A}$ в зависимости от входной окрестности.

Все преобразования можно разделить на три типа: преобразования на вставку, удаление и перемещение. Цель ПДИГ создать динамический список, поэтому алгоритм вставки нового элемента в список будет следующим. Автомат должен пройти по всему списку и если не обнаружит вставляемой записи — вставить запись в конец списка. Поэтому преобразований на вставку будет три, и они изображены на рисунках 2.1 и 2.3. Преобразование, изображенное на рисунке 2.1, соответствует случаю пустой базы данных. Преобразования, изображенные на рисунке 2.3, соответствует случаю добавлению новой записи в конец списка.

формальное преобразование пример преобразования ИГ

Рис. 2.1: Вставка для случая пустой базы данных.

На рисунке 2.1 слева направо изображены соответственно простой шаблон $\mathcal{T}_1$, шаблон $\mathcal{T}_2$, ИГ U_1 (ИГ к которому применяем преобразование), и последним на рисунке изображен пример ИГ U_2. Здесь $M(\mathcal{T}_1) = \emptyset$, $p_L^0 \in \mathcal{P}_L \setminus M(\mathcal{T}_1)$, $I(p_L^0) = y$ (запрос на вставку), $r_J(p_L^0) = p_J$, $r_L(p_L^0) = p_L^0$, где $I(p_J) = f_y$.

Так же буква "к" находящаяся рядом с вершиной обозначает корень, вершина обведенная кругом означает центр рассматриваемой окрестности (текущее положение автомата $\mathcal{A}$), а так же новое текущее положение автомата $\mathcal{A}$, которое будет соответствовать компоненте β выходного символа $b \in B$ автомата $\mathcal{A}$. При этом если в правой части преобразования нет вершины, обведенной кругом, то это означает, что автомат завершает функционирование. Например автомат на поиск завершает функционирование, когда он находит запись, или не находит, но при этом находиться в конце списка.

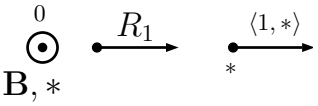
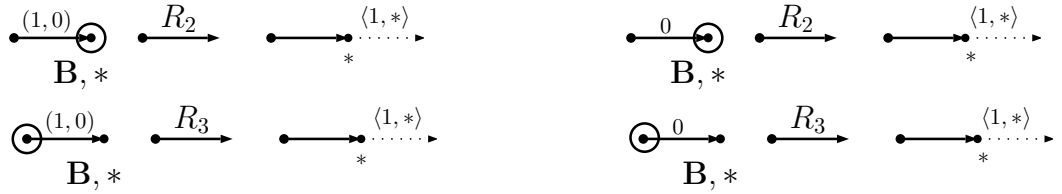


Рис. 2.2: Преобразование R_1 на вставку, с точки зрения автомата.

На рисунке 2.2, изображено преобразование R_1 с точки зрения автомата. Автомат на вход получает код ИГ на запросе $*$. Вершина, которой соответствует код 0, является корнем. Автомат дает инструкцию ИГ, что данную окрестность нужно заменить на окрестность, соответствующей правой части преобразования. В этой окрестности есть одно предикатное ребро. Этому ребру присвоен предикат первого типа $\langle 1, * \rangle$, который будет интерпретировать как f_*.



запись с типами предикатов сокращенная запись

Рис. 2.3: Преобразование R_2 и R_3 на вставку, с точки зрения автомата.

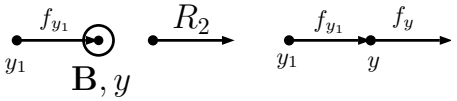


Рис. 2.4: Преобразование R_2 на вставку, с точки зрения ИГ.

В дальнейшем будем описывать все преобразования с точки зрения автомата. Так же будем использовать сокращенную запись преобразования. В преобразованиях не будем указывать код вершин, а так же тип предикатов. При этом в преобразованиях будем выделять ребра жирным, если автомат не меняет это ребро. Ребро будем изображать пунктиром, если автомат удаляет или вставляет это ребро. Это поможет понять какие ребра автомат преобразовывает, а какие ребра нет. Пример сокращенного преобразования изображен на рисунке 2.3.

Преобразования R_2 и R_3 добавляют запрос в конец списка. Они различаются только положением автомата. Пример изменения ИГ для преобразования R_2, изображен на рисунке 2.4.

Для большей ясности смысла введенных преобразований на вставку, приведем пример ИГ, который получен после нескольких преобразований на вставку. ИГ после последовательности запросов (B,y_1),(B,y_2),(B,y_3), (B,y_4),(B,y_5), изображен на рисунке 2.5.

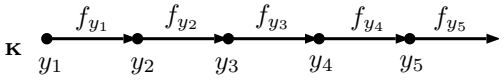


Рис. 2.5: Пример ИГ после нескольких вставок.

Преобразования на удаление изображены на рисунках 2.6 - 2.7. Преобразование R_4 применяется если удаляется последняя запись (самая дальняя от корня) в списке. Преобразование R_5 удаляет соседнюю справа запись, при этом вершина $\mathbf{v}_3$ может быть не последней в списке. В этом преобразование удаляется ребро ($\mathbf{v}_1$, $\mathbf{v}_2$), а ребро ($\mathbf{v}_2$, $\mathbf{v}_3$) меняет начало с $\mathbf{v}_2$ на $\mathbf{v}_1$. При этом черточкой на рисунке показано совпадение нагрузок ребер, то есть ребру ($\mathbf{v}_2$, $\mathbf{v}_3$) будет присвоен предикат, который соответствовал ребру ($\mathbf{v}_1$, $\mathbf{v}_2$) до преобразования.

Преобразование R_6 удаляет ребро, инцидентное вершине $\mathbf{v}_1$, при этом саму вершину не удаляет. Это преобразование удаляет вершину $\mathbf{v}_2$, а в качестве начала инцидентных ей ребер устанавливает вершину $\mathbf{v}_1$. Черточками показано соответствие нагрузок ребер, одинаковому количеству черточек соответствуют одинаковые предикаты, то есть нагрузка ребра $(\mathbf{v}_2, \mathbf{v}_3)$ совпадает с нагрузкой ребра $(\mathbf{v}_1, \mathbf{v}_3)$. Так же на рисунке 2.7 через "звездочки" со штрихами, обозначены записи, соответствующие вершинам. Преобразование R_6 меняет запись, находящуюся в вершине $\mathbf{v}_1$ на запись, находившуюся в вершине $\mathbf{v}_2$.

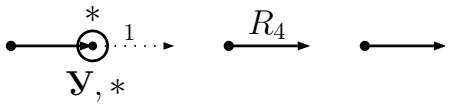


Рис. 2.6: Удаление последней записи.

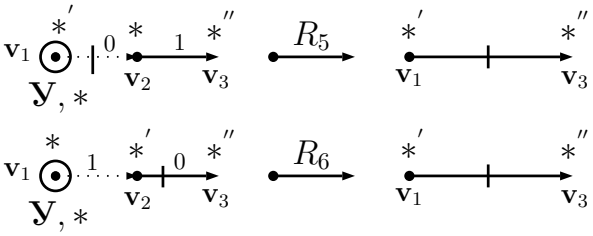


Рис. 2.7: Удаление в общем случае.

Преобразования $R_7, \ldots, R_{12}$ — это преобразования на перемещение по ИГ. Эти преобразования изображены на рисунке 2.8. R_7 это перемещение автомата по ИГ в случае если запись еще не найдена. Заметим, что автомат по возможности передвигается сразу на два ребра. R_8 и R_9 применяются автоматами для запроса на поиск и вставку, если запись найдена. Для запросов на удаление в этом случае применяется преобразование R_4, R_6 или R_5. В преобразованиях R_8, R_9 автомат завершает функционирование с сигналом $e = 1$ (запись найдена). R_{10} и R_{11} применяется для автоматов на поиск и удаление, если запись не найдена, при этом автомат завершает функционирование с сигналом $e = 2$ (запись не найдена). Для автоматов на вставку применяются преобразования R_3 и R_2 соответственно. Преобразование R_{12} применяется в случае пустой базы данных для запросов типа поиск и удаление. В этом случае автомат, обслуживающий запрос, завершает функционирование с сигналом $e = 2$.

Алгоритм работы автомата $\mathcal{A}$ зависит от типа запроса, который нужно обработать. Поэтому для каждого типа запроса отдельно опишем работу $\mathcal{A}$. Заметим, что все преобразования $R_1, \ldots, R_{12}$ однозначно применяются в зависимости от входной окрестности и типа запроса. Алгоритм автомата будет состоять в последовательном

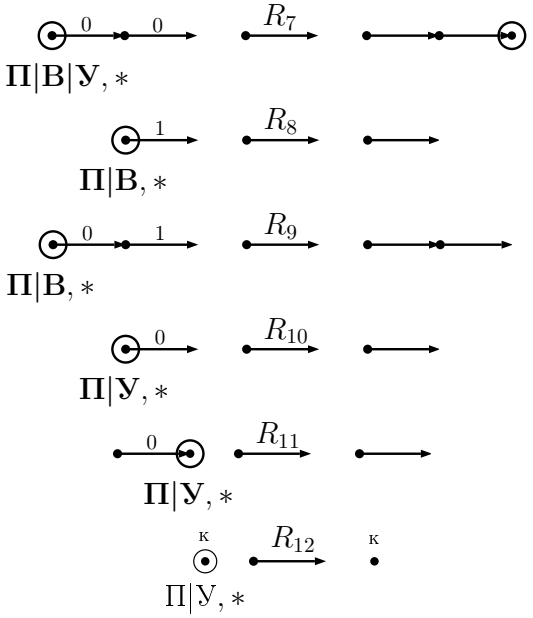


Рис. 2.8: Перемещение по ИГ.

применении этих преобразований, пока в правой части преобразования не будет присутствовать кружок возле вершины. Это будет означать, что автомат завершил свою работу.

Когда база данных пустая, ИГ U состоит из одной вершины (корень). Если поступил запрос на поиск или удаление, то применяется преобразование R_{12} и после этого функционирование завершается с сигналом $e = 2$ (запись не найдена). Если поступил запрос на вставку, то применяется преобразование R_1 и после этого функционирование завершается с сигналом $e = 1$ (запись вставлена).

Для общего случая, алгоритм автомата $\mathcal{A}$ состоит в последовательном прохождении ребер, начиная с корня, при помощи преобразований $R_7, \ldots, R_{11}$.

Автомат на поиск применят преобразование в зависимости от входной окрестности. Если окрестность соответствует левой части преобразования R_7, то автомат применяет преобразование R_7. Он перемещается по списку, просмотрев два его элемента. Заметим, что автомат находит искомую запись если применяет преобразование R_8 или R_9, и не находит, если применяет преобразование R_{10} или R_{11}.

Алгоритм работы автомата на запросах типа вставка или удаление, повторяет алгоритм автомата на запросе типа поиск. Отличие алгоритма на вставку заключается в том, что вместо преобразований R_{10} или R_{11}, автомат применяет преобразование R_3 или R_2 соответственно. Отличие алгоритма на удаление заключается в том, что вместо преобразований R_8 или R_9, автомат применяет преобразование R_4, R_6 или R_5.

Докажем несколько вспомогательных лемм, для доказательства, что приведен-

ный ПДИГ $\mathcal{D}_M = (\mathcal{A}, U)$ удовлетворяет условиям теоремы 2.

Лемма 1. *Для любого потока запросов H из $\mathcal{H}$, ПДИГ $\mathcal{D}_M$ функционирует без конфликтов.*

Доказательство. Запросы типа поиск, пустые запросы, а так же преобразования на перемещение по ИГ $R_7, \ldots, R_{12}$ не влияют на корректность функционирования ПДИГ, так как не преобразуют окрестности. Поэтому чтобы доказать корректность функционирования ПДИГ достаточно показать, что запросы на вставку и удаления не образуют конфликтов.

Докажем, что если нет запросов на удаление, то конфликтов не возникает. Рассмотрим два автомата с запросом на вставку. Каждый такт расстояние между ними не меньше двух ребер. Докажем это утверждение. Действительно рассмотрим первый автомат на вставку, который применил преобразование R_7. Это единственный случай, когда автомат продолжит функционирование. Если в следующий такт поступит еще один запрос на вставку, то автомат его обслуживающий, будет находиться в корне. Расстояние между эти автоматами будет ровно два ребра. Это расстояние не сможет уменьшиться, так как автоматы на вставку имеют одинаковый алгоритм. При этом если один из автоматов завершает свое функционирование, то расстояния между автоматами не будет. Следовательно утверждение доказано. Аналогично можно доказывается, что расстояние между любыми автоматами на поиск и вставку не меньше двух. в дальнейшем будем говорить, что выполняется свойство удаленности, если расстояние между любыми двумя работающими автоматами не меньше двух ребер.

Как следствие мы получаем, что одновременно не могут произойти два преобразования на вставку R_2 или R_3. А это означает, что преобразования на вставку не вызывают конфликтов. Пример передвижения нескольких автоматов на вставку изображен на рисунке 2.9.

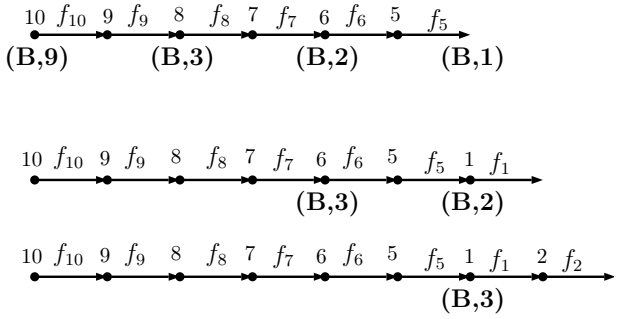


Рис. 2.9: Пример нескольких вставок.

Пусть имеется один запрос на удаление. Покажем, что при этом так же не возникнет конфликтов. Запрос на удаление повторяет алгоритм вставки до того момента как найдет искомую запись. Следовательно если автомат на удаление не найдет

искомую запись, то конфликтов не возникнет. При этом будет сохранено свойство удаленности. Пусть существует запись, которую он должен удалить. Преобразование R_4 удаляет только ребро и запись и не удаляет вершину в которой находиться. Поэтому это не вызовет конфликта с автоматом, который в следующий такт должен был перейти в эту вершину.

Осталось разобрать два внутренних случая. Это преобразование R_5 или R_6. Рассмотрим первый случай, когда автомат применил преобразование R_5. Это преобразование удаляет элемент списка. При этом в эту вершину не мог придти не один автомат. Докажем это.

Действительно, до момента удаления алгоритм удаления совпадал с алгоритмом на вставку (поиск). Как уже было доказано выше расстояние между любыми автоматами на вставку и поиск не меньше двух ребер. Следовательно ближайшие автоматы к рассматриваемому, находятся не меньше чем на расстоянии в 2 ребра. Поэтому в удаляемою вершину в следующий такт не придет ни один автомат. Следовательно это преобразование не вызывает конфликт. Так же это преобразование не нарушает свойство удаленности. При этом после этого преобразования расстояние между запросами, пришедшие до и после рассматриваемого, обязательно будет не меньше трех ребер. Пример передвижения нескольких автоматов на вставку и одного удаления R_5 изображен на рисунке 2.10.

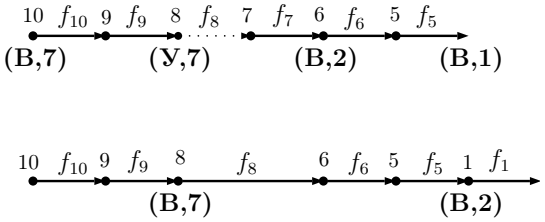


Рис. 2.10: Пример нескольких вставок и одного удаления R_5.

Рассмотрим теперь второй вариант, когда автомат применил преобразование R_6. Как и в предыдущем случае автоматы находятся на расстоянии не меньше чем два ребра друг от друга. Поэтому в следующий такт в этой вершине мог оказаться автомат, который перешел в нее только по преобразованию R_7. Кроме того в вершине $\mathbf{v}_2$ не может находиться автомат, а следовательно и применяться преобразование. Преобразование R_6 удаляет ребро, инцидентное вершине $\mathbf{v}_1$, при этом саму вершину не удаляет. Так же преобразование удаляет вершину $\mathbf{v}_2$, а инцидентное ей ребро, присваивает вершине $\mathbf{v}_1$. Следовательно это преобразование не вызывает конфликтного изменения одного и того же участка памяти. Кроме того это преобразование как и преобразование R_5 сохраняет свойство удаленности. Причем аналогично R_5 расстояние между запросами до и после, рассматриваемого, будет не меньше трех ребер. Пример передвижения нескольких автоматов на вставку и одного удаления R_6 изображен на рисунке 2.11.

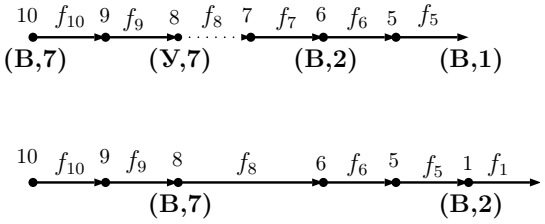


Рис. 2.11: Пример нескольких вставок и одного удаления R_6.

Итак мы доказали, что одно удаление не вызывает конфликтов. Рассмотрим теперь случай, когда одновременно обрабатываются два запроса на удаление. Аналогично случаю с одним удалением, алгоритм удаления совпадают с алгоритмом на поиск и вставку, пока не будет найдена искомая запись. Если хотя бы один из автоматов на удаление не найдет искомую запись, то этот случай равносилен разобранному выше случаю с одним удалением.

Рассмотрим все сценария функционирования ПДИГ, в случае двух запросов на удаление. Будем считать, что имеется два запроса на удаление $У_1$ и $У_2$, и запрос $У_1$ поступил раньше запроса $У_2$. Тогда, как уже было доказано выше, между этими запросами будет расстояние не меньше двух ребер. Если запрос на удаление $У_2$ завершает свое функционирование раньше, чем запрос $У_1$, то этот случай равносилен случаю с одним удалением. Если запрос $У_1$ применил преобразование, а $У_2$ в этот такт не удаляет запись, то его можно считать запросом на поиск. Этот случай был разобран выше. Остался последний вариант когда оба запроса на удаление одновременно удаляют записи. Если между ними был запрос на поиск или вставку, то это случай был разобран выше. Если же нет, то минимальное возможно расстояние между этими запросами два ребра.

Если $У_1$ применил преобразование R_4, то оно не вызовет конфликта. Осталось рассмотреть четыре варианта. Понятно, что достаточно рассмотреть случай минимального возможного расстояния между автоматами в два ребра.

Первый случай — оба автомата применили преобразование R_5. Преобразование R_5 удаляет вершину справа и меняет информацию о ребре для вершины в которой находиться автомат. При этом это преобразование не меняет информацию во второй справа вершине в которой находится другой автомат. Следовательно конфликтов не возникает. Пример для этого случая изображен на рисунке 2.12.

Второй случай — $У_2$ применил преобразование R_5, а $У_1$ применил преобразование R_6. Преобразование R_5 не меняет информацию в которой находится другой автомат. При этом автомат $У_1$ меняет информацию только в вершине в которой находиться. Следовательно конфликтов не возникает. Пример для этого случая изображен на рисунке 2.13.

Третий случай — $У_2$ применил преобразование R_6, а $У_1$ применил преобразование R_5. В этом случае так же не возникает конфликтов. Преобразования меняют инфор-

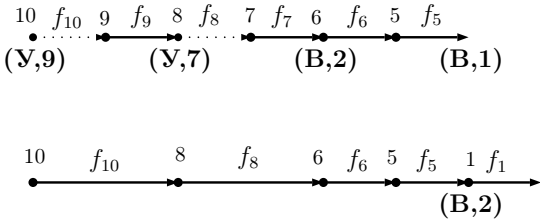


Рис. 2.12: Пример двух удалений. Случай R_5, R_5.

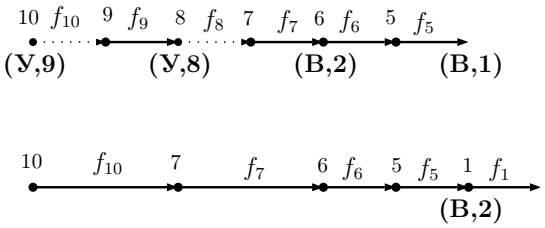


Рис. 2.13: Пример двух удалений. Случай R_5, R_6.

мацию в вершине в которой находятся, при этом изменяемые ребра не пересекаются. Пример для этого случая изображен на рисунке 2.14.

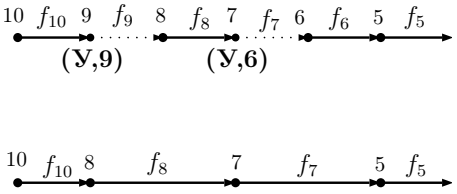


Рис. 2.14: Пример двух удалений. Случай R_6, R_5.

В последнем четвертом случае, когда оба автомата применили преобразование R_6, также не возникает конфликтов. Пример для этого случая изображен на рисунке 2.15 (преобразования (У,5) и (У,3)). Следовательно мы доказали, что в случае двух удалений конфликтов не возникает.

Рассмотрим общий случай, когда удалений может быть больше двух. Рассмотрим произвольный запрос на поиск или вставку. Автоматы обслуживающие эти запросы не конфликтуют с удалениями. Действительно если перед автоматом на вставку или поиск идет запрос на удаление, то конфликтов не возникает в силу разобранного выше случая с одним удалением. Таким образом если мы докажем, что три и более подряд удалений не конфликтуют, то мы докажем лемму.

Рассмотрим автомат на удаление, поступивший в такт i. Если в такт $i+1$ или в такт $i-1$ не поступали запросы на удаление, то этот случай был разобран выше, так как при этом не возникнет больше двух удалений подряд.

Рассмотрим теперь вариант, когда и в $i+1$ и в i такт поступили запросы на удаление. Конфликтов между запросами, поступившие в такты $i-1$ и i не возникает, так как этот случай был разобран выше. Аналогично для пары запросов, поступившие в такты $i+1$ и i. Очевидно, что между запросами, поступившими в такты $i-1$ и $i+1$ так же нет конфликтов. Следовательно среди трех удалений не возникает конфликтов. Аналогично доказывается, что для любого числа, идущих подряд удалений, не возникает конфликтов. Пример для этого случая изображен на рисунке 2.15.

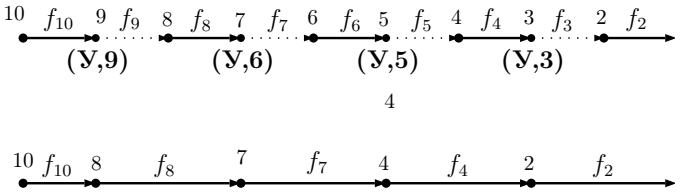


Рис. 2.15: Пример трех и более удалений.

Лемма доказана. □

Лемма 2. *Для любого потока запросов $H, H \in \mathcal{H}$, ПДИГ $\mathcal{D}_M$ функционирует корректно, кроме того то для любого натурального i выполняется*

$$T(\mathcal{D}_M, H(i)) \leq]|V(i,H)|/2[, \text{ если } W(H)(i) = \Pi.$$

Доказательство. Нужно показать, что для любого запроса на поиск $H(i), H(i) = (\Pi, x)$, ПДИГ выдает в ответ $\{x\}$, если $x \in V(i,H)$ и пустое множество в противном случае.

Из леммы 1 в частности следует, что любые преобразования сохраняют свойство удаленности для любого потока запросов H из $\mathcal{H}$. Другими словами автоматы идут друг за другом в порядке появления запроса. Рассмотрим произвольный запрос на поиск, поступивший в i такт.

Рассмотрим преобразование на вставку (удаление) поступившее в такт $j, 1 \leq j \leq i-1$ и которое еще не завершило свое функционирование. Так как ПДИГ сохраняет свойство удаленности, то в стадии поиска между ними будет расстояние $i - j \geq 2$ ребра, следовательно автомат на вставку (удаление) успеет вставить (удалить) запись прежде чем автомат на поиск сможет сократить расстояние.

Поэтому ПДИГ работает корректно так как любое преобразование завершиться не меньше чем за такт, до того как запрос на поиск их достигнет. Следовательно корректность доказана.

Из корректности ПДИГ $\mathcal{D}_M$, а так же его сохранения свойства удаленности следует, что автомат на поиск затратит не больше $]|V(i,H)|/2[$ тактов. Действительно свойство удаленности означает, что любой запрос на преобразование завершиться таким образом, что другие автоматы будут идти уже по обновленному ИГ, если они

поступили на такт позже. При этом за один такт автомат проходит сразу два элемента списка или находит искомый. Таким образом получаем оценку на сложность поиска

$$T(\mathcal{D}_M, H(i)) \leq]|V(i,H)|/2[.$$

Лемма доказана. □

Утверждение теоремы является следствием лемм 1 и 2.

□

2 ПДИГ, допускающий параллельную обработку произвольных потоков запросов с логарифмической сложностью

Пусть

$$f_a(x) = f_a^h(x) = \begin{cases} 1, & \text{если } x \leq a; \\ 0, & \text{иначе} \end{cases}; f_{=,a}(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{иначе} \end{cases};$$

$$f_a^{\text{П}}(x) = f_a^{\text{В}}(x) = f_a^{\text{У}}(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{иначе} \end{cases}; f^h(x) \equiv 0;$$

$$F = \left\{ f_a(x), f_a^h(x), f_{=,a}(x), f_a^{\text{П}}, f_a^{\text{В}}, f_a^{\text{У}} : a \in \mathbb{R} \right\} \cup \{ f^h \}.$$

Так же, будем рассматривать базовое множества преобразований $\mathcal{R}$, которое будет строго определено ниже, в доказательстве теоремы 3.

В качестве базового множества для ПДИГ будем рассматривать множество

$$\mathcal{F} = \langle F, \emptyset, \mathcal{R}, \eta_{id} \rangle. \tag{2.2}$$

Справедлива следующая теорема.

Теорема 3. *Существует конечный, селекторный ПДИГ типа (8,4) над базовым множеством $\mathcal{F}$, определяемым соотношением (2.2), который решает ДЗПИО для любого потока запросов $H, H \in \mathcal{H}$, со сложностью $L, L(n) = \left] \frac{\log_2 \max(n,2)}{3} \right[+ 1$.*

Доказательство. Структура доказательства будет следующей. Сначала будет описан алгоритм для 2-3-4 дерева в классической форме и на языке информационного-графа. Преобразования, которые будут применять автоматы, будут связаны с алгоритмами вставки и удаления в 2-3-4 дереве. Поэтому после описания алгоритма 2-3-4 дерева, опишем преобразования, которые будут применять автоматы. Далее покажем, что эти преобразования не вызывают конфликтов, обусловленных эксклюзивной записью, и в заключении оценим сложность полученного алгоритма.

Опишем сначала алгоритм 2-3-4 дерева. Структура данных 2-3-4 дерево является частным случаем B-деревьев. В [27] был приведен ДИГ, который поддерживал базу данных, решая ДЗПИО, для одиночных запросов. Алгоритм ДИГ основывался на структуре данных 2-3 дерево. Для потоковых запросов необходимо, чтобы вставка и удаление могли осуществляться за один нисходящий проход от корня к листьям. Поэтому здесь используется 2-3-4 дерево.

B-дерево можно рассматривать как иерархический индекс. Корень B-дерева является индексом первого уровня. Каждый нелистовой узел на B-дереве имеет форму $(p_0, k_1, p_1, k_2, p_2, \ldots, k_n, p_n)$, где p_i является указателем на $i-$го сына, $0 < i \leq n$, а k_i — ключ, $1 < i \leq n$. Ключи в узле упорядочены, так что $k_1 < k_2 < \cdots < k_n$. Все ключи в поддереве, на который указывает p_0, меньше или равно k_1. В случае $1 < i < n$ все ключи в поддереве, на который указывает p_i, принадлежат полуинтервалу $(k_i, k_{i+1}]$. Все ключи в поддереве, на который указывает p_n, больше k_n. В случае 2-3-4 дерева, $n = 3$. Пример 2-3-4 дерева изображен на рисунке 2.16. На этом рисунке узлы изображены сокращенно, например, в корне должны быть еще пустые ячейки k_2, p_2, k_3, p_3.

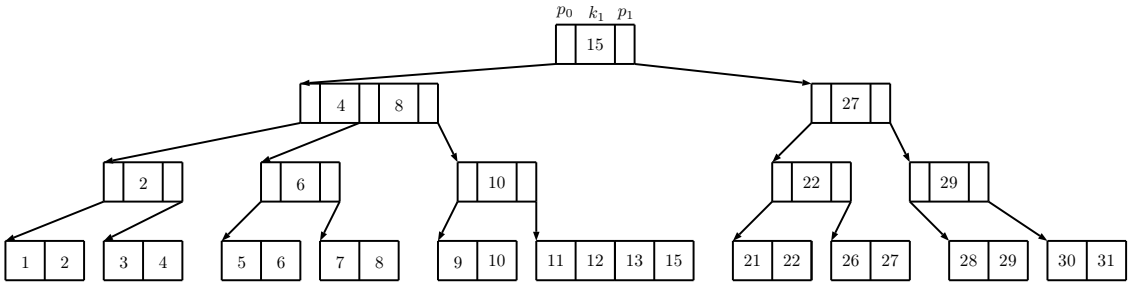


Рис. 2.16: Пример 2-3-4 дерева.

Поиск записи r с ключом x осуществляется по следующему алгоритму. Нужно проследить путь от корня к листу, который содержит r, если такая запись существует в базе данных. Каждый шаг вычисляется положение x в узле ключей. Если $x < k_0$ ($k_i \leq x < k_{i+1}$ или $x \geq k_n$), то на следующий шаг будет рассмотрен узел, на который указывает p_0 (p_i или p_n). Шаг, на котором x ищется в листе, — заключительный. B-дерево, все записи, которого хранятся в листовых вершинах, так же называют $B+$ дерево. Так же, в этой работе будем считать, что множество ключей совпадает с множеством записей.

Существуют различные алгоритмы для вставки и удаления из B-дерева. Здесь, важны алгоритмы, которые позволяют вставить и удалить из дерева за один нисходящий проход. Такой алгоритм для вставки существует [13], но для удаления автору работы не удалось найти явную ссылку на такой алгоритм. На самом деле, его можно получить из уже существующих идей. Рассмотрим алгоритм, описанный в [13]. Этот алгоритм в некоторых случаях требует замены в узле ключа, для которого необходимо повторное прохождение от листа к этому узлу. Проблема заключалась в том, что было рассмотрено B-дерево, которое хранило записи в узловых вершинах. Если рассмотреть дерево, которое описано выше, так называемое $B+$ дерево (все данных хранятся в листьях), то можно сделать удаление за один проход от корня к листу. Удалось заметить, что, используя алгоритм удаления, описанный в [13], на $B+$ дереве, не обязательно совершать замену удаленного ключа в узле. Естественно, итоговое 2-3-4 дерево будет отличаться от 2-3-4 дерева, полученного в результате известного алгоритма. Но 2-3-4 дерево останется деревом, которое будет корректно работать. Тем самым, можно осуществить удаление из $B+$ дерева за один нисходящий проход, что является ключевым моментом в его распараллеливании, с помощью автоматов.

Напомним известный алгоритм вставки в 2-3-4 дереве, за один нисходящий проход, и, предлагаемый в данной работе, алгоритм удаления из 2-3-4 дерева, за один нисходящий проход.

Начнем со вставки. Алгоритм вставки в 2-3-4 дерево следующий. Он повторяет алгоритм поиска за некоторыми отличиями. Первое отличие заключается в том, что этот алгоритм требует, чтобы каждый узел который он посещает, не был полностью заполнен, то есть имел бы не более трех сыновей. Если узел полностью заполнен (имеет ровно 4 сына), то алгоритм делает перестройку. Допустим, вставка находится в корне, и он имеет ровно 4 сына. В этом случае создается новый корень, а узел разбивается на два новых. Пример такого преобразования изображен на рисунке 2.17.

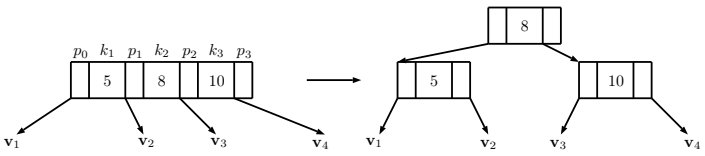


Рис. 2.17: Пример вставки в 2-3-4 дереве. Случай корня.

Далее по индукции можно считать, что родитель узла всегда не более трех сыновей. Рассмотрим теперь случай, когда вставка находится во внутреннем узле, и он имеет 4 потомка. В этом случае делается преобразование, изображенное на рисунке 2.18. Оно разбивает узел на два узла. Это можно сделать, так как отец рассматриваемого узла не полностью заполнен, следовательно, ему можно добавить еще одного сына.

Второе отличие алгоритма вставки от алгоритма поиска заключается в том, когда

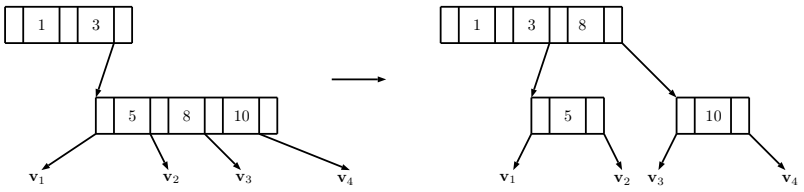


Рис. 2.18: Пример вставки в 2-3-4 дереве. Случай внутреннего узла.

вставка оказывается в листе и не находит запись. Если запись уже есть, то алгоритм завершается. Если же записи нет и лист не заполнен, то вставляется запись так, чтобы сохранить порядок ключей. Если же лист полностью заполнен, то делается перестройка, аналогичная перестройке в узле, описанной выше, и после этого запись вставляется в нужное место. Пример этого случая изображен на рисунке 2.19. Заметим, что этот алгоритм позволяет вставить новую запись в 2-3-4 дерево за один нисходящий проход.

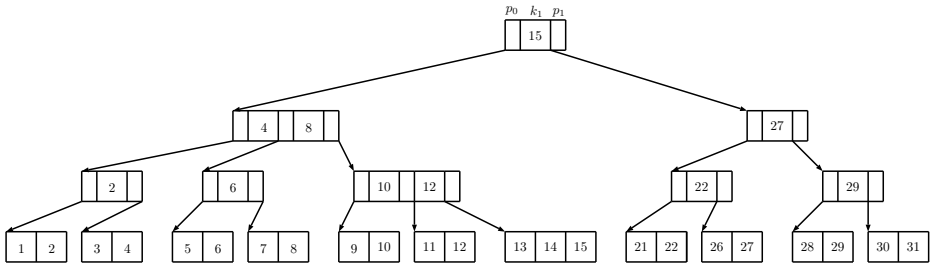


Рис. 2.19: Пример вставки в 2-3-4 дереве. Случай листа. Добавлена запись 14 к примеру 2-3-4 дерева, изображенного на рисунке 2.16.

Теперь опишем алгоритм удаления. Он, как и алгоритм вставки, повторяет алгоритм поиска за некоторым исключением. Первое отличие заключается в том, что этот алгоритм требует, чтобы в узле, в котором он находится, было не меньше двух ключей, то есть не меньше трех сыновей. Рассмотрим случай корня. Если у корня есть двое сыновей, каждый из которых содержит только по одному ключу (двое детей), то применяется преобразование, изображенное на рисунке 2.20. Оно удаляет корень, уменьшая при этом высоту дерева, и создает новый корень, объединяя ключи старого корня и его детей и записывая их в новый корень. Если же у одного из детей есть три сына, то алгоритм может применить преобразование переброски детей, изображенное на рисунке 2.21, если он должен перейти в узел, содержащий только один ключ. Если же на следующий шаг удаление переходит, в вершину, у которой не меньше трех сыновей, то преобразование не применяется.

Далее по индукции можно считать, что родитель узла содержит не менее трех сыновей. Если у рассматриваемого узла есть брат, который содержит не менее двух ключей (трех сыновей), то применяется преобразование переброски детей. Возможно,

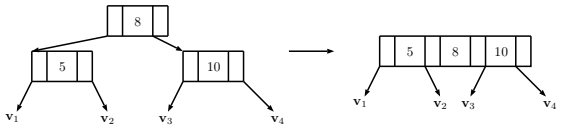


Рис. 2.20: Пример удаления в 2-3-4 дереве. Случай корня.

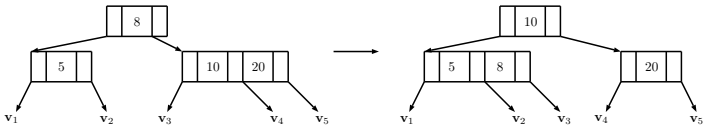


Рис. 2.21: Пример удаления в 2-3-4 дереве. Случай переброски детей. Это необходимо, если следующий шаг автомата на удаление оказывается в узле с одним ключом.

придется применить его не один раз, если братья не соседние. Разберем случай, когда все братья содержат ровно по два сына. Тогда применяется преобразование, изображенное на рисунке 2.22. Это преобразование объединяет двух братьев в один узел. Это можно сделать в силу того, что у отца есть не менее трех сыновей.

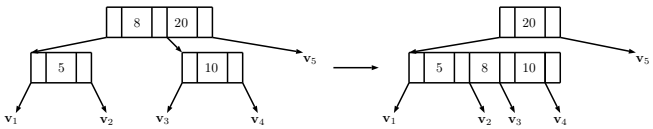


Рис. 2.22: Пример удаления в 2-3-4 дереве. Случай объединения братьев. Алгоритм переходит в вершину с ключом 5.

Второе отличие алгоритма на удаление от алгоритма на поиск заключается в последнем шаге алгоритма. Если алгоритм на удаление не находит запись, то он завершается. Если находит, то он удаляет ее, при необходимости применяя перестройки, описанные выше, чтобы сохранить структуру 2-3-4 дерева. Удаление записи 15 из дерева, изображенного на рисунке 2.16, представлено на рисунке 2.23. На этом рисунке изображено отличие от стандартного алгоритма удаления в 2-3-4 дереве за один нисходящий проход, описанное в [13]. Предлагается не заменять ключ 15, находящийся в корне на 13, а оставить его. Ключ 15 в корне не будет означать максимальный элемент в левом поддереве. Но это не мешает поиску правильно находить нужный запрос, так как в листьях идет проверка на равенство. Это наблюдение позволяет сделать алгоритм удаления за один нисходящий проход без исключений.

Для распараллеливания этого алгоритма на случай произвольного потока запроса переведем структуру 2-3-4 дерево на язык информационного-графа. Проведем следующую аналогию между классическим 2-3-4 деревом и информационным-графом. Узлу будет соответствовать предикатная вершина. Ключам в узле будут соответствовать предикатные ребра первого типа f_k. Пример соответствия между классическим

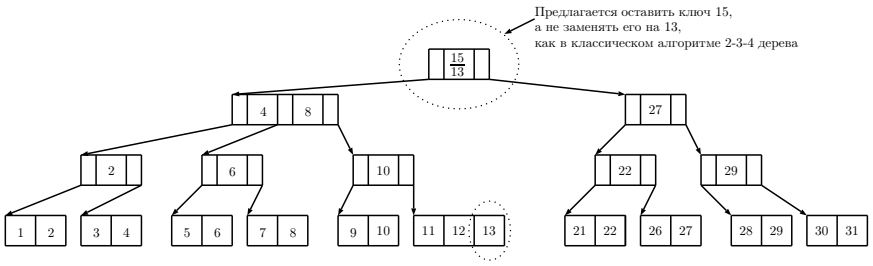


Рис. 2.23: Пример удаления в 2-3-4 дереве. Случай листа. Удаление записи 15 из примера, изображенного на рисунке 2.16. Отличие от стандартного алгоритма.

узлом в 2-3-4 дереве и его аналогом на информационном-графом языке, изображен на рисунке 2.24. Заметим, что самый правый предикат (f_{30}) означает, что максимальным элемент в правом поддереве 30. В классическом 2-3-4 дереве не хранится максимальный элемент. Можно было бы заменить f_{30} на предикат тождественно равный 1, но так удобнее объяснять преобразования, которые будут применять автоматы.

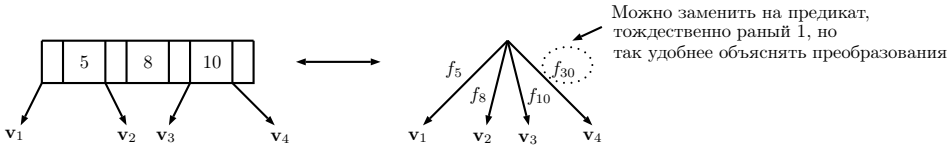


Рис. 2.24: Соответствие между узлом 2-3-4 дерева и его аналогом в информационном-графе.

Листьям 2-3-4 дерева, будут соответствовать пара предикатного ребра первого типа (f_k) и предикатного ребра второго типа ($f_{=,k}$). Напомним, что автомат знает не только значение предиката на запросе, но и его тип. Это возможно так как количество типов предикатов конечно. В данной теореме это количество равно 7. Это сделано для удобства описания преобразований с помощью автоматов. Пример соответствия между классическим листом в 2-3-4 дереве и его аналогом на информационном-графом языке, изображен на рисунке 2.25. Обратим внимание, что предикаты первого уровня для листа не обязательны, но это упростит преобразование для автомата, так как он сможет найти правильное место для вставки новой записи, используя эти предикаты. Под правильным понимается место, которое не нарушает порядок ключей в листе (от меньшего к большему).

Информационного-граф соответствующий примеру 2-3-4 дерева, изображенного на рисунке 2.16, представлен на рисунке 2.26. На нем не изображены предикаты второго типа, но их можно однозначно восстановить по записям, к которым они ведут.

Прежде чем перейти к описанию преобразований, которые будут использовать автоматы для перестроения ИГ, опишем их смысл. Один шаг алгоритма на поиск

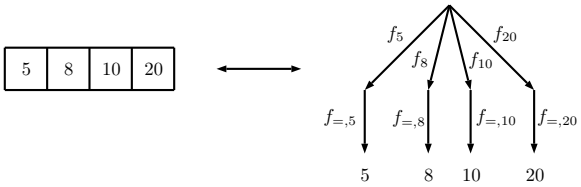


Рис. 2.25: Соответствие между листом 2-3-4 дерева и его аналогом в информационном-графе.

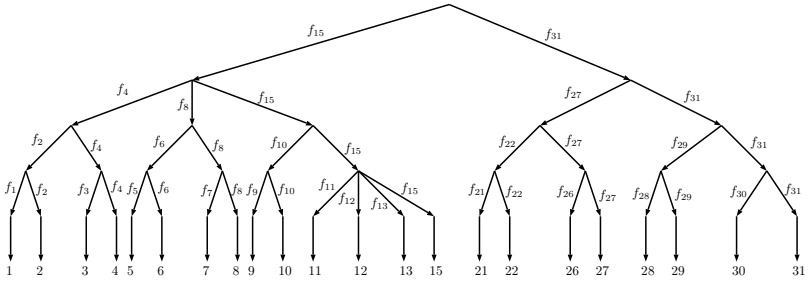


Рис. 2.26: Информационный-граф, соответствующий 2-3-4 дереву, изображенному на рисунке 2.16.

заключается в прохождении одного уровня дерева. При этом он переходит по самому левому ребру, код которого равен 1. Если таких ребер нет, то самым левым считаем самое правое ребро. Например для ИГ, изображенного на рисунке 2.26, автомат на поиск 15 пошел по левому ребру, а на поиск 30 пошел бы по правому ребру. Алгоритм преобразований основывается на классических перестройках 2-3-4 дерева, которые были описаны выше.

Например, рассмотрим автомат на вставку. Один шаг автомата на вставку аналогичен шагу автомата на поиск, если автомат переходит в вершину, у которой не более двух братьев. Если же нет, то автомат применят одно из преобразований 2-3-4 дерева, описанных выше, для его перестройки, а после переходит в эту вершину. Аналогично поступает автомат на удаление. Но если автоматы за один такт будут делать только один шаг алгоритма, то не удается избежать конфликтов, связанных с эксклюзивностью записи. Два автомата могут перестроить одну и ту же вершину или преобразовать одно и тоже ребро и так далее. Поэтому, чтобы избежать конфликтов, автомат за один такт функционирования будет делать, по возможности, 3 шага алгоритма. Отдельно разбираются случаи вблизи листьев, когда автомат должен завершить функционирование.

Не трудно увидеть, что если автомат каждый такт перемещается ровно на 3 уровня вниз, то расстояние между различными автоматами не меньше 3 ребер. Но при удалении корня может возникнуть конфигурация, в которой расстояние между автоматами станет 2 ребра. Разбор этих ситуаций будет произведен ниже. Заметим

только, что корень не может удалиться два такта подряд. Действительно, после удаления корня и перемещения автомата из корня будет выходить минимум 3 ребра. Поэтому корень не может быть удален два такта подряд.

В случае удаления корня могут возникнуть конфликты вблизи листьев. Чтобы их избежать, автоматом необходимо знать дополнительную информацию. А именно, автомат должен знать, есть ли в его окрестности другие автоматы, какие типы запросов они выполняют (поиск, вставка, удаление) и сравнить, совпадает ли обслуживаемый запрос с запросом других автоматов. Для получения этой информации, предназначены три специальных типа предикатов. Номера их типов: 4 ($f_a^{\text{П}}$), 5 ($f_a^{\text{В}}$) и 6 ($f_a^{\text{У}}$). В каждой вершине графа добавим две петли. Если в вершине нет ни одного автомата, то петлям будет присвоены предикаты 7-го типа f^h. Автоматы при переходе в новую вершин, будут изменять значение одной из петель, нагрузка которой есть f^h, на предикат одного из типов 4, 5 или 6, аргументом которого будет запрос, который обслуживает автомат. Если автомат был автомат на поиск (вставку или удаление), то тип предиката будет 4 (5 или 6). При уходе из вершины, автомат будет изменять предикат на петле, на f^h. Пример изменения нагрузок петель изображен на рисунке 2.27.

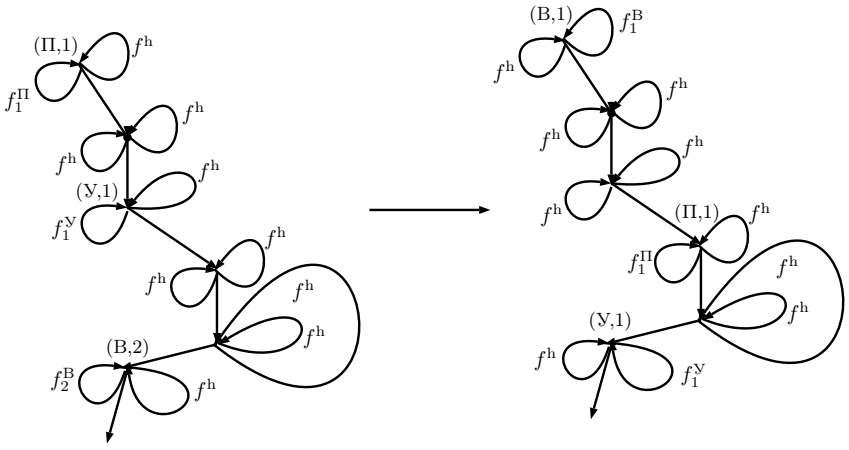


Рис. 2.27: Пример изменения нагрузок петель.

Заметим, что при изменении нагрузок петель конфликтов не возникает, так как у каждой вершины есть обязательно одна петля с нагрузкой f^h. Тем самым, автоматы не конфликтуют, изменяя нагрузки петель. Все предыдущие преобразования, на самом деле, содержат в каждой вершине по 2 петли. Это не было обозначено на рисунках, чтобы не загромождать их. Далее будем считать, что автомат знает информацию об наличии других автоматов в области своей видимости, и совпадают ли у них запросы или нет.

Зная эту информацию, некоторые автоматы могут закончить свое функционирование на первом такте. Будем говорить, что один автомат видит *впереди* другой

автомат, если тот находится ближе к листьям. Это может произойти, только если другой автомат начал свое функционирование ровно на 1 такт раньше. Аналогично скажем, что автомат видит другой автомат *сзади*, если тот поступил на 1 такт позже, и находиться в его области видимости.

Если автомат на поиск x видит впереди автомат на удаление x, то он может завершить свое функционирование с сигналом $e = 2$(запись не найдена). Так же, если автомат на преобразование видит впереди другой автомат с тем же заданием, то он может завершить функционирование с соответствующим сигналом. Если это автомат на вставку, то он завершает функционирование с сигналом $e = 3$ (запись уже есть), а если это автомат на удаление, то с сигналом $e = 2$ (запись не найдена). В случае, когда автомат на удаление(вставку) x видит сзади другой автомат на вставку(удаление) x, то он завершает свое функционирование.

Кроме этого автомату требуется завершить функционирование, если расстояние до записи не больше 4. Так же введены вспомогательные ребра f_a^h, которые нужны, чтобы связывать ИГ до и после преобразования. Это может понадобиться, например, в следующем ситуации. Два автомата идут друг за другом на расстоянии 3 ребра. Автомат, который начал функционирование позже, переходит в вершину, которую автомат впереди разбил на две. Следовательно, автомат, который был сзади должен иметь ребра ко всем сыновьям разбитой вершины.

Итак, начнем описывать преобразования, используемые автоматами. Формальных преобразований будет большое количество, поэтому мы опишем только смысл данных преобразований, приводя примеры из них.

Рассмотрим первый блок преобразований. Преобразование, изображенное на рисунке 2.28, соответствует случаю пустой базы данных.

формальное преобразование пример преобразования ИГ

Рис. 2.28: Вставка для случая пустой базы данных.

На рисунке 2.28 слева направо изображены соответственно простой шаблон $\mathcal{T}_1$, шаблон $\mathcal{T}_2$, ИГ U_1 (ИГ к которому применяем преобразование), и последним на рисунке изображен пример ИГ U_2. Здесь $M(\mathcal{T}_1) = \emptyset$, $p_L^0 \in \mathcal{P}_L \setminus M(\mathcal{T}_1)$, $I(p_L^0) = y$ (запрос на вставку), $r_J^1(p_L^0) = p_J^1$, $r_J^2(p_L^0) = p_J^2$, $r_L(p_L^0) = p_L^0$, где $I(p_J^1) = f_y$, $I(p_J^2) = f_{=,y}$.

Так же буква "к", находящаяся рядом с вершиной, обозначает корень, а вершина, обведенная кругом, означает центр рассматриваемой окрестности (текущее положение автомата $\mathcal{A}$), а так же новое текущее положение автомата $\mathcal{A}$, которое будет соответствовать компоненте β выходного символа $b \in B$ автомата $\mathcal{A}$. При этом, если

в правой части преобразования нет вершины, обведенной кругом, то это означает, что автомат завершает функционирование. Например, автомат на поиск завершает функционирование, когда он находит запись, или не находит, но при этом находиться в конце ИГ.

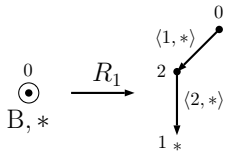


Рис. 2.29: Преобразование R_1 на вставку, с точки зрения автомата.

На рисунке 2.29, изображено преобразование R_1 с точки зрения автомата. Автомат на вход получает код ИГ на запросе $*$. Вершина, которой соответствует код 0, является корнем. Автомат дает инструкцию ИГ, что данную окрестность нужно заменить на окрестность, соответствующей правой части преобразования. В этой окрестности есть два предикатных ребра. Одному ребру присвоен предикат первого типа $\langle 1, * \rangle$, который будем интерпретировать как f_*, второму ребру присвоен предикат второго типа $\langle 2, * \rangle$, который будет интерпретировать как $f_{=,*}$.

В дальнейшем будем описывать все преобразования с точки зрения автомата. Так же будем использовать сокращенную запись преобразования. В преобразованиях не будем указывать код вершин, а так же тип предикатов первого типа. При этом в преобразованиях будем выделять ребра жирным, если автомат не меняет это ребро. Ребро будем изображать пунктиром, если автомат удаляет или вставляет это ребро. Это поможет понять, какие ребра автомат преобразовывает, а какие ребра нет.

В случае пустой базы данных для автоматов на поиск и удаление применяется преобразование R_2, изображенное на рисунке 2.30. В этом случае автомат, обслуживающий запрос, завершает функционирование с сигналом $e = 2$ (запись не найдена).

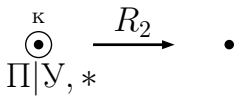
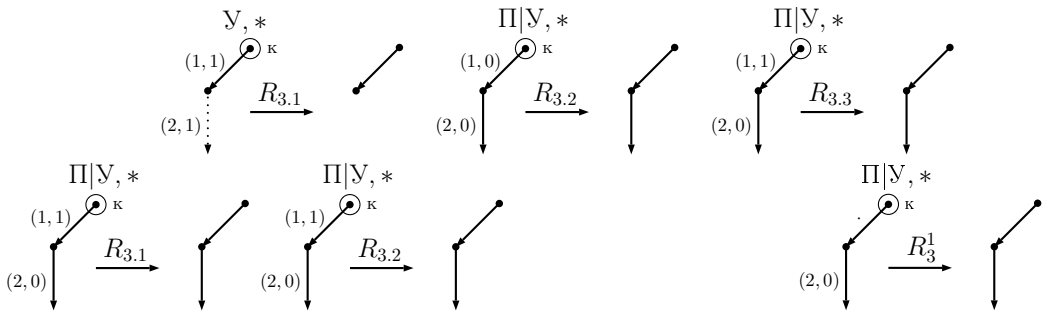


Рис. 2.30: Преобразование R_2.

Если база данных состоит из одной записи, то применяются преобразования R_3, R_4, изображенные на рисунках 2.31 и 2.32. На этих рисунках ребрам приписаны код ребер на запросе, а именно первая координата соответствует типу предиката, а вторая - его значению на запросе. Например, ребро с кодом $(1, 0)$ означает, что в ИГ этому ребру соответствует предикат первого типа и его значение на запросе, который обслуживает автомат, равен 0. Преобразование $R_{3.1} \in R_3$ завершается с сигналом $e = 1$ (запись найдена). Преобразования $R_{3.2}, R_{3.3}$ завершаются с сигналом



обобщенное преобразование преобразование без первого типа

Рис. 2.31: Группа преобразований $R_3 = \{R_{3.1}, R_{3.2}, R_{3.3}\}$.

$e = 2$ (запись не найдена). В дальнейшем для ребер, которым соответствует предикат первого типа, будем применять сокращенную запись. А именно не будем писать тип предиката, а так же если его значение не важно 0 или 1, то будем писать точку на этом ребре. Пример сокращенных преобразований изображен на рисунке 2.31.

В преобразовании $R_{4.1} \in R_4$ есть ребро с кодом $(2, 1)$ — это означает, что вставляемая запись уже существует и ее не нужно вставлять.

В преобразованиях $R_{4.1}, R_{4.2}$ вставляемая запись не совпадает с уже имеющейся, поэтому к базе данных добавляется новая запись соответствующая запросу. При этом в зависимости от значения предиката первого типа запись вставляется либо справа, либо слева от уже имеющейся.

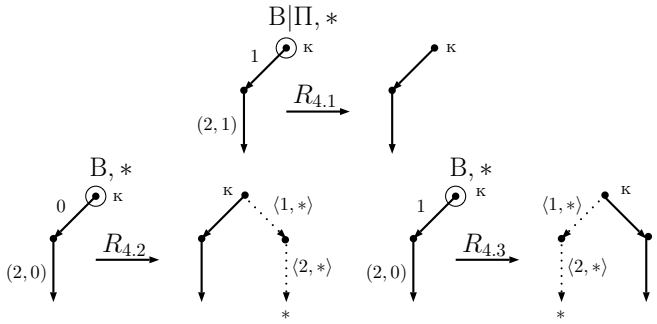


Рис. 2.32: Группа преобразований $R_4 = \{R_{4.1}, R_{4.2}, R_{4.3}\}$.

Для случая, когда в базе данных больше одной записи, рассмотрим по отдельности все преобразования на вставку, на удаление и на поиск.

Рассмотрим сначала все преобразования на вставку. Основная идея для вставки схематично изображена на рисунке 2.33. При этом заметим, что преобразование добавляет вспомогательные ребра третьего типа (f_a^h). Рассмотрим вспомогательное ребро и вершину из которой оно выходит. Оно соединяет ее с ребенком брата этой вершины. Нагрузка этого ребра является копией нагрузки ребра, которое идет от

брата рассматриваемой вершины, до этого ребенка. Эти ребра нужны, чтобы не нарушить алгоритм поиска автомата, который идет вслед за рассматриваемым. Сразу заметим, что если автомат видит по движению вспомогательные ребра, то он обязательно их стирает в следующий такт. Так как вспомогательное ребро является копией ребра ИГ, то количество таких ребер не превосходит количество ребер в ИГ, то есть его объем. Кроме того, если автомат видит вспомогательное ребро перед собой, то он обязательно его стирает. Следовательно, в любой такт времени количество вспомогательных ребер не превосходит объема ИГ.

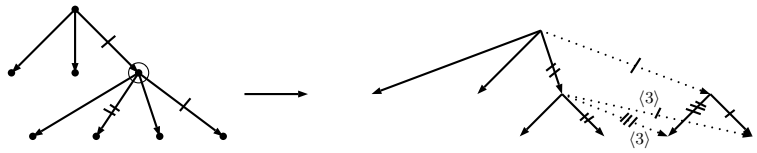


Рис. 2.33: Основная идея перестройки графа автоматом на вставку.

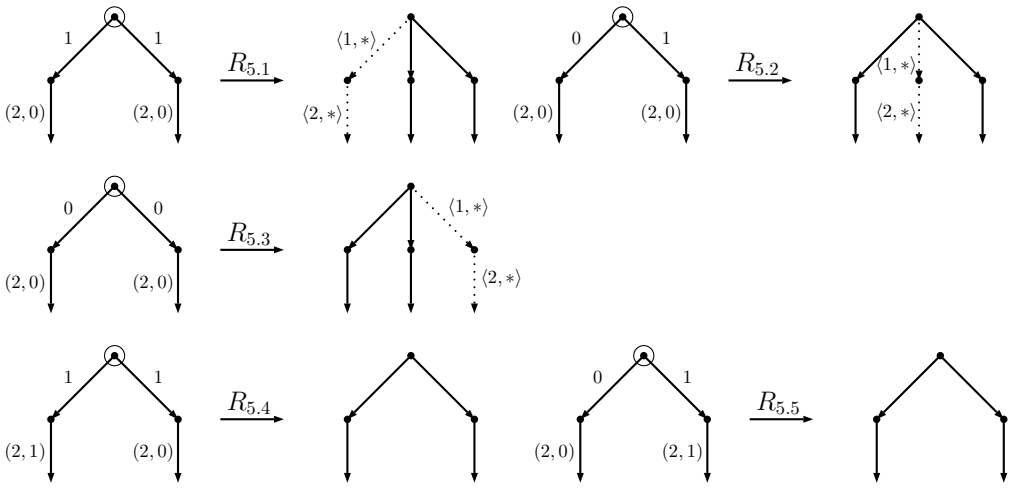


Рис. 2.34: Группа преобразований R_5

Итак, сначала опишем преобразования на вставку, которые применяет автомат, находясь на расстоянии в два ребра от записей. Группа преобразований $R_5, R_5 = \{R_{5.1}, R_{5.2}, R_{5.3}, R_{5.4}, R_{5.5}\}$, соответствует случаю вставки третьей записи и изображена на рисунке 2.34. Видно, что эта группа преобразований схожа с уже разобранной группой R_4. Преобразования $R_{5.4}$ и $R_{5.5}$ — это случаи, когда вставляемая запись уже есть.

Группа преобразований R_6 аналогична группе преобразований R_5. В этой группе вставка добавляет четвертую запись, если ее не было. Пример преобразования из этой группы изображен на рисунке 2.35.

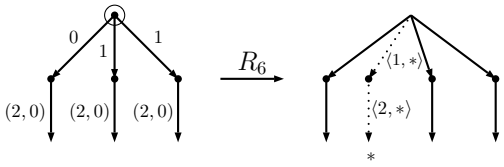


Рис. 2.35: Пример преобразования из группы R_6

Следующая группа преобразований R_7, образует новый корень (увеличивает высоту дерева). Условно можно сказать, что эта группа преобразований разбивает пятерку на сумму двойки и тройки. Преобразования из этой группы отличаются только тем, в какое по порядку место вставляется новая запись. Пример преобразования их этой группы изображен на рисунке 2.36.

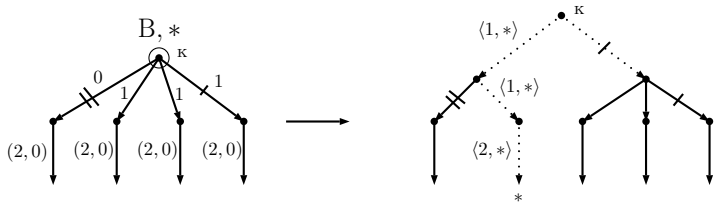


Рис. 2.36: Группа преобразований R_7.

Для дальнейшего описания преобразований на вставку введем несколько пояснений. Один шаг алгоритма на вставку заключается в прохождении одного уровня дерева. При этом он переходит по самому левому ребру, которому соответствует значение 1. Если таких ребер нет, то самым левым считаем самое правое ребро. Так же за один такт функционирования автомат делает три шага по дереву, если это возможно, или завершает функционирование с необходимым действием. Заметим, что автомат во время шага может перестроить ИГ при необходимости, применяя преобразование условно изображенное на рисунке 2.33. Основная цель этих преобразований состоит в том, чтобы автомат, оказавшийся в вершине, в которую нужно вставить запись (предпоследний уровень дерева), смог ее вставить и завершить функционирование.

Пример одного такта (трех шагов) автомата изображен на рисунках 2.37 и 2.38. На рисунке 2.37 изображен пример перехода автомат на 3 уровня вниз без применения перестроек. Автомат на вставку не применяет перестроек, так как он оказывается в вершине, у которой не более двух братьев. Заметим, что при этом, если автомат вставляет запись так, что она больше максимального элемента в поддереве, то автомат меняет нагрузку самого правового ребра, заменяя ее на предикат $\langle 1, * \rangle$. Он делает это при необходимости на предикате, который в первый шаг стоит на уровень выше, и не делает этого преобразования на третьем шаге. Это необходимо, чтобы избежать конфликтов между автоматами.

На рисунке 2.38 изображен пример одного такта функционирования автомата на

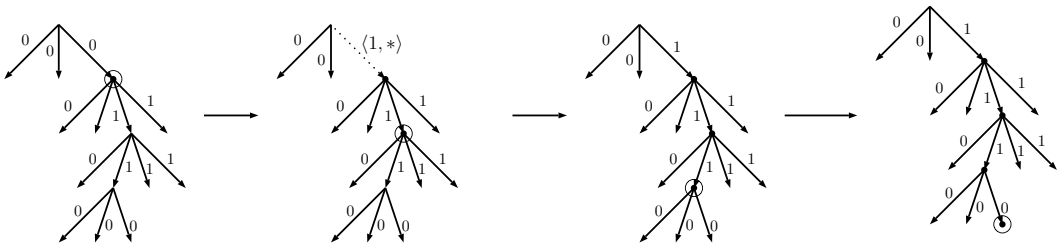


Рис. 2.37: Примера одного такта автомата на вставку, без применения перестроек.

вставку с применением перестроек. Заметим, что после преобразования добавляются вспомогательные ребра третьего типа. Смысл данных ребер был описан выше, заметим только, что вспомогательные ребра находятся только на уровне, откуда автомат начал применять свое преобразование. На других уровнях вспомогательные ребра не обязательны, поэтому автомат их не добавляет. Действительно, минимальное расстояние от автомата на вставку, до следующего за ним автомата, не меньше трех ребер. Поэтому нужно добавить вспомогательные ребра, только на уровень, на котором находится рассматриваемый автомат, так как другие уровни не будут влиять на автоматы, которые начали свое функционирование после него.

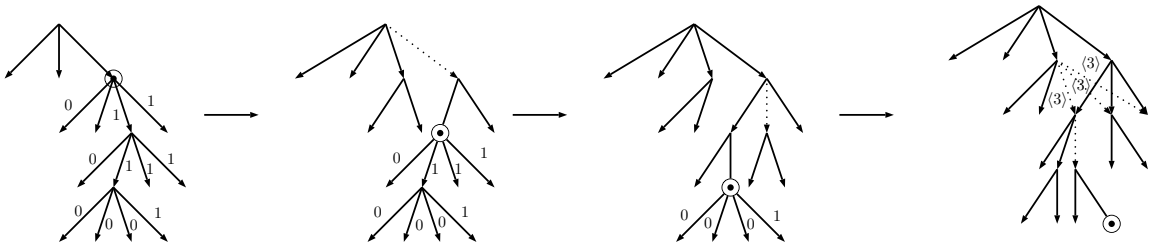


Рис. 2.38: Пример одного такта автомата на вставку, с применением перестроек.

Может возникнуть такая ситуация, что автомат на вставку окажется в вершине, у которой три брата. Это может произойти из-за того, что он применил преобразование без перестроек, в то время как автомат впереди мог применить перестройки на том уровне, в который перейдет рассматриваемый автомат. Для этого случая автомат, при необходимости, применяет преобразование, изображенное на рисунке 2.39. Рассмотрим вершину, в которой находится автомат. Необходимость в таком преобразовании возникает, если количество детей, у рассматриваемой вершины, равно 4. Заметим, что обязательно найдется брат у рассматриваемой вершины, у которого не более трех детей. Это не трудно заметить. Действительно, автомат мог оказаться в вершине, у которой три брата, только в том случае, если другой автомат, который идет впереди, применил преобразования перестройки. Видно, что после преобразования перестройки у родителя рассматриваемой вершины будут дети, у которых не более трех детей. На самом деле, у родителя рассматриваемой вершины, будут двое

детей, у которых в сумме не более 5 детей.

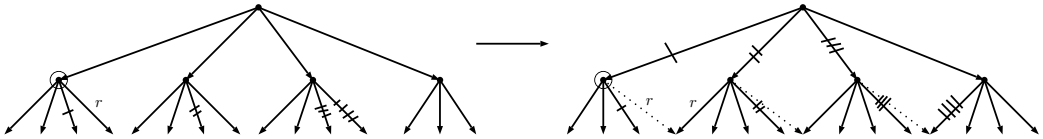


Рис. 2.39: Пример преобразования в случае трех братьев.

Разберем граничные случаи, когда автомат находиться на расстоянии 2,3 или 4 от записей. Автомат на вставку в этих случаях вставляет запись, если ее нет в базе данных, и завершает функционирования с нужным сигналом. Пример преобразования на вставку для расстояния 4 изображен на рисунке 2.40. По условию теоремы радиус видимости автомата 4, поэтому автомат сможет применить эти преобразования. Для оставшихся случаев - аналогично.

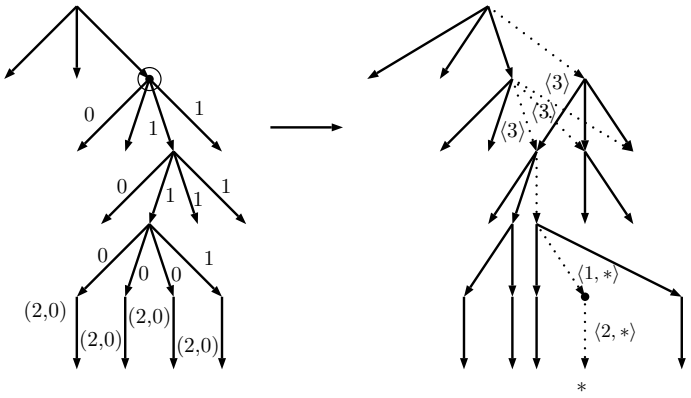


Рис. 2.40: Один такт автомата на вставку, когда расстояние до записей равно 4. Вставка записи и завершение функционирования.

Заметим, что при таком алгоритме не может возникнуть ситуация, когда автомат оказывается на расстоянии 1 от записей, так как за один такт до этого расстояние до записей было равно 4, и он должен был завершить функционирование. Это свойство будет использовано ниже для доказательства, что преобразования на вставку не вызывают конфликтов.

Итак, основные преобразования на вставку описаны. Дополнительные преобразования на вставку опишем ниже. Эти дополнительные преобразования будут введены для избежания конфликтов с преобразованиями на удаления.

Опишем теперь основные преобразования на удаления. Алгоритм автомата на удаление будет схож с алгоритмом автомата на вставку. За один такт автомат на удаление делает по возможности 3 шага, спускаясь на 3 уровня вниз. Автомат на удаление делает перестройки, чтобы выполнялось свойство братьев вершины, в которой

он будет находиться. А именно, чтобы количество братьев было не меньше двух. Основная идея преобразования на удаление изображена на рисунке 2.41. На нем видно, что автомат, переходя на один уровень вниз, увеличивает количество братьев, объединяя сыновей соседних вершин. Если у брата рассматриваемой вершины есть три сына и более, то автомат перекидывает одного из них, делая его сыном вершины, в которой он стоит. Если же у брата вершины, в которой стоит автомат, тоже только два сына, то автомат объединяет этих двух братьев, делая из них новую вершину, у которой четыре ребенка. При этом оставляет вершину брата и добавляет вспомогательные ребра третьего типа. Это необходимо делать, чтобы избежать некорректной работы автоматов, которые начали функционирование позже.

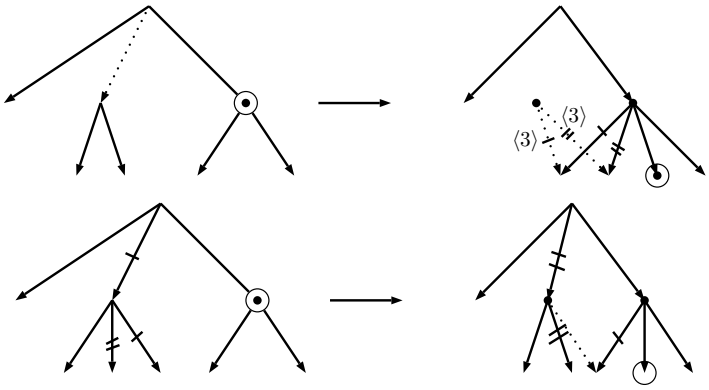


Рис. 2.41: Основная перестройка графа, автоматом на удаление.

Аналогично алгоритму работы автомата на вставку, автомат на удаление не применяет перестройки графа, если он после трех шагов переходит в вершину, у которой не менее двух братьев.

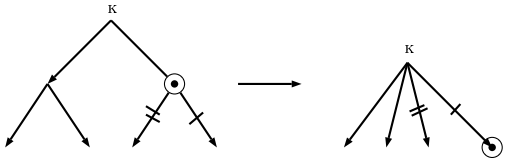


Рис. 2.42: Преобразование на удаление корня.

Возможна ситуация, когда автомат на удаление оказывается в вершине, у которой всего один брат. Если автомат находиться в корне, у которого два сына, то он применяет преобразование на удаление корня. В этом случае высота дерева уменьшается на один. Пример преобразования на удаление корня изображен на рисунке 2.42. Естественно, что вершину, являющуюся корнем, автомат не удаляет, так как туда может придти новый автомат. Это преобразование удаляет сыновей корня, а всех детей сыновей корня делает детьми корня.

Разберем другой возможный случай, когда автомат оказывается во внутренней вершине, у которой только один брат. Это могло произойти только тогда, когда другой автомат на удаление применил преобразование графа на этой вершине. Действительно, если бы никто не применил преобразования на этой вершине, то она бы не изменилась, и автомат перешел бы в вершину, у которой не менее двух братьев, или он сам бы перестроил ее, если у нее был бы только один брат.

Допустим, что автомат оказался в вершине, у которой только один брат. В этом случае сумма детей у рассматриваемой вершины и ее брата не меньше 5. Это следует из того, что такт назад в этой вершине была перестройка другим автоматом на удаление, который не мог сделать сумму детей меньше, чем $2 + 3 = 5$. Следовательно, автомат может перераспределить детей так, чтобы в вершине, в которую он собирается перейти, было не менее трех детей. Тем самым граф уже на втором шаге автомата будет удовлетворять предположению, что автомат на удаление находится в вершине, у которой не меньше двух братьев. Пример преобразования перераспределения детей можно увидеть на нижнем преобразовании из рисунка 2.41, если считать, что из вершины выходит не 3 ребра, а только 2.

Как и в случае со вставкой, автомат на удаление, находясь на расстоянии 2, 3 или 4 ребра от записей, в следующий такт завершает функционировании с нужным сигналом. Так же автомат на удаление не может оказаться на расстоянии один от записей. Это свойство доказывается аналогично случаю вставки. Пример преобразования на удаление, для расстояния 4, изображен на рисунке 2.43. На этом рисунке нарисованы не все коды ребер. Так же, на нем условно обозначены вспомогательные ребра через $\langle 3 \rangle$. Смысл этих ребер был описан ранее. Напомним, что они служат для корректной работы других автоматов. В данном случае, эти ребра нужно добавлять на 2 яруса, так как удаление могло удалить корень, и следовательно, следующий за ним автомат мог попасть на один из этих уровней. Для оставшихся случаев - аналогично.

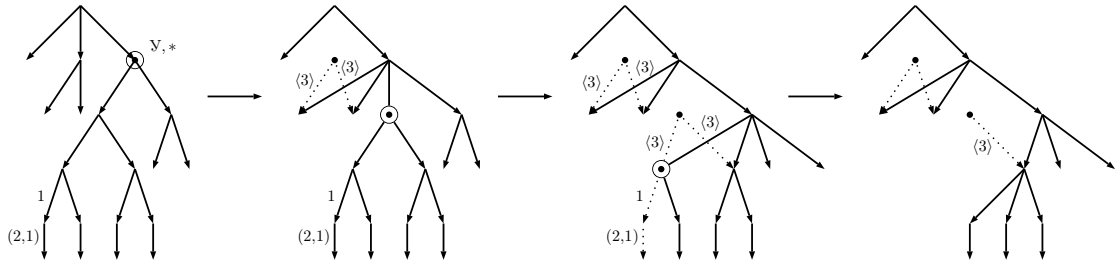


Рис. 2.43: Один такт автомата на удаление, когда расстояние до записей равно 4. Удаление записи и завершение функционирования.

Итак, осталось рассмотреть случаи, когда автоматы выполняют разные задания и видят друг друга.

Рассмотрим всевозможные пары автоматов, которые видят друг друга. Заметим

сначала, что стирания вспомогательных ребер, нагрузка которых есть предикаты третьего типа (f_a^h), не влияют на конфликтность. Поэтому автоматы на поиск не влияют на конфликтность преобразований. В дальнейшем не будем обращать внимания на вспомогательные ребра, так как их удаление не будет влиять на конфликтность.

Прежде чем приступим к разбору случаев, оценим возможные расстояния между автоматами. Заметим, что автомат может изменить высоту дерева только в первый такт своего функционирования. Причем автомат на вставку может увеличивает высоту дерева ровно на 1, а автомат на удаление может уменьшить высоту дерева ровно на 1. Начиная со второго такта, все автоматы перемещаются либо на 3 уровня вниз, либо завершают функционирование, поэтому либо один из автоматов завершает функционирование, либо расстояние между ними сохраняется и равно либо 2 ребра, либо 3 ребра. Расстояние в 2 ребра может быть только между автоматом на удаление, который удалил корень, и следующим за ним автоматом. Если был удален корень, то расстояние между автоматом на удаление A_1 и следующим автоматом A_2 будет 2 ребра, которое не будет меняться. При этом, расстояние между автоматом A_2 и следующего за ним автоматом будет обязательно не меньше 3 ребер. Это было доказано выше, когда разбирался случай, что корень не может быть удален два такта подряд. Расстояние между любыми автоматами на вставку всегда не менее трех.

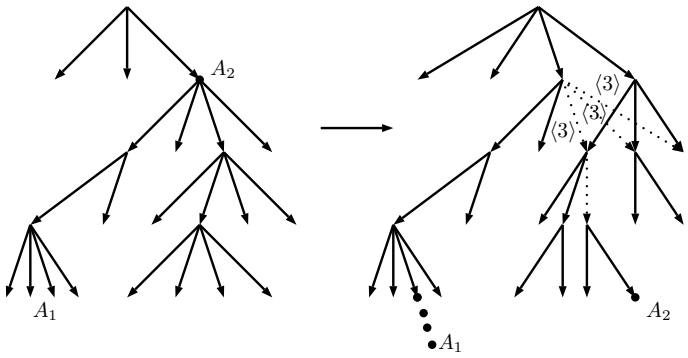


Рис. 2.44: Первый пример не конфликтности двух автоматов на вставку.

Итак, разберем первый случай. Два автомата на вставку видят друг друга. Как было показано выше, расстояние между ними не меньше трех ребер. Будем считать, что между ними ровно 3 ребра. Пусть это автоматы A_1 и A_2, причем автомат A_1 начал свое функционирование на такт раньше.

Рассмотрим первый случай. Если автомат A_2 не идет в вершину, в которой стоит автомат A_1 или к ее братьям, то конфликта не возникнет. Докажем это. Автомат A_1 преобразует только те вершины, которые находятся на его уровне и, может быть, преобразует еще и родительскую вершину. Автомат A_2 не преобразует эти вершины, так как идет в другие. Поэтому в этой ситуации конфликтов не возникнет. Пример этого случая изображен на рисунке 2.44.

Пусть автомат A_2 идет в вершину, где стоит автомат A_1, или идет к ее братьям. Пусть при этом автомат A_1 находится в вершине, у которой не больше двух братьев. Тогда автомат A_2 не применяет преобразования на перестроение и не меняет значение предикатов на третьем шаге. Тем самым конфликтов не возникает. Пусть автомат A_1 находится в вершине, у которой ровно 3 брата. Как было разобрано выше, автомат A_1 применит преобразование для случая трех братьев, изображенное на рисунке 2.39. Это преобразование затрагивает только вершину, в которой стоит A_1, и ее братьев, и, возможно, изменяет значение предикатов у ребер исходящих из родителя. Таким образом, автомат A_2 применит преобразования с перестройкой, которое не будет конфликтовать с преобразованием A_1, так как на последнем третьем шаге, автомат A_2 только перераспределяет ребра, но не меняет значения на них. Пример этого случая изображен на рисунке 2.45.

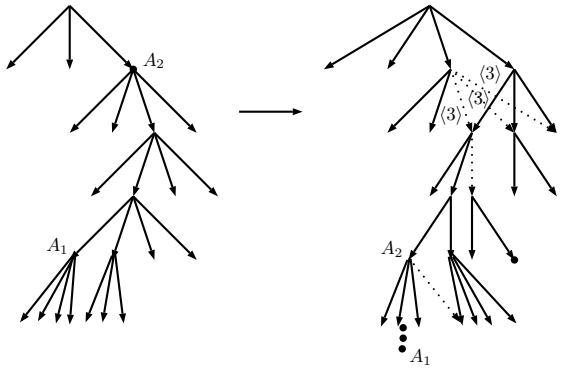


Рис. 2.45: Второй пример не конфликтности двух автоматов на вставку.

Аналогичные случаи, когда автомат A_1 находиться рядом с записями, на расстоянии не более, чем 4 ребра, уже разобранным. Заметим, только что автомат не может находиться на расстоянии 1 от записей. Это было показано выше. Для большей ясности, что конфликтов не возникнет, приведен пример на рисунке 2.46.

Итак, мы показали , что автоматы на вставку не вызывают конфликтов. Теперь разберем неконфликтность двух удалений подряд. Первый случай, расстояние между автоматами на удаление 3 ребра.

Если расстояние между автоматами на удаление равно 3, то неконфликтность двух преобразований на удаление доказывается схоже с доказательством неконфликтности преобразований на вставку. Действительно, пусть даны два автомата на удаление A_1 и A_2, и автомат A_1 начал свое функционирование на 1 такт раньше A_2.

Для простоты объяснения, будем считать, что автомат $A_1(A_2)$ находится в вершине $v_1(v_2)$ на первом шаге своего такта.

Первый случай, когда автомат A_2 переходит в вершину, которая не совпадает с v_1 и не является ее братом. В этом случае автоматы не конфликтуют. Пример, иллю-

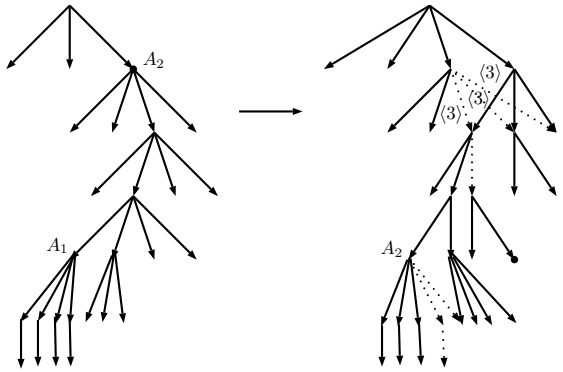


Рис. 2.46: Третий пример неконфликтности двух автоматов на вставку. Автомат A_1 вставил запись и завершил функционирование.

стрирующий этот случай, изображен на рисунке 2.47. На нем изображены два овала. Овал, который находиться ниже уровнем, чем вершина v_1, в которой находиться A_1, включает вершины и ребра, которые возможно будет преобразовывать A_1 на пером шаге функционирования. Другой овал показывает, какие вершины и ребра будет затрагивать автомат A_2 на третьем шаге своего такта функционирования. Видно, что эти преобразования не конфликтуют.

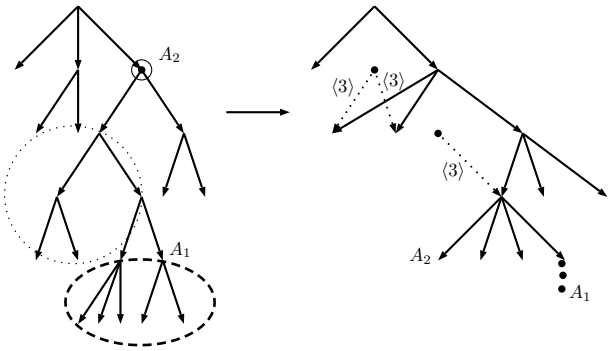


Рис. 2.47: Первый пример неконфликтности двух автоматов на удаление.

Второй случай, когда автомат A_2 переходит в вершину v_1, в которой находится A_1, или к ее братьям. При этом либо вершина v_1, либо существует ее брат, который имеет не менее трех потомков. Доказательство этого случая также схоже с уже разобранным случаям для вставок. Если автомат A_1 находится в вершине, у которой как минимум 2 брата, то автомат A_2 не применяет преобразований, и, следовательно, конфликтов не возникает. Если же автомат A_1 находится в вершине, у которой ровно один брат, то он обязательно применит преобразование перераспределения детей, изображенное на рисунке 2.41. Чтобы не загромождать доказательство рисунками, пример, иллюстрирующий этот случай, можно увидеть на рисунке 2.47. Достаточно

только заменить положение автомата A_2, на положение, относящееся к этому случаю.

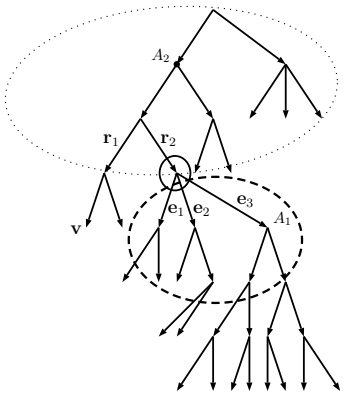


Рис. 2.48: Конфигурация при которой возможен конфликт, при использовании стандартного алгоритма. Автомат A_2 должен перейти в вершину $\mathbf{v}$.

Третий случай, если автомат A_1 находится в вершине v_1 у которой два(три) брата, но суммарное количество сыновей и племянников, у этой вершины, равно 4(6). В этом случае автоматы A_1 и A_2 уже не могут действовать по стандартному алгоритму, так как может возникнуть конфликт в вершине, которая является родителем v_1. Пример этого случая изображен на рисунке 2.48. Автомат A_2 переходит в вершину v, и он будет вынужден совместить ребра $\mathbf{r}_1$ и $\mathbf{r}_2$. Автомат A_1 совмещает ребра $\mathbf{e}_2$ и $\mathbf{e}_3$. Конфликт возникает в вершине, которая является родителем v_1(обведена овалом, граница которого непрерывна).

Прежде чем начать разбирать алгоритм действия автоматов в этой конфигурации, рассмотрим причину возникновения конфликта и разберем конфигурации, при которых автоматы будут действовать стандартно.

Если у вершины v два и более братьев или входная окрестность автомата A_1 такая же, как и на рисунке 2.49, то автоматы действуют стандартно. Случай, когда у вершины v два и более братьев очевиден, так как автомат A_2 не делает перестроек графа. Случай, изображенный на рисунке 2.49, означает, что автомат A_2 совместит ребра $\mathbf{r}_1$ и $\mathbf{r}_2$, но это не вызовет конфликта, так как при этом не будет задействован родитель вершины v_1.

Итак, разберем алгоритм автоматов для случая, изображенного на рисунке 2.48. Чтобы избежать конфликтов, автомат A_1 поменяет первый шаг своего такта, а автомат A_2 поменяет третий шаг своего такта. Заметим, что другие шаги автоматов не могут создать конфликтов.

Рассмотрим конфигурацию, когда автомат A_2 готов сделать третий шаг, а автомат A_1 делает первый шаг. Пример изображен на рисунке 2.50. На этом рисунке изображено три преобразования. Первое преобразование, обозначенное стрелкой A_1, соответствует первому шагу автомата A_1. Он соединяет ребра $\mathbf{e}_1$ и $\mathbf{e}_2$, делая из них $\mathbf{e}_1^2$, а так же помечает вместо ребра e_2, делает вспомогательное ребро. Это ребро

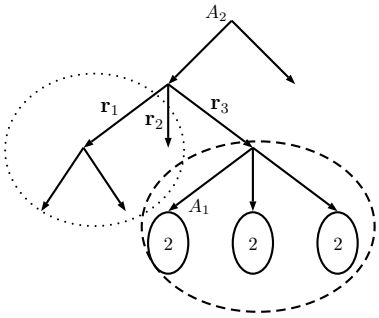


Рис. 2.49: Пример, когда автоматы на удаления A_1 и A_2 действуют стандартно. Число в овале означает количество ребер.

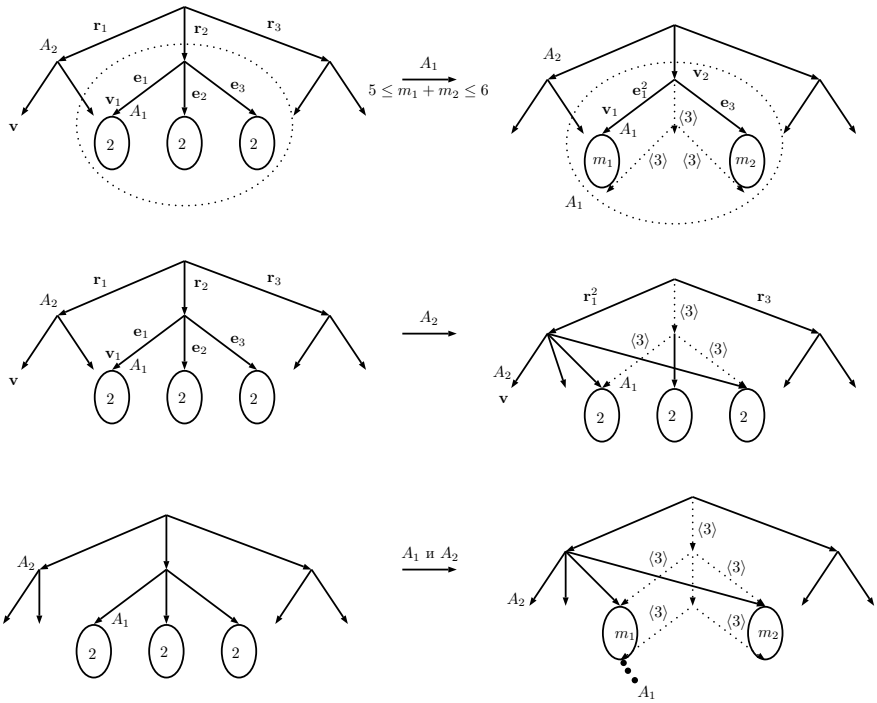


Рис. 2.50: Преобразование автомата A_1 и A_2, отличающиеся от стандартных, для избежания конфликта.

нужно оставить, чтобы не взывать конфликта, в этом и заключается отличие первого шага автомата A_1. Кроме того, автомат A_1 обязательной делает в первый шаг перестройку, даже если это не нужно было в стандартном алгоритме. Например, A_1 переходит в вершину, у которой не меньше двух братьев. В этом преобразовании, были использованы параметры m_1 и m_2. Они будут нужны для избежания конфликтов, когда расстояниями между автоматами будет равно двум, а не трем. Ниже будет пояснение.

Второе преобразование, обозначенное стрелкой A_2, соответствует третьему шагу автомата A_2. Он соединяет ребра $\mathbf{r}_1$ и $\mathbf{r}_2$, делая из них $\mathbf{r}_1^2$. Важно, что он помечает ребра $\mathbf{e}_1$ и $\mathbf{e}_3$, как вспомогательные, чтобы следующий автомат их мог удалить. Тем самым, показано, что конфликтов не возникает и в специальных случаях.

Осталось рассмотреть случай неконфликтности двух преобразований на удаление вблизи записей. Этот случай является частным случаем уже разобранных выше, так как минимальное расстояние до записей равно 2. Это было доказано выше, когда разбирались алгоритм удаления вблизи записей. Пример, иллюстрирующий этот случай, изображен на рисунке 2.51.

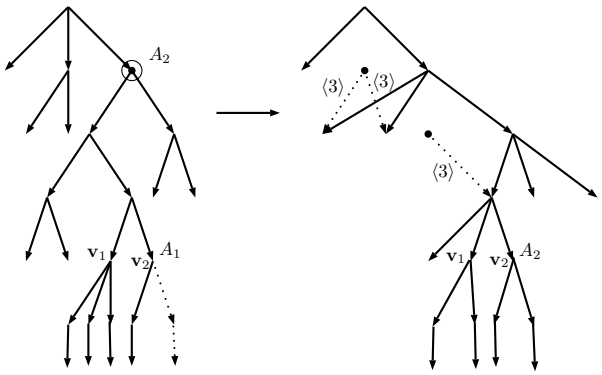


Рис. 2.51: Пример неконфликтности двух автоматов на удаление. Случай вблизи листьев. Автомат A_1 удалил запись и завершил функционирование.

Рассмотрим теперь случай, когда расстояние между двумя запросами на удаление равно двум. В такой конфигурации автомат A_2 после одного такта окажется на один уровень ниже, чем тот уровень, на котором находиться автомат A_1. Для избежания конфликтов в этих случаях, алгоритм перестройки автоматов будет немного отличаться от уже введенных преобразований. При этом данные алгоритмы будут применяться исключительно в этих случаях, так как иначе перестроенный граф не будет 2-3-4 деревом. Пусть автомат A_1 находиться в вершине v_1, а автомат A_2 находиться в вершине v_2. Дополнительные преобразования будут влиять только на уровни между вершиной v_2 и вершиной v_1, на других уровнях автоматы будут применять уже введенные ранее преобразования. Как уже было доказано выше, двух подряд удалений корня не может быть. Рассмотрим последовательность автоматов A_0, A_1, A_2, A_3. Если расстояние между A_1 и A_2 равно двум, то расстояние между A_0 и A_1, A_2 и A_3 не меньше трех. Поэтому, если мы покажем, что между автоматами A_1 и A_2 конфликтов не возникает, то конфликтов не возникнет для любого количества подряд идущих автоматов на удаление. Докажем это.

Опишем отдельно преобразования для автомата A_1 и A_2. После этого покажем, что конфликтов не возникнет, а перестроенный граф останется 2-3-4 деревом. Сначала опишем преобразования автомата A_1.

Пусть автомат A_1 находиться в вершине v_1. Рассмотрим первый случай, когда у v_1 только один брат, или два брата, но либо вершина v_1 имеет трое потомков, либо среди братьев v_1 есть вершина, у которой трое детей.

Преобразование на первом шаге автомата A_1 для этого подслучая изображено на рисунке 2.52. Оно отличается от стандартного тем, что автомат делает суммарное количество внуков не меньше 5, и суммарное количество внуков брата не меньше 5. При этом автомат A_2 не будет делать преобразований, если он идет в вершину, которая является сыном или племянником вершины v_1. При этом автомат A_2 окажется в хорошей для себя конфигурации. Под хорошей конфигурацией понимается, что если автомат A_2 оказывается в вершине, у которой только 1 брат, то количество детей и племянников этой вершины должно быть не меньше 5. Это возможно из-за того, что $m_1 + m_2 \geq 5$ и $m_3 + m_4 \geq 5$. Если же автомат A_2 идет в другие вершины, то конфликтов опять не возникает, так как автомат A_1 не преобразует родителя вершины v_1.

Нужно показать, что обязательно найдутся такие m_1, m_2, m_3, m_4, что $m_1+m_2 \geq 5$, $m_3 + m_4 \geq 5$, $m_1 + \cdots + m_4 = n_1 + \cdots + n_5$ или $m_1 + \cdots + m_4 = n_1 + \cdots + n_5 - 1$. Поясним, откуда может взяться минус один.

Алгоритм автомата на удаление устроен таким образом, что на одном шаге он может совместить только одну пару ребер в одно ребро. Следовательно, автомат A_1 может уменьшить сумму на один. Таким образом, если $n_1+\cdots+n_5 \geq 11$, то для своих целей автомат A_1 может уменьшить это количество только на 1. Поэтому после того как автомат A_1 перестроит граф для себя, он сможет перестроить граф и для A_2, так как $m_1 + \cdots + m_4 \geq 10$. Отметим, что для этого преобразования необходимо, что радиус видимости 4. Иначе автомат A_1 не смог бы увидеть внуков брата.

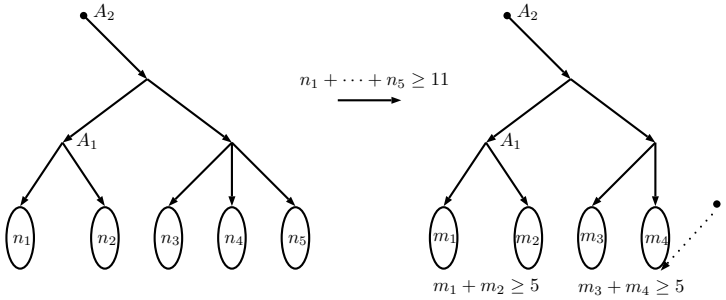


Рис. 2.52: Дополнительная перестройка автоматом A_1, в случае расстояния 2 до другого автомата на удаление.

Докажем, что $n_1 + \cdots + n_5 \geq 11$. Для доказательства этого факта нужно посмотреть на конфигурацию за 1 такт до этого. Автомат A_1 оказался в вершине, у которой всего 1 брат, потому что за такт до этого в этой вершине был другой автомат A_0, и он применил перестройку в этой вершине. Для более понятного объяснения представлено вспомогательное изображение на рисунке 2.53.

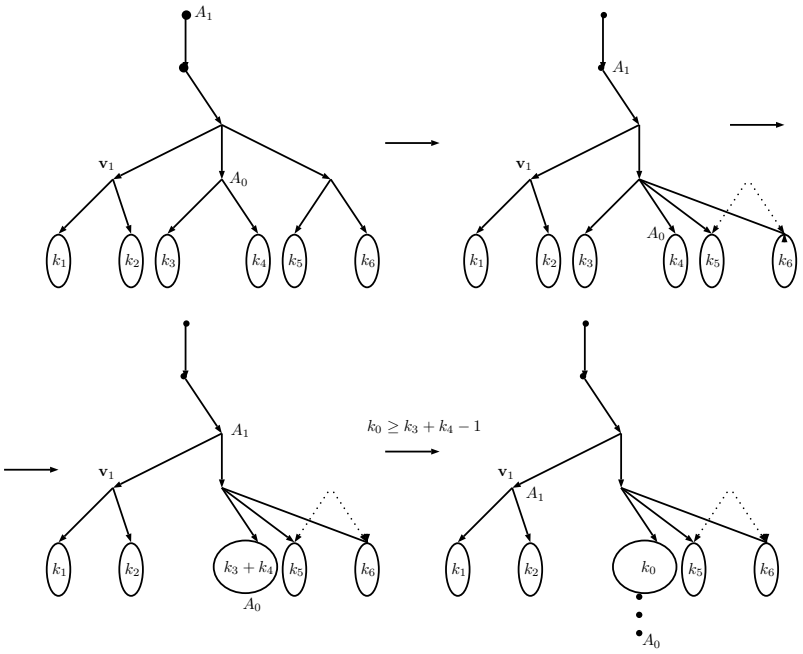


Рис. 2.53: Вспомогательное изображение. За такт до того, как автомат A_1 оказался в вершине v_1.

На этом рисунке изображена одна из возможных конфигураций. В частности, видно, что автомат A_0 объединил вершины k_3 и k_4, после чего мог сократить их количество на 1. Сократить количество детей у вершины более чем на 1 автомат на удаление не может, так как использует определенные преобразования, описанные выше, которые этого не делают. Из примера видно, что в момент, когда автомат A_1 попадет в вершину v_1, суммарного количество внуков и детей племянников было $k_1 + k_2 + k_0 + k_5 + k_6 \geq k_1 + \cdots + k_6 - 1 = n_1 + \cdots + n_5 \geq 12 - 1 = 11$. Утверждение доказано.

Итак, мы разобрали преобразования автоматов A_1 и A_2 для случая, когда у вершины v_1 только 1 брат, или есть сын или племянник, у которого трое детей. Остается разобрать случай, когда у вершины v_1 более одного брата, но у всех детей и племянником по двое сыновей.

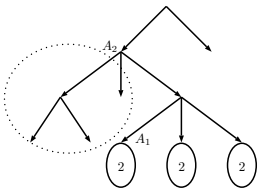


Рис. 2.54

Если входная окрестность автомата A_1 такая же, как и на рисунке 2.54, то автомат A_1 делает стандартные преобразования и в конце, при необходимости, делает дополнительные преобразования, как для случая, когда у вершины v_1 был один брат. Алгоритм A_2 остается таким же, как был описан выше. А именно, если он переходит в вершину, которая является сыном или племянником вершины v_1, то автомат A_2 не делает преобразований. Иначе делает стандартные преобразования, так как конфликтов при этом не возникнет. На рисунке 2.54, пунктирным овалом выделены вершины, которые при преобразованиях будет задействовать A_2. Видно, что при этом конфликтов с A_1 не возникнет.

Итак, осталось рассмотреть последний вариант, когда отец вершины v_1 является средним братом или правым, но когда количество детей у вершины v_2 равно двум. Идея избежания конфликтов совпадает с алгоритмом, изображенным на рисунке 2.50. Как упоминалось ранее, этот алгоритм содержит переменные m_1 и m_2, $m_1 + m_2 \geq 5$. Автомат A_1 на первом шаге делает $m_1 = m_2 = 3$, но на втором шаге может склеить пару ребер, и сделать например $m_1 = 2$. Автомат A_2 может не делать преобразования, если идет в вершину, которая является сыном или племянником v_1. Сделаем пояснение. Пусть автомат A_2 оказывается в овале m_1 и $m_1 \geq 3$, тогда эта конфигурация хорошая для A_2, так как количество братьев не меньше двух. Если же m_1 равно двум, то это означает, что автомат A_1 склеил два ребра на втором шаге своего функционирования. Следовательно, количество детей у вершины, в которой будет находиться A_2, не меньше 5, поэтому конфигурация и в этом случае хорошая для A_2.

Осталось рассмотреть граничные случаи. Единственный случай, который еще не разобран, когда автомат A_1 находиться на расстоянии 2 от записей, при этом автомат A_2 находиться на расстоянии 4 от записей. Преобразования на удаление устроено так, что автомат A_2 должен применить свое преобразование и завершить функционирование. Поэтому автомат A_1, чтобы не вызвать конфликт, помечает свою запись как удаленную, меняя предикат на ней на вспомогательный (f^h). При этом он не завершает функционировании. На следующий такт автомат A_1 смотрит, есть ли на расстоянии 4 от записей новый автомат. Если другой автомат есть, то автомат A_1 завершает функционировании, в силу того, что другой автомат, которого он видит, сотрет вспомогательные ребра. Если другого автомата нет, то A_1 удаляет запись. Заметим, что автоматы A_1 и A_2 обслуживают разные запросы. Это было разобрано выше. Поэтому они не смогут одновременно поменять информацию на одном и том же ребре.

Пример, иллюстрирующий этот случай, изображен на рисунке 2.55. Заметим, что автомат на удаление 13 (A_2) делает перестройку, так как видит, что автомат на удаление 12 (A_1) или какой-то другой автомат должен будет удалить 12 на следующий такт. Таким образом, конфликтов не возникает. Также обратим внимание, что после удаления 13 выделенные предикаты на ребрах f_{13} не заменяются на f_{12}. Заметим, что при этом корректность работы не нарушится. Благодаря этому наблюдению ав-

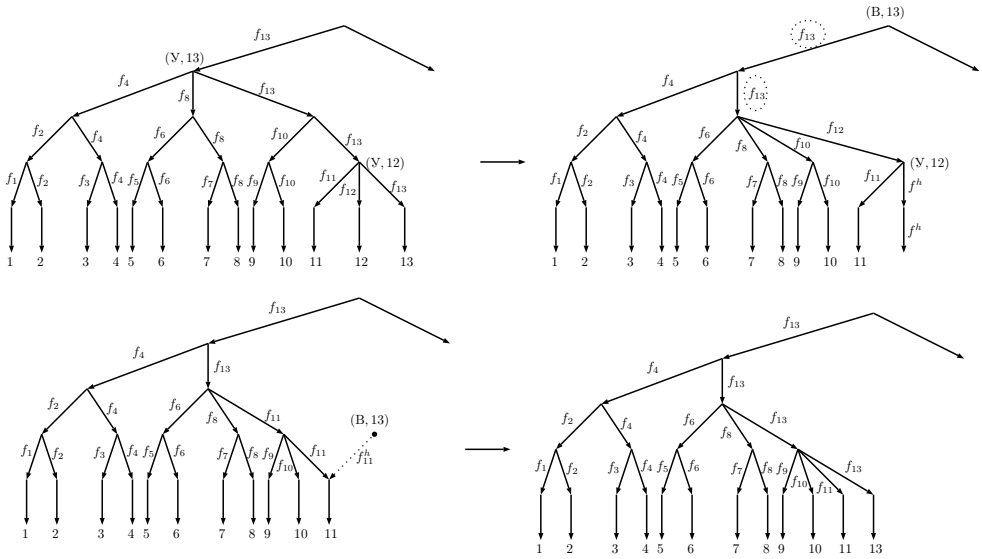


Рис. 2.55: Пример граничного случая.

томат на удаление может удалить запись в один проход. Для наглядности в пример добавлен автомат на вставку 13. Расстояние до записей у него 5, поэтому автомат на удаление 12 сам удалит вспомогательные ребра. Заметим, что автомат на вставку не применяет перестройку графа на третьем шаге. Это будет разобрано ниже, когда будут исключаться конфликты между автоматами на вставку и автоматами на удаление. Если бы расстояние до записей у автомата на вставку 13 было бы 4, то автомат на удаление 12 завершил бы функционирование, а автомат на вставку 13 удалил бы сам вспомогательные ребра.

Итак, доказано, что автоматы на удаление не вызывают конфликтов. Осталось доказать, что автоматы на вставку и автоматы на удаление не конфликтуют.

Рассмотрим автомат $A_{\text{В}}$ на вставку, а автомат $A_{\text{У}}$ на удаление. Причем автомат $A_{\text{В}}$ начал функционирование раньше. Расстояние между ними не меньше трех ребер. Пусть автомат $A_{\text{В}}$ стоит в вершине v_1. Если у этой вершины братьев не меньше 2, то автомат $A_{\text{В}}$ не уменьшит это количество, поэтому автомату $A_{\text{У}}$ не нужно перестраивать граф, если он идет в вершину v_1 или к ее братьям. Если же $A_{\text{У}}$ идет в другие вершины, то он может делать необходимые перестроения, и это не вызовет конфликта. Если же у v_1 только один брат, то автомат $A_{\text{У}}$ должен посмотреть на количество детей у вершины v_1. Если их количество не превосходит 3, то он может перестраивать граф. Если же их количество равно 4, то $A_{\text{У}}$ не нужно перестраивать граф, если он идет в вершину v_1 или к ее брату, так как автомат на вставку применит перестройку, и у вершину v_1 станет 2 брата. При этом автомат на вставку видя, что за ним идет автомат на удаление должен применить перестройку, даже в том случае, если ему это было не нужно. На рисунке 2.56(а) изображен пример конфигурации, когда действие алгоритма на удаление отличается от стандартного из-за автомата на

вставку.

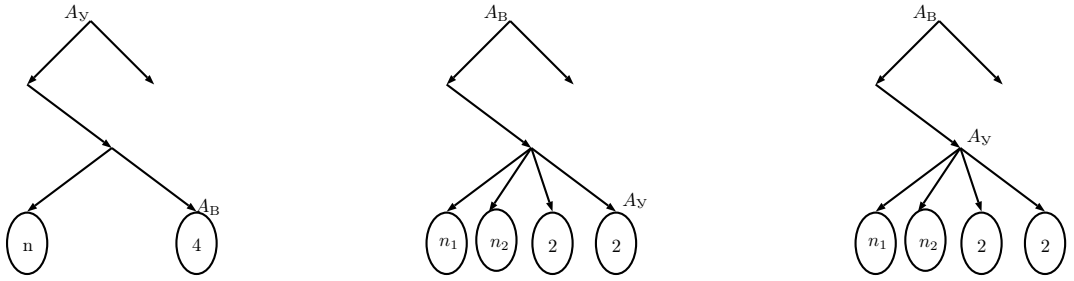


Рис. 2.56: Пример, когда действие алгоритма на удаление и вставку отличается от стандартных из-за их близости.

Рассмотрим теперь случай, когда автомат $A_У$ начал функционирование раньше, причем расстояние между ним и автоматом $A_В$ не меньше трех ребер. Доказывается симметрично предыдущему случаю. Единственное исключение изображено на рисунке 2.56(b). Автомат на вставку видит, что автомат на удаление применит перестроение, и не делает перестроение, которое мог бы сделать.

Рассмотрим оставшийся случай, когда расстояние между автоматом на удаление $A_У$ и автоматом на вставку $A_В$ равно двум. Это может произойти только тогда, когда автомат на удаление $A_У$ начал функционирование раньше. Этот случай похож на предыдущий. Так же конфликтов можно избежать. Пример исключения изображен на рисунке 2.56(c).

Единственный случай, который еще не разобран, когда автомат $A_У$ находиться на расстоянии 2 от записей, а автомат на вставку находиться на расстоянии 4 от записей. В этом случае, как и в разобранном ранее с двумя автоматами на удаление, автомат $A_У$ не удаляет свою запись, а меняет предикаты на вспомогательные. После этого он на следующий такт смотрит, есть ли в области видимости другой автомат. Если нет, то сам удаляет запись, иначе завершает функционирование. Пример этого случая изображен на рисунке 2.57. Заметим, что вставка делает перестройку, и это не вызывает конфликт с удалением, так как удаление меняет нагрузки ребер, которые вставка не изменяет. Так же автомат на удаление оказывается в вершине, в поддереве которого нет предикатов f^h. Но он все равно может удалить эти вспомогательные ребра, так как радиус видимости у него 4 ребра.

В качестве следствия вышеперечисленного справедливо следующее утверждение.

Лемма 3. *Для любого потока запросов H из $\mathcal{H}$, ПДИГ $\mathcal{D}$ функционирует без конфликтов.*

Докажем еще одно утверждение о корректности работы ПДИГ и о его сложности.

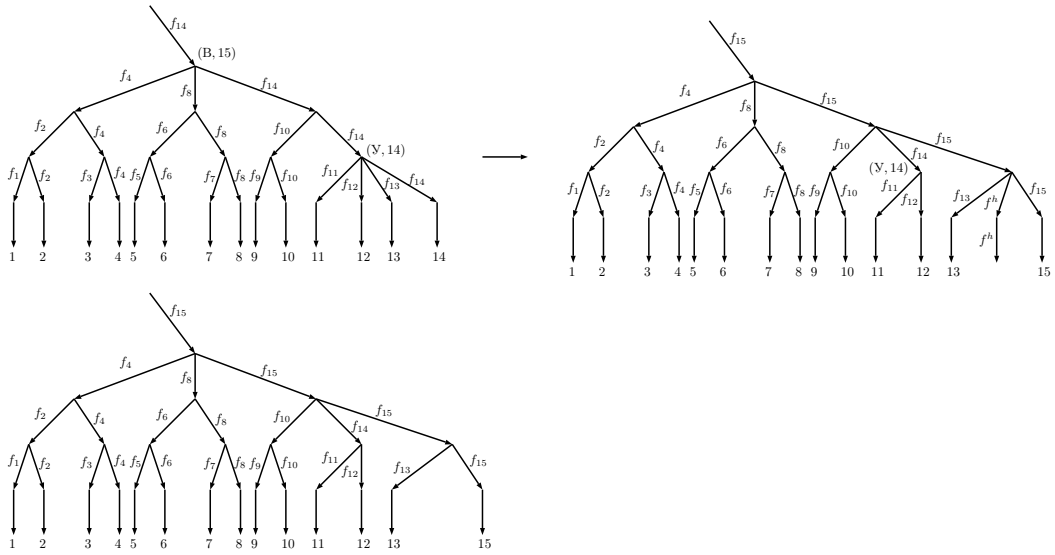


Рис. 2.57: Граничный случай, удаление и вставка.

Лемма 4. *Для любого потока запросов $H, H \in \mathcal{H}$ ПДИГ $\mathcal{D}$ функционирует корректно, кроме того, для любого натурального i количество тактов, необходимого на выдачу ответа на поисковый запрос $H(i)$, не превосходит*

$$\left]\frac{\log_2(\max\{|V(i,H)|,2\})}{3}\right[.$$

Доказательство. Нужно показать, что для любого запроса на поиск $H(i), H(i) = (\Pi, x)$, ПДИГ выдает в ответ $\{x\}$, если $x \in V(i,H)$ и пустое множество в противном случае.

Из леммы 3, в частности, следует, что любые преобразования сохраняют свойство удаленности для любого потока запросов H из $\mathcal{H}$. Другими словами, автоматы идут друг за другом в порядке появления запроса. Рассмотрим произвольный запрос на поиск, поступивший в i такт.

Рассмотрим преобразование на вставку (удаление), поступившее в такт $j, 1 \leq j \leq i-1$ и еще не завершившее свое функционирование. Так как ПДИГ сохраняет свойство удаленности, то в стадии поиска между ними будет расстояние $i - j \geq 2$ ребра, следовательно, автомат на вставку (удаление) успеет вставить (удалить) запись прежде, чем автомат на поиск сможет сократить расстояние.

Поэтому ПДИГ работает корректно, так как любое преобразование завершиться не меньше, чем за такт до того, как запрос на поиск их достигнет. Преобразования на изменение ИГ, устроены так, что после каждого такта полученный ИГ является 2-3-4 деревом, не учитывая вспомогательные ребра. Причем, если нагрузка ребра есть предикат f_a, то это означает, что в этом поддереве находятся записи, которые

не превосходят a. Кроме того, к каждой записи y ведет ребро с нагрузкой $f_{=,y}$. Следовательно, поиск найдет искомую запись, если она есть в базе данных, и не найдет, если ее нет. Корректность доказана.

Оценим теперь сложность ПДИГ $\mathcal{D}$. Заметим, что автомат может изменить высоту ИГ только один раз, причем ровно на 1 и только в первый такт своего функционирования. Это следует из алгоритма автомата. Все преобразования не изменяют высоту ИГ, кроме преобразований в корне. Преобразование на вставку может увеличить на 1 высоту ИГ, а преобразование на удаление может уменьшить на 1 высоту ИГ.

Рассмотрим произвольный поток запросов H. Пусть запрос на поиск пришел в такт i. База данных в этот такт есть $V(i,H)$. Следовательно, высота ИГ не превосходит $]\log_2(\max\{|V(i,H)|,2\})[$. Автомат на поиск каждый такт проходит по 3 уровня вниз или завершает функционирование. Следовательно, он затратит на поиск не больше, чем $\left]\frac{\log_2(\max\{|V(i,H)|,2\})}{3}\right[$ тактов, так как подграфы ИГ не изменят свою высоту.

□

Утверждение теоремы является следствием лемм 1 и 2.

□

3 Бесконечный ПДИГ со степенью ветвления один, решающий ДЗПИО

Рассмотрим множество предикатов $F(T) = \{f^0, f^1, f_a^t(x) | t \in T, a \in \mathbb{N}\}$, где $|T| < \infty$ и $\forall t \in T, a \in \mathbb{N}$:

$$f_a^t(x) = \begin{cases} 1, & \text{если } x = a; \\ 0, & \text{иначе} \end{cases}, f^0(x) \equiv 0, f^1(x) \equiv 1.$$

Пусть $T_0 = \{\text{П}, \text{В}_1, \text{В}_2, \text{У}\}$, то есть $F(T_0) = \{f^0, f^1, f_a^{\text{П}}, f_a^{\text{В}_1}, f_a^{\text{В}_2}, f_a^{\text{У}} : a \in \mathbb{N}\}$. Введем базовое множества преобразований $\mathcal{R}$. Базовое множество $\mathcal{R}$ состоит из 13 преобразований, изображенных на рисунках 2.59 - 2.70. Преобразования будут описаны в доказательстве теоремы 4.

В качестве базового множества для ПДИГ будем рассматривать множество

$$\mathcal{F}_0 = \langle F(T_0), \emptyset, \mathcal{R}, \eta_{id} \rangle. \tag{2.3}$$

Справедлива следующая теорема.

Теорема 4. *Существует бесконечный, селекторный ПДИГ типа (1,1) над базовым множеством $\mathcal{F}_0$, определяемым соотношением (2.3), который решает ДЗПИО для любого потока запросов H из $\mathcal{H}$ со сложность $L, L \equiv 3$.*

Доказательство. Теорема 4 утверждает, что существует бесконечный ПДИГ типа (1,1), решающий ДЗПИО для любого потока запросов. Причем время поиска любого запроса всегда равно трем тактам.

Идея построения ПДИГ будет следующей. Типы предикатов П,В,У будут означать, что их создал автомат на поиск, вставку или удаление. Значение индексов 1 и 2 у вставки $В_1$ и $В_2$ объясним ниже.

ИГ будет состоять из одного ребра. Пример такого ИГ изображен на рисунке 2.58. В ИГ всего две вершины. Корень будем называть первой вершиной, соседнюю с корнем второй. Тип второй вершины будет "л" или "в".

В первый такт своего функционирования любой автомат меняет предикат, выходящий из корня, на новый, у которого тип совпадает с типом запроса, который он обслуживает, а аргумент равен самому запросу. Например, если поступил запрос на поиск 7, то на следующий такт из корня будет выходить ребро с предикатом $f_7^{\text{П}}$. Исключение составляет автомат на вставку. Если тип предиката, выходящего из корня был $В_1$, то после преобразования автомат поменяет тип на $В_2$, а в других случаях он поменяет тип на $В_1$. Смысл данного преобразования будет разъяснен ниже.

Опишем алгоритм работы автоматов на удаление и поиск. Автомат на удаление после первого такта завершает функционирование. Автомат на поиск переходит во вторую вершину. На второй такт автомат на поиск запоминает тип второй вершины (либо "л", либо "в"). В третий такт он смотрит поменялся ли тип второй вершины. Если тип поменялся, то в качестве ответа он выдает запись, приписанную второй вершине и завершает функционирование. Если тип вершины не поменялся, то он выдает ответ, что записи нет, после чего завершает функционирование. Видно, что автомату на поиск при таком алгоритме нужно три такта, чтобы выдать ответ.

Итак, выше описан алгоритм работы автоматов на удаление и на поиск, осталось описать алгоритм автомата на вставку. Автомат на вставку в отличие от автомата на поиск и удаление может функционировать бесконечно долго.

Основная идея состоит в том, что все автоматы на вставку, начиная со второго такта, находятся во второй вершине и смотрят на предикат, выходящий из корня. При этом двух автоматов на вставку одного и того же элемента не будет. Как это можно сделать будет описано ниже. Все автоматы на вставку смотрят значение предиката f_a^t первого ребра. Если для какого-то автомата это значение равно 1, то это значит, что этот автомат обслуживает запись a. Возможно пять вариантов: предикат первого ребра поменялся на $f_a^{\text{П}}$, $f_a^{\text{У}}$, $f_a^{\text{В}_1}$, $f_a^{\text{В}_2}$ или предикат не поменялся.

Рассмотрим первый вариант, когда предикат поменялся на $f_a^{\text{П}}$. Если среди автоматов на вставку, находящихся во второй вершине, есть автомат на вставку a, то этот автомат на следующий такт поменяет тип второй вершины с "в" на "л" или с "л" на "в", а так же присвоит ей запись a. Отметим еще раз, что автомат понимает, что его запись совпадает с запись автомата на поиск, если значение предиката на первом ребре будет равно одному.

Второй случай, когда предикат поменялся на $f_a^{\text{У}}$. Если среди автоматов на встав-

ку, находящихся во второй вершине, есть автомат на вставку a, то этот автомат на следующий такт завершает свое функционирование.

Третий и четвертый случаи одинаковые. В первый такт своего функционирования автомат держит внутренний параметр, который равен 0, а также запоминает тип предиката, который он создаст (В_1 или В_2). Начиная со второго такта, он смотрит на тип предиката, выходящего из корня ребра. Если он поменялся, то автомат меняет свой внутренний параметр на 1. Если кроме этого тип предиката поменялся на вставку с другим индексом, и записи совпали, то автомат завершает свое функционирование (его место займет другой автомат с той же записью). Аналогично, если внутренний параметр автомата на вставку уже был равен 1, и он увидел, что пришел новый автомат на вставку той же записи, то этот автомат так же завершает свое функционирование.

Итак, выше был описан алгоритм работы ПДИГ. Как видно из данного алгоритма, нагрузку первого ребра меняет только тот автомат, который находится в корне. Нагрузку второго ребра меняет только один автомат на вставку. Поэтому ПДИГ работает бесконфликтно.

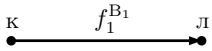


Рис. 2.58: Пример ИГ, участвующий в доказательстве теоремы 4.

Для формального доказательства осталось предъявить базовое множество преобразований $\mathcal{R}$,

$$\mathcal{R} = \{R_1, R_2, \ldots, R_{13}\},$$

которые будет применять автомат $\mathcal{A}$ в зависимости от входной окрестности, в соответствии с описанным выше алгоритмом. А так же доказать корректность работы ПДИГ.

Для простоты объяснения будем считать, что до поступления запросов ИГ имел вид, изображенный на рисунке 2.58.

Все преобразования можно распределить на три группы: поиск, вставка и удаление. Рассмотрим сначала преобразования, которые делают автоматы на вставку, обрабатывая свои запросы.

Код предикатного ребра ИГ это пара (тип предиката, значение предиката на запросе). Например, рассмотрим предикат $f_7^{\text{П}}$ и автомат, обслуживающий запрос на вставку 6. Тогда код, который он увидит на этом ребре будет $(\text{П}, 0)$. Если бы он обслуживал запрос 7, то он увидел бы код $(\text{П}, 1)$.

Итак, разберем преобразования на вставку. Преобразование R_1 – это перемещение автомата на вставку в соседнюю с корнем вершину и изменение предиката ребра. Это преобразование изображено на рисунке 2.59. Пунктиром для наглядности обозначено ребро, предикат которого изменяется в результате преобразования. Автомат

на вставку делает данное преобразование в случае, если код ребра был равен $(\mathrm{B}_1, 0)$ или $(\mathrm{B}_1, 1)$. В правой части преобразования над пунктирным ребром написано $\langle \mathrm{B}_2, * \rangle$. Это означает, что автомат говорит ИГ, что нагрузкой этого ребра нужно сделать предикат $f_*^{\mathrm{B}_2}$. Пример применения преобразования R_1 с точки зрения ИГ изображен на рисунке 2.60. Если код входной окрестности отличен от $(\mathrm{B}_1, 0)$ и $(\mathrm{B}_1, 1)$, то автомат на вставку применяет преобразование R_2. Оно изображено на рисунке 2.59. Пример применения преобразования R_2 с точки зрения ИГ изображен на рисунке 2.60. Как было описано в алгоритме, автомат на вставку имеет внутреннее состояние 0 в первый такт функционирования, кроме этого он запоминает тип предиката, который он создаст. На рисунке внутреннее состояние автомата обозначено через q, а тип который он запомнит через t.

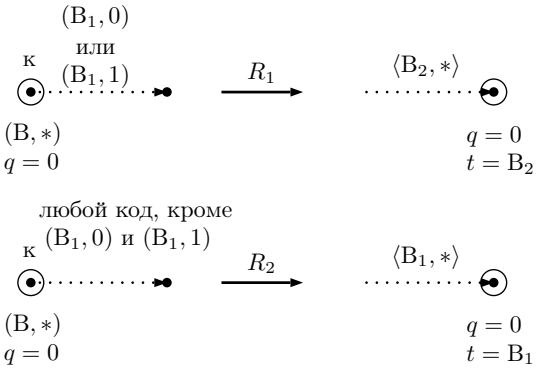


Рис. 2.59: Преобразования R_1 и R_2 на вставку с точки зрения автомата. q это внутренне состояние автомата, t это тип, который он запоминает.

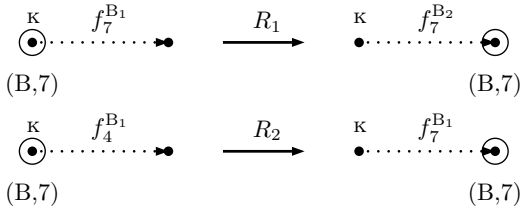


Рис. 2.60: Пример применения преобразований R_1 и R_2 с точки зрения ИГ.

Преобразование R_3 изображено на рисунке 2.61. Оно означает, что тип предиката, выходящего из корня не поменялся. Такое могло произойти, только в случае, если в потоке запросов был пустой запрос. Поэтому автомат не применяет никаких преобразований, а просто продолжает функционировать.

Преобразование R_4 изображено на рисунке 2.62. Оно применяется, когда тип предиката поменялся с предыдущего такта, но его значение равно нулю, либо внутреннее состояние автомата было равно единице, и значение предиката рано нулю. В этом

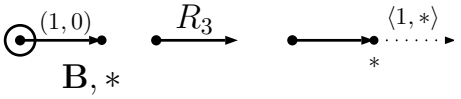


Рис. 2.61: Преобразование R_3.

случае автомат меняет свое состояние на единицу и забывает про тип. Если состояние автомата уже было рано одному, то он его оставляет.

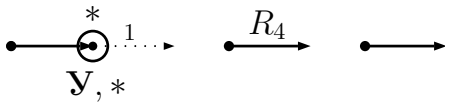


Рис. 2.62: Преобразование R_4. Случай когда $q = 0$ и $t^{'} = t$ это преобразование R_3.

Преобразование R_5 изображено на рисунке 2.63. Оно применяется, когда тип предиката поменялся с предыдущего такта и равен либо В_1 или В_2, а значение равно 1. В этом случае автомат завершает свое функционирование.

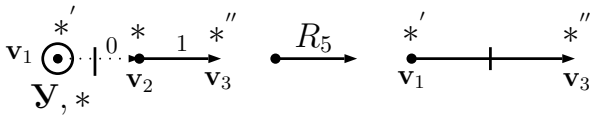


Рис. 2.63: Преобразование R_5.

Преобразование R_6 изображено на рисунке 2.64. Оно применяется, когда внутреннее состояние автомата равно одному, а тип предиката ребра, выходящего из корня, стал равен либо В_1, либо В_2, и значение равно 1. В этом случае автомат завершает свое функционирование.

В преобразованиях R_5 и R_6 автомат завершает свое функционирование, так как вместо него встанет новый автомат, обслуживающий тот же запрос.

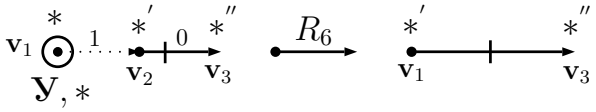


Рис. 2.64: Преобразование R_6.

Преобразование R_7 изображено на рисунке 2.65. Автомат на вставку завершает свое функционирование вне зависимости от внутреннего состояния q.

Осталось рассмотреть последние два преобразования для вставки. Преобразования R_8 и R_9 изображены на рисунке 2.66. Автомат на вставку видит, что пришел

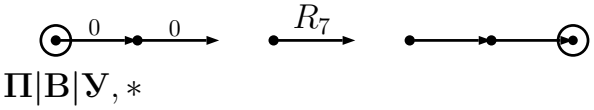


Рис. 2.65: Преобразование R_7.

запрос на поиск того же элемента, что он обслуживает. Поэтому он меняет тип вершины с "в"на "л"(R_8) или с "л"на "в"(R_9), а так же присваивает ей запись, которую он обслуживает.

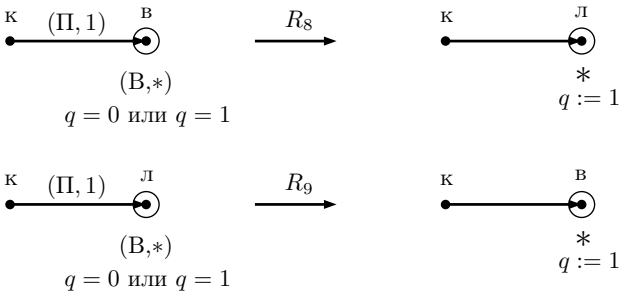


Рис. 2.66: Преобразования R_8 и R_9.

Рассмотрим преобразования, которые применяет автомат на удаление. Как видно из алгоритма, такое преобразование всего одно (R_9). Оно изображено на рисунке 2.67. Автомат на удаление меняет предикат и завершает функционирование.

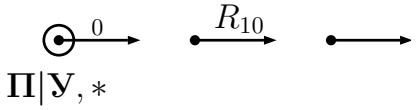


Рис. 2.67: Преобразование R_{10}.

Осталось рассмотреть преобразования, применяемые автоматом на поиск. Из алгоритма видно, что автомат на поиск применяет всего три преобразования. Первое преобразование R_{11} аналогично преобразованию R_{10} на удаление и изображено на рисунке 2.68. Единственное отличие состоит в том, что автомат на поиск переходит в соседнюю с корнем вершину и продолжает функционировать.

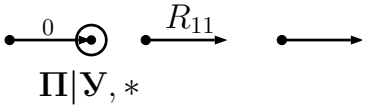


Рис. 2.68: Преобразование R_{11}.

Преобразование R_{12}, применяемое автоматом на поиск, изображено на рисунке 2.69. Автомат на поиск запоминает тип второй вершины. Для простоты объяснения, считаем, что состоянию p автомата на поиск присваивается тип вершины t_v.

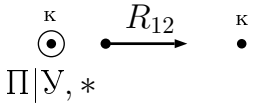


Рис. 2.69: Преобразование R_{12}.

Последнее преобразование R_{13} изображено на рисунке 2.70. Если тип второй вершине не поменялся ($p = t_v$), то искомой записи нет. Если тип второй вершины поменялся, то значит его поменял автомат на вставку искомой записи, поэтому автомат на поиск выдает в качестве ответа запись, приписанную второй вершине. В любом случае после этого автомат на поиск завершает функционирование.

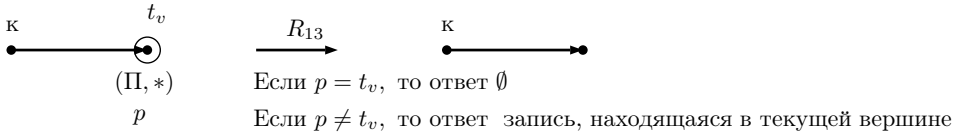


Рис. 2.70: Преобразование R_{13}.

Итак, описаны все преобразования, которые будут использовать автоматы. Покажем теперь, что построенный ПДИГ работает бесконфликтно, корректно для любого потока запросов H, и количество тактов необходимое на поисковый запрос равно трем.

Последнее утверждение очевидно, так как автомат на поиск функционирует ровно три такта, после чего выдает ответ. Осталось доказать, что ПДИГ работает бесконфликтно и корректно.

Докажем, что ПДИГ работает бесконфликтно. Это означает, что не бывает такой ситуации, при которой два автомата изменяют нагрузку одно и того же ребра или вершины.

Как мы уже отмечали, автоматы не могут вызвать конфликт, изменяя первое ребро. Действительно, первое ребро изменяет только автомат, находящийся в корне. В корне находится не более одного автомата, следовательно конфликтов возникнуть не может. Нагрузка корня не изменяется, поэтому конфликтов так же не возникает.

Осталось доказать, что конфликтов не возникает при изменении нагрузки второй вершины.

Как видно из алгоритма вставки, автомат после получения запроса на вставку оказывается во второй вершине. При этом, если в этой вершине уже был автомат на вставку той же записи, то он завершает функционирование. Следовательно, во второй вершине могут находиться только два автомата на вставку одной и той же

записи, причем ровно один такт и на следующий такт один из автоматов обязательно завершит функционирование. Пример описанной ситуации изображен на рисунке 2.71 (3, 4 и 5 такты). Второе ребро и третье вершину может изменить только автомат на вставку, запись которого совпала с записью поиска. Как было доказано выше, такой автомат может быть только один, следовательно, конфликтов не возникнет.

Итак, доказано, что ПДИГ работает бесконфликтно. Осталось доказать, что он работает корректно. Это означает, что когда поступает запрос на поиск записи, в ответ он должен дать эту запись, если она принадлежит базе данных или сказать, что такой записи нет.

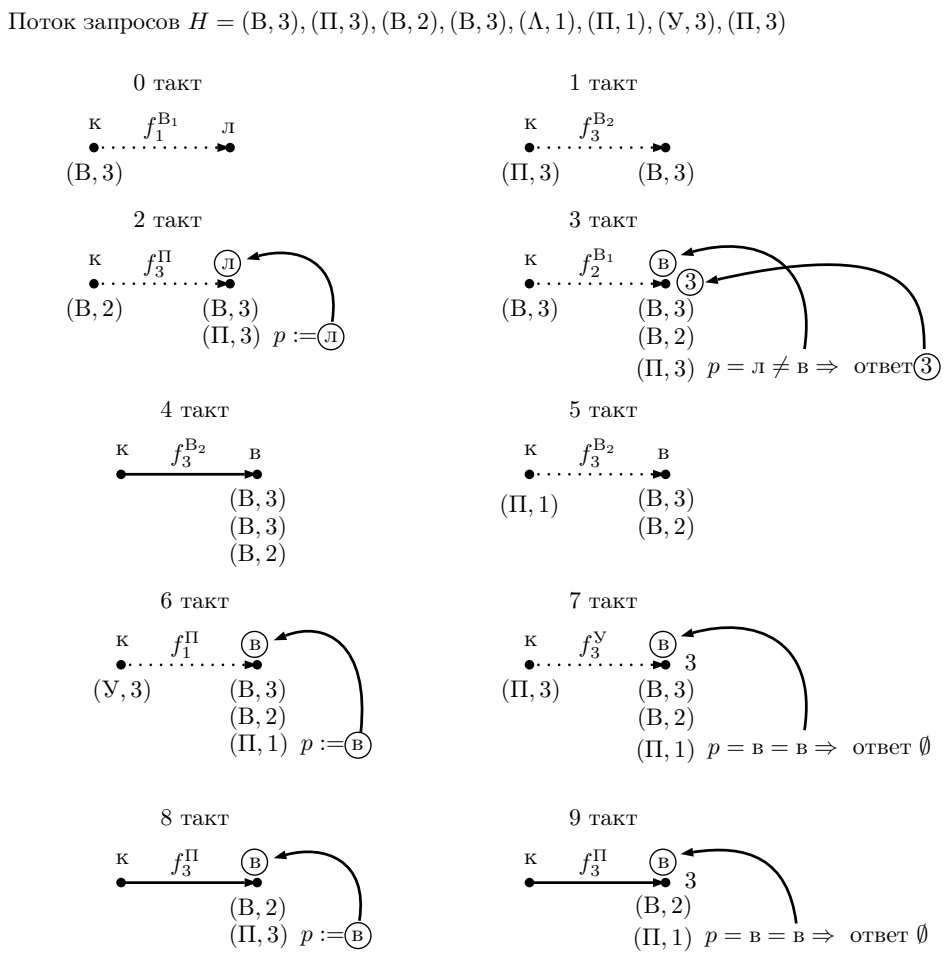


Рис. 2.71: Пример функционирования ПДИГ, удовлетворяющий теореме 4. Пунктиром изображены ребра, нагрузка которых изменится на следующий такт.

Если к ПДИГ поступал запрос на вставку, то обслуживающий его автомат постоянно функционирует во второй вершине. При этом он может завершить свое функционирование, только если поступил новый запрос на вставку той же записи, или

поступил запрос на ее удаление. В других случаях он продолжает функционировать. Поэтому, если поступит запрос на поиск этой записи после того, как был запрос на ее вставку, то во второй вершине обязательно будет функционировать автомат, который ее обслуживает. Следовательно, ПДИГ будет работать корректно. Если был послан запрос на удаление, а после него шел запрос на поиск той же записи, то автомат на вставку завершит свое функционирование. В этом случае опять ПДИГ будет работать корректно, так как поиск выдаст ответ $\emptyset$ в этом случае. Для лучшего понимания на рисунке 2.71 изображен пример функционирования ПДИГ на небольшом количестве запросов.

Теорема доказана.

□

4 Конечный не селекторный ПДИГ со степенью ветвления один, решающий логическую ДЗПИО

Пусть

$$T_1 = \{\text{да}, \text{нет}\},$$

то есть $F(T_1) = \{f^0, f^1, f_a^{\text{да}}, f_a^{\text{нет}} : a \in \mathbb{N}\}$, где $f^0 \equiv f_a^{\text{да}} \equiv f_a^{\text{нет}} \equiv 0, f^1 \equiv 1$. Введем базовое множества преобразований $\mathcal{R}_1$. Базовое множество $\mathcal{R}_1$ состоит из 4 преобразований, изображенных на рисунке 2.73. Преобразования будут описаны в доказательстве теоремы 5.

В качестве базового множества для ПДИГ будем рассматривать множество

$$\mathcal{F}_1 = \langle F(T_1), \mathcal{R}_1 \rangle. \tag{2.4}$$

Теорема 5. *Существует конечный, не селекторный ПДИГ типа (1,1) над базовым множеством $\mathcal{F}_1$, определяемым соотношением (2.4), решающий логическую ДЗПИО со сложностью $L, L \equiv 2$.*

Доказательство. Искомый ПДИГ будет достаточно простым, в силу того, что разрешается в преобразованиях использовать не селекторные функции.

Рассмотрим функцию $P : \mathbb{N} \to \mathbb{N}$, которая на входе получает натуральное число n, а на выходе выдает n-е простое число. То есть $P(1) = 2, P(2) = 3, P(3) = 5, P(4) = 7, \ldots$.

Будем использовать следующие функции в преобразованиях.

$$\phi_{\text{В}}(a, x) = \begin{cases} f_a^{\text{нет}}, & \text{если } 0 = a \pmod{P(x)}; \\ f_{aP(x)}^{\text{нет}}, & \text{иначе}; \end{cases};$$

$$\phi_{\text{У}}(a, x) = \begin{cases} f_{\frac{a}{P(x)}}^{\text{нет}}, & \text{если } 0 = a \pmod{P(x)}; \\ f_a^{\text{нет}}, & \text{иначе}; \end{cases};$$

$$\phi_{\text{П}}(a,x) = \begin{cases} f_a^{\text{да}}, & \text{если } 0 = a \pmod{P(x)}; \\ f_a^{\text{нет}}, & \text{иначе;} \end{cases};$$

ИГ будет состоять из одного ребра. Изначально считаем, что ИГ состоит из одного ребра с предикатом $f_1^{\text{нет}}$. Пример начального ИГ изображен на рисунке 2.72.

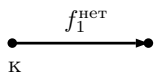


Рис. 2.72: Изначальный ИГ.

Основная идея состоит в следующем. Аргумент единственного предиката в ИГ каждый такт времени равен произведению простых чисел, соответствующих элементам базы данных. Например, если в базе данных находятся числа 1, 4, 7, то аргумент предиката будет равен $P(1)P(4)P(5) = 2 * 7 * 11 = 154$. Имея такой аргумент, поиск осуществляется проверкой деления его на простое число соответствующее поисковому запросу. Например если поступает запрос на поиск 3, то проверяется делится число $154 = P(1)P(4)P(5)$ на $P(3) = 5$. Оно не делится, значит 3 в базе данных нет. Не сложно заметить, что ответ будет положительный только при поиске 1,4 или 5. Тем самым ПДИГ будет работать корректно. Чтобы добиться такого аргумента, будут использованы преобразования на основе не селекторных функций $\phi_{\text{В}}(a,x)$ и $\phi_{\text{У}}(a,x)$.

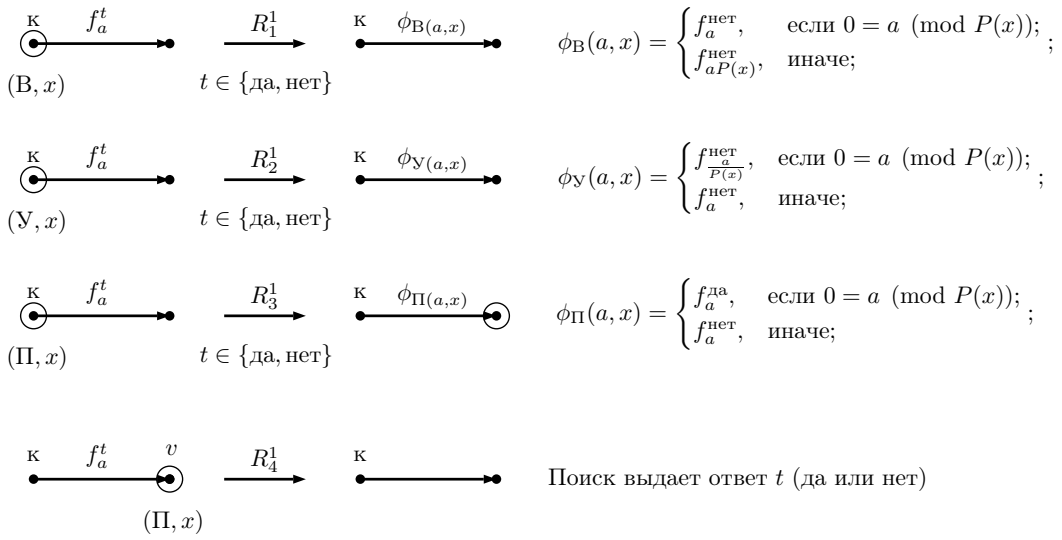


Рис. 2.73: Преобразования R_1^1, R_2^1, R_3^1 и R_4^1.

Итак, разберем преобразования, применяемые ПДИГ. Все преобразования изображены на рисунке 2.73. Преобразование R_1^1 это преобразование на вставку. Автомат

на вставку x функционирует ровно один такт. Он применяет преобразование, изменяющее аргумент предиката на новый. Если аргумент предиката делился на $P(x)$, значит в базе данных уже есть x и его не нужно вставлять. Поэтому аргумент предиката остается прежним. В противном случае аргумент предиката домножается на $P(x)$.

Преобразование R_2^1 на удаление аналогично преобразованию на вставку. Аргумент предиката делится на $P(x)$, если тот делился на $P(x)$ и остается прежним в противном случае.

Преобразование R_3^1 на поиск x проверяет, делится ли аргумент предиката на $P(x)$. Если делится, то в следующий такт автомат на поиск увидит тип предиката да, иначе увидит нет. Поэтому преобразование R_4^1 – это выдача ответа. Автомат на поиск возвращает ответ да, если тип предиката равен да, и иначе ответ нет.

Приведенный алгоритм работает корректно. Пусть база данных V состоит из элементов $V = \{y_1, \dots y_k\}$. Не трудно заметить, что аргумент предиката будет равен $P(y_1) \dots P(y_k)$. Очевидно, что если поступит запрос на поиск $x, x \in V$, то $P(y_1) \times \dots P(y_k)$ разделится на $P(x)$. Если же $x \notin V$, то $P(y_1) \dots P(y_k)$ не разделится на $P(x)$.

Конфликтов также не возникает, так как только один автомат меняет нагрузку предиката в один такт. В конце приведем наглядный пример, изображенный на рисунке 2.74, работы ПДИГ.

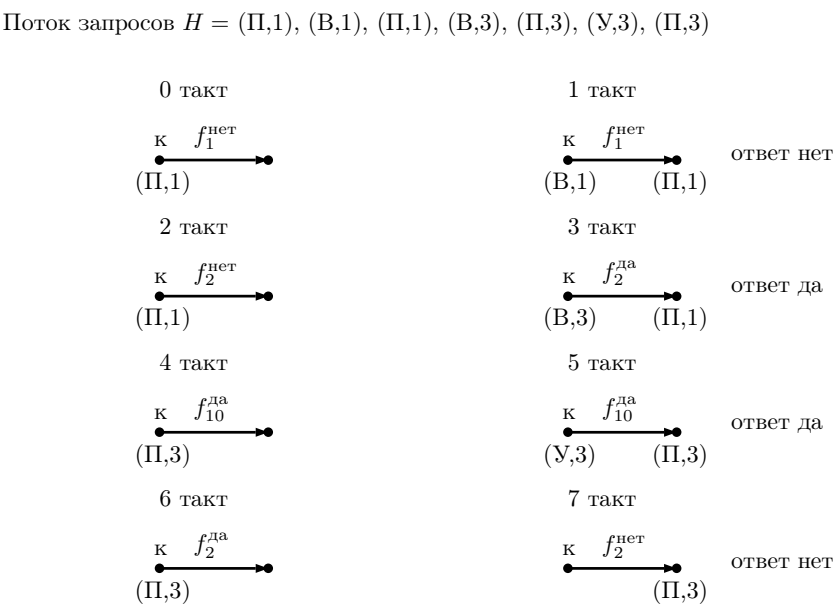


Рис. 2.74: Пример функционирования ПДИГ.

Теорема 5 доказана. □

5 Минимально возможный по степени ветвления ПДИГ с радиусом видимости один, решающий ДЗПИО

В этой главе докажем, что существует конечный ПДИГ типа (2,1), который решает ДЗПИО для любого потока запросов.

В ИГ каждой вершине присвоен один из трех типов: "к"(корень), "в"(внутренняя) и "л"(листовая). Для доказательства теоремы 6 разрешим использовать другие типы вершин в ИГ. А именно, введем три новых типа вершин ИГ: "у", "к$_{\leftarrow}$" и "к$_{\rightarrow}$". Эти типы будут нужны для доказательства теоремы 6. Сразу сделаем замечание, что можно было бы и не вводить новые типы вершин. Для этого вместо типа "у" нужно подставить тип "к", вместо типа "к$_{\leftarrow}$" тип "в", а вместо типа "к$_{\rightarrow}$" тип "л". В этом случае доказательство теоремы 6 сильно бы усложнилось дополнительными объяснениями. Поэтому будем считать, что у нас есть три новых типа вершин, которые могут встречаться в ИГ.

Пусть

$$T = \{\text{а}, \text{н}\} \times \{-2, -1, 0, 1, 2\} \times \{\text{П}_{\leftarrow}, \text{П}_{\rightarrow}, \text{В}_{\leftarrow}, \text{В}_{\rightarrow}, \text{У}_{\leftarrow}, \text{У}_{\rightarrow}, \text{н}\},$$

$$F_2 = \{f^t_{l,r,a}(x) : t \in T, l, r, a, x \in \mathbb{N}\}, \text{ где}$$

$f^t_{l,r,a}(x) = 1$, если $x = a$ и 0 иначе.

Теорема 6. *Существует множество преобразований $\mathcal{R}$ и существует конечный ПДИГ типа (2,1) над базовым множеством $\langle F_2, \emptyset, \mathcal{R}, \eta_{id}\rangle$, решающий ДЗПИО со сложностью $L, L(n) =]n/2[+5$.*

Доказательство. Множество преобразований $\mathcal{R}$ будет представлено явно в процессе доказательства. В данной теореме не будем представлять иллюстрацию к каждому преобразования, так как здесь более понятно и удобнее описать алгоритм автомата. К некоторым преобразованиям приведем иллюстрации, а к остальным по аналогии можно так же их предъявить.

Идея доказательства будет следующей. ПДИГ типа (2,1), поэтому база данных будет представлять из себя список, причем корень ИГ будет находиться посередине. Следовательно, запросу на поиск потребуется в худшем случае $]n/2[$ тактов на выдачу ответа, где n — это мощность базы данных. В теореме сложность оценивается как $]n/2[+5$. Ниже объясним, откуда могут взяться еще 5 тактов.

Чтобы поддерживать корень посередине базы данных, будет использоваться идея из теоремы 5 с некоторыми дополнениями. А именно, в первый такт своего функционирования автомат меняет предикат или предикаты, выходящие из корня, спрашивая есть ли искомая запись в базе данных или нет. Во второй такт автомат смотрит значение t_3, полученного предиката. Например, если это автомат на поиск, то на месте

t_3 он может увидеть "П$_{\leftarrow}$", "П$_{\rightarrow}$" или "н". "П$_{\leftarrow}$" ("П$_{\rightarrow}$") означает, что искомая запись существует и она находится слева(справа) от корня. Если автомат видит в качестве t_3 символ "н", это означает, что искомой записи нет.

Основная идея алгоритма

Итак, опишем основную идею алгоритма. Автомат в первый такт своего функционирования выясняет, есть ли обслуживаемый им запрос в базе данных или нет. Кроме этого, он смотрит нарушался ли баланс. Под балансом понимаем соотношение количества записей слева и справа от корня. Баланс будет находиться в параметре t_2 типа предиката.

Алгоритм различается у автоматов на поиск, вставку и удаление, но их объединяет общая идея, что в первый такт своего функционирования, автомат должен узнать есть искомая запись в базе данных или нет. Если есть, то он должен знать слева или справа от корня она находится. Чтобы узнать есть ли искомая запись, автомат меняет ребра или ребро, выходящее из корня, и на следующий такт считывает информацию с этих ребер или ребра. Автомат не всегда может поменять оба ребра, выходящих из корня, так как одно из них может менять автомат, который начал функционирование раньше. Ниже будет доказано, что ситуации, когда автомат не сможет поменять ни одного ребра не возникнет.

Все ребрам будут соответствовать только предикаты. Поэтому, будем говорить тип ребра, подразумевая тип предиката ему соответствующего.

Для того, чтобы корректно определить есть ли обслуживаемая запись в базе данных или нет, введем понятие *актуальноого* ребра. *Актуальным* будем называть ребро, которое выходит из корня и параметр t_1 его типа равен "а". Ниже будет показано, что может возникнуть ситуация, когда такого ребра нет, при этом тип корня будет обязательно равен "к$_{\leftarrow}$" ("к$_{\rightarrow}$"). В этом случае актуальным будем называть левое (правое) ребро.

Сделаем небольшое замечание, что случай, когда оба ребра имеют t_1 равным "н", возникает из-за пустых запросов в совокупности с запросами на вставки. Если пустых запросов нет, то такой ситуации вообще не возникнет.

Автоматы на вставку могут поддерживать баланс, вставляя новую запись в нужную сторону. Автоматы на поиск, как и пустые запросы, не изменяют баланс. Единственное, что может нарушить баланс, это автомат на удаление. Поэтому если автомат на удаление видит, что баланс нарушен, то есть он равен 2 (-2), то он "перебрасывает" запись справа(слева) от корня, несмотря на то, будет он удалять запись или нет. После этого преобразования баланс станет равен 0. Следовательно, если автомату на удаление нужно будет удалить запись, баланс станет равным 1 или -1, а если не нужно, то останется 0. В любом случае, он будет принадлежать множеству $\{-1, 0, 1\}$.

К дополнительным исключениям так же можно отнести тот факт, когда автомат на удаление удаляет запись, в следующий такт в эту вершину не должен перехо-

дить ни один автомат, чтобы избежать конфликтов. Это так же вносит коррективы в функционирование автоматов. Например, автомат на поиск должен будет один такт постоять на месте, если в вершину, в которую он собирается перейти, будет удаляться (тип вершины равен "в"). Будет доказано, что на корректность работы это не повлияет, и этот автомат на поиск все равно найдет искомую запись, при этом он позволит автомату на удаление удалить свою, не беспокоясь о том, что в удаляемую вершину кто-то может перейти.

Таким образом, описан основной смысл всех преобразований. Далее в доказательстве будет большой разбор случаев, а именно, как и что должен делать автомат, чтобы избежать конфликта с другими автоматами в различных ситуациях.

Формальное описание преобразований

Сначала рассмотрим случай, когда ИГ состоит из одной вершины (корень), или из корня может выходить одно ребро в вершину с типом "у". Естественно, что запросы на поиск и удаление видят, что ИГ состоит из одной вершины или в нем есть одно ребро в "пустую" вершину, поэтому сразу завершают свое функционирование. Одно из преобразований, изображено на рисунке 2.75. Знак "|" на рисунке означает "или". То есть, это общее преобразование для автомата на поиск и удаление. Второе преобразование, когда в пустой базе данных есть одно ребро, ведущее в вершину с типом "у", будет разъяснено ниже.

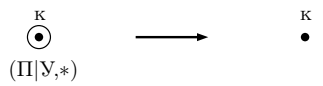


Рис. 2.75: Преобразование автоматов на удаление и поиск. Случай пустой базы данных.

Рассмотрим теперь поведение автомата на вставку в этом случае. Автомат вставляет запись слева от корня. Эти преобразования изображены на рисунке 2.76.

Разберем детально это преобразование. Автомат создает ребро и вершину, которой приписывает запись, которую он обслуживает. Ребру присваивает предикат $f^{(\text{а},-1,\text{В}_{\leftarrow})}_{P(*),1,*}$. У этого предиката есть верхний индекс $(\text{а},-1,\text{н})$ и нижний индекс $(P(*),1,*)$. Первый аргумент нижнего индекса означает произведение простых чисел, соответствующих записям слева от корня. Функция P — эта та же функция, что использовалась в теореме 5. Например, если $*$ равнялась бы 5, то нижний индекс был бы равен $(P(5),1,5)=(11,1,5)$, так как пятое простое число это 11 (2,3,5,7,11). Первый аргумент верхнего индекса означает, что числа $P(*)$ и 1 актуальны. Второй параметр верхнего индекса -1 означает баланс, то есть, что слева от корня записей на одну больше, чем справа. Третий параметр означает, что была вставка влево.

Разберем теперь случай, когда ИГ состоит из одного ребра, ведущего в вершину "в". Этот случай аналогичен уже разобранному выше, так как в базе данных нет

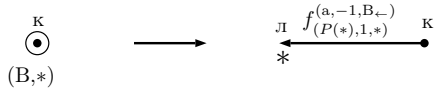


Рис. 2.76: Преобразование автомата на вставку. Случай пустой базы данных.

записей, за исключением вставки. Вставка вставляет новую запись в противоположную сторону. Ниже будет разъяснено, как правильно должны удаляться граничные вершины ИГ, чтобы избежать конфликтов.

Случай, когда ИГ состоит из одного ребра ведущего в вершину "л", попадает под общий случай, когда из корня выходит два ребра, считая условно, что второе ребро существует, и его предикат равен $f^{\text{н},0,\text{н}}_{1,1,1}$.

Теперь перейдем к общему случаю. Опишем алгоритма автомата на поиск.

Поиск, первый такт

В дальнейшем фразу "меняет ребро" нужно понимать как "меняет предикат, принадлежащий ребру". Отдельно будем оговаривать ситуации, когда ребро будет удаляться или будут меняться инцидентные вершины.

Первый такт любого автомата заключается в определении искомой записи в базе данных. Для этого автомат преобразует одно или два ребра, выходящих из корня. Количество преобразуемых ребер зависит от входной окрестности. Основная цель — не создать конфликт. Автомат может понять, будет ли преобразовываться одно из ребер или нет другим автоматом. Если оно будет преобразовываться, то он меняет другое ребро, иначе меняет оба ребра.

Опишем сначала ситуации, когда автомат на поиск меняет два ребра. Тип корня должен быть "к", а типы вершины слева от корня и справа от корня должны принадлежать множеству {"л", "у"}.

Теперь опишем входные окрестности, при которых автомат на поиск меняет только левое или правое ребро. Если тип корня равен "$\text{к}_{\leftarrow}$" ("$\text{к}_{\rightarrow}$"), то автомат меняет только правое (левое) ребро. Так же, если тип левой (правой) вершины равен "в", то он меняет только правое (левое) ребро.

Осталось описать, как именно меняется ребро, то есть каким образом меняется его предикат. Для этого автомат определяет актуальное ребро. Понятие актуального ребра было описано выше. От актуального ребра важно знать, какие записи есть слева, а какие справа. Идея преобразования ребра аналогична идее, используемой в теореме 5.

Преобразования соответствующие этим случаям изображены на рисунке 2.77. Функция ϕ^f_{Π} описывается таблицей (2.5). Столбцы 2—3 таблицы 2.6 содержат значения параметров актуального ребра, а в столбце 4 представлен новый предикат ребра. Разберем подробнее функцию ϕ^f_{Π}. Точкой обозначены параметры, которые не будут меняться. Параметр t_1 нового предиката в любом случае становится равен "а".

Тип t_3 равен "н" если искомой записи нет в базе данных и равен "$\Pi_{\leftarrow}$" ("$\Pi_{\rightarrow}$") если запись находится в левой (правой) половине ИГ относительно корня.

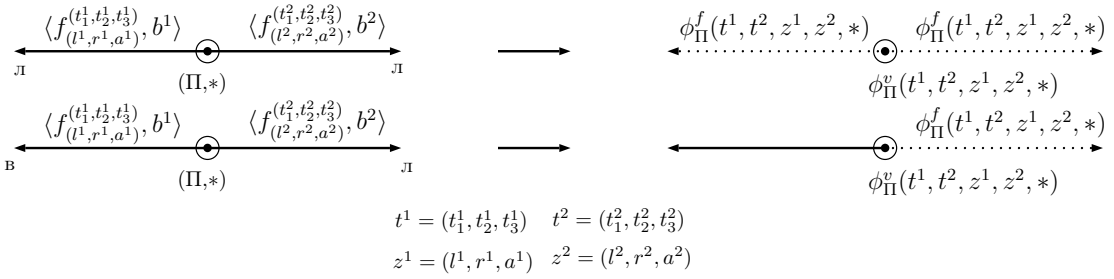


Рис. 2.77: Преобразование первого такта автомата на поиск в общем случае.

	$l \vdots P(*)$	$r \vdots P(*)$	ϕ_{Π}^{f}
1	нет	нет	$f_{(\cdot,\cdot,\cdot)}^{(\text{а},\cdot,\text{н})}$
2	да	-	$f_{(\cdot,\cdot,\cdot)}^{(\text{а},\cdot,\Pi_{\leftarrow})}$
3	-	да	$f_{(\cdot,\cdot,\cdot)}^{(\text{а},\cdot,\Pi_{\rightarrow})}$

(2.5)

Поиск, второй и следующие такты

Второй такт функционирования автомата на поиск заключается в определении, есть ли запись в базе данных или нет. Если записи нет, то автомат завершает свое функционирование. Он понимает, что записи нет, если значение t_3 ребра, которое он менял на предыдущем такте равно "н". Если же значение t_3 равно "$\Pi_{\leftarrow}$" ("$\Pi_{\rightarrow}$"), то он понимает, что запись есть, и она находится слева (справа) от корня.

В этих случаях автомат применяет следующий алгоритм. Он меняет свое внутренне состояние на двигаться влево (вправо). Если предикат слева (справа) принимает значение один, то автомат выдает в качестве ответа запись приписанной вершине слева (справа), за небольшим исключением, которое поясним ниже. Если значение предиката равно 0, то автомат смотрит на тип вершины слева (справа). Если он не равен "в", то он перемещается в эту вершину, иначе он остается на месте. Напомним, что тип вершины "в" в данном алгоритме означает, что эта вершина будет удаляться на следующий такт. Поэтому автомат на поиск не может перейти в эту вершины, чтобы избежать конфликта.

Исключение, которое упоминалось выше, состоит в следующем. Автомат на поиск может стоять в корне некоторое количество тактов из-за того, что вершина куда он должен двигаться имеет тип "в". Поэтому он следит за параметром актуального ребра. Если оно меняется и становится таким, что его параметр t_3 принял значение

"$\text{В}_{\rightarrow}$" ("$\text{В}_{\leftarrow}$"), где "направление вставки" совпадает с направлением движения автомата на поиск, то автомат это запоминает. На следующий такт вставка вставит на место вершины с типом "в" новую запись, которая могла совпасть с той, которую ищет поиск. Если такое происходит, то поиск пропускает эту вершину и продолжает искать свою, несмотря на то, что предикат был равен одному. Ниже подобная ситуация будет объяснена для автомата на удаление.

Итак опишем теперь алгоритм, который применяет автомат на вставку.

Вставка, первый такт

Здесь будет много случаев в зависимости от типов вершин входной окрестности и баланса. Рассматриваемый случай будем писать курсивом.

Обе вершины слева и справа от корня имеют тип "л", корень имеет тип "к"

	t_2	$l \dot{:} P(*)$	$r \dot{:} P(*)$	$\phi^f_{\text{В}}$	$\phi^v_{\text{В}}$
1	-	да	-	$f^{(\text{а},\cdot,\text{н})}_{(l,r,\cdot)}$	к
2	-	-	да	$f^{(\text{а},\cdot,\text{н})}_{(l,r,\cdot)}$	к
3	$-2,-1,0$	нет	нет	$f^{(\text{а},t_2+1,\text{В}_{\rightarrow})}_{(l,r*P(*),\cdot)}$	$\text{к}_{\rightarrow}$
4	$1,2$	нет	нет	$f^{(\text{а},t_2-1,\text{В}_{\leftarrow})}_{(l*P(*),r,\cdot)}$	$\text{к}_{\leftarrow}$

(2.6)

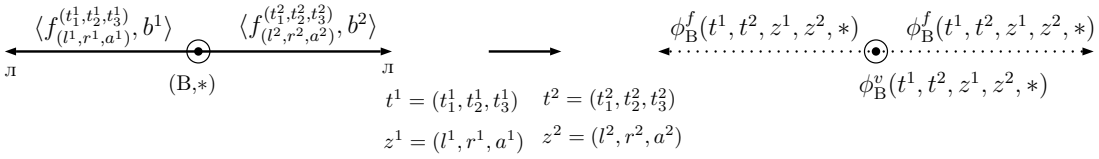


Рис. 2.78: Преобразование на вставку. Первый такт, общий случай.

Преобразование, описывающее первый такт автомата на вставку, когда слева и справа от корня вершины имеют тип "л", изображено на рисунке 2.78. Опишем функцию $\phi^f_{\text{В}}$, которую использует автомат. Обратим внимание, что после преобразования слева и справа от корня будут одинаковые предикаты.

Функции $\phi^f_{\text{В}}$ и $\phi^v_{\text{В}}$ описываются таблицей (2.6). Столбцы 2—4 таблицы 2.6 содержат значения параметров актуального ребра, а в столбцах 5—6 представлен новый предикат ребра и новый тип корня.

Строчки 1—2 таблицы, изображают изменение предиката, когда автомат вставляет уже существующую запись. Строчки 3—4 таблицы соответствуют случаям, когда автомат вставляет новую запись. Функция $\phi^v_{\text{В}}$ меняет тип вершины. При этом не

важно какой тип был у корня. Если автомат на вставку собирается вставить запись влево (вправо), то функция $\phi^v_{\text{В}}$ делает тип вершины "к$_{\leftarrow}$" ("к$_{\rightarrow}$"). Это будет использоваться, чтобы избежать конфликтов между вставкой и удалением. Заметим, что типы "к$_{\leftarrow}$" ("к$_{\rightarrow}$") введены для лучшего понимания. На самом деле, им соответствуют типы в(л), а автомат знает, что, если корень имеет тип в (л), то его нужно воспринимать как тип "к$_{\leftarrow}$" ("к$_{\rightarrow}$"). Далее везде будем писать типы "к$_{\leftarrow}$" ("к$_{\rightarrow}$") вместо в (л).

Разберем подробнее функцию $\phi^f_{\text{В}}$. Прежде всего отметим, что точкой обозначены параметры, которые не будут меняться. Параметр t_1 нового предиката в любом случае становится равен "а". Начнем со строчек 1—2. В них представлены случаи, когда вставляемая запись уже существует. Параметр t_3 нового предиката становится равным "н", что и будет означать, что вставлять запись не нужно.

Разберем теперь строчки 3—4 таблицы 2.6. Это случаи, когда вставляется запись, которой в базе данных нет. В этом случае функция $\phi^f_{\text{В}}$ говорит, куда вставке нужно будет вставить запись $*$ на следующий такт. Выбор осуществляется, исходя из баланса t_2. Баланс — это разница между количеством записей справа от корня и количеством записей слева от него. Например, баланс -1 означает, что слева от корня на одну запись больше, чем справа. Следовательно, в этом случае вставке лучше вставить запись вправо от корня, так как после этого преобразования баланс станет равен 0. Ниже будет доказано, что баланс будет всегда принадлежать множеству {-2, -1,0,1, 2}.

Вершина слева (справа) от корня имеет тип "в", а другая тип "л" или корень имеет тип "к$_{\leftarrow}$" ("к$_{\rightarrow}$")

Аналогично разобранному выше алгоритму поиска, в этом случае можно поменять только одно ребро. Опишем преобразование, когда вершина справа имеет тип "в" или тип корня "к$_{\leftarrow}$". Случай, когда вершина слева имеет тип "в" или тип корня "к$_{\rightarrow}$", будет симметричен предыдущему. По сути это то же преобразование, что и в первом случае, только автомат на вставку меняет ребро ведущие в вершину с типом "л", а другое ребро не изменяет. Но есть небольшое исключение. Оно заключается в том, что, если можно вставить в сторону вершины с типом "в" так, чтобы баланс принадлежал множеству $\{-1, 0, 1\}$, то автомат будет вставлять в эту сторону. Пример этого преобразования изображен на рисунке 2.79. Функции $\widetilde{\phi}$ отличатся от уже рассмотренных выше функций ϕ, описанным выше свойством, что если нет разницы куда вставлять, с точки зрения баланса, то лучше вставлять запись в сторону вершины "в".

Вершина слева или справа от корня имеет тип "в", другая тип "у"

Этот случай аналогичен предыдущему, за исключением того, что автомат меняет предикат, ведущий в вершину с типом "у".

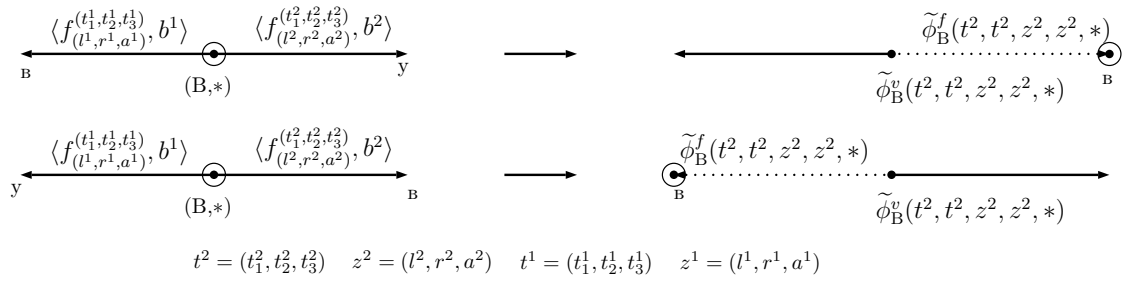


Рис. 2.79: Преобразования на вставку. Первый такт, меняется одно ребро.

Вершина слева (справа) от корня имеет тип "л", вершина справа (слева) тип "у"

Этот случай аналогичен уже рассмотренному выше, когда одна из вершин имеет тип "в", а другая тип "л", но есть небольшое исключение. Разница заключается в том, что автомат меняет оба ребра, выходящих из корня на актуальные.

Напомним, что вставка всегда вставляет так, чтобы баланс после ее преобразования принадлежал множеству $\{-1, 0, 1\}$.

Вставка, второй такт

Автомат на вставку смотрит тип корня. Если тип корня равен "к", то автомат на вставку завершает функционирование.

Рассмотрим, оставшиеся случаи. Если тип корня не равен "к", то он обязательно равен "к$_{\leftarrow}$" или "к$_{\rightarrow}$". Разберем первый из них, второй разбирается аналогично. В этом случае возможно два варианта.

Вставка влево и вершина слева от корня имеет тип "л"

В этом случае автомат применяет преобразование, изображенное на рисунке 2.80. Ребро, нагрузка которого не меняется, обозначено одной палочкой. Ребро, изображенное сплошным отрезком, не изменяется вообще.

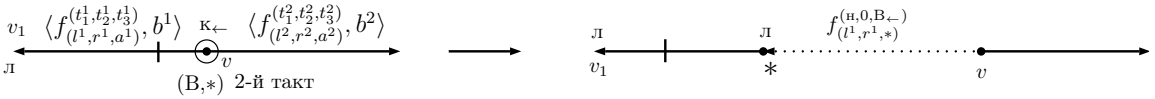


Рис. 2.80: Преобразование на вставку второй такт. Вставляется в сторону вершины с типом "л".

Вставка влево и вершина слева от корня имеет тип "в" или "у"

В этом случае автомат применяет преобразование, изображенное на рисунке 2.81. Автомат вставляет новую запись вместо вершины, которую нужно было удалить. Конфликтов не возникнет, так как автомат на удаление по типу корня поймет, что вставка произойдет в этот такт и не будет удалять свою вершину.

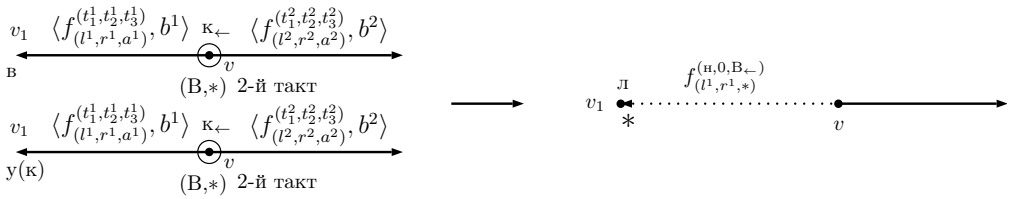


Рис. 2.81: Преобразование на вставку второй такт. Вставляется в сторону вершины с типом "в" или "у".

Опишем теперь преобразования, которые применяет автомат на удаление.

Удаление, первый такт

Сделаем небольшое замечание, связанное с пустыми запросами. Первый такт автомата на удаление, как и автомат на вставку, меняет тип корня, если нужно будет изменить базу данных и не нужно делать переброску вершины. Чтобы различить, кто именно поменял тип корня - автомат на вставку или автомат на удаление - нужно посмотреть на параметр t_3 актуального ребра. Если его тип "У$_\leftarrow$" или "У$_\rightarrow$", то этот тип сделал автомат на удаление, иначе — автомат на вставку. Такая нагроможденная конструкция нужна лишь для того, чтобы обеспечить актуальность в случае пустых запросов, которые могли придти после автомата на удаление. В дальнейшем, если будем писать тип корня "к$_\leftarrow$" ("к$_\rightarrow$"), то это будет значить, что его создал автомат на вставку. Так же будем считать, что автомат на удаление меняет тип корня только на "к", поэтому в таблице 2.7 нет функции $\phi^v_{\text{У}}$. Это упростит доказательство, не ограничивая его в общности.

Как и другие автоматы, автомат на удаление меняет либо одно ребро, выходящее из корня, либо два. Одно ребро автомат меняет, если баланс принадлежит множеству $\{-1, 0, 1\}$, и выполнено одно из следующих условий: одна из соседних с корнем вершин имеет тип "в"; тип корня равен "к$_\leftarrow$" или "к$_\rightarrow$". В других случаях автомат меняет два ребра, выходящих из корня. Преобразование, соответствующее этому случаю, изображено на рисунке 2.82.

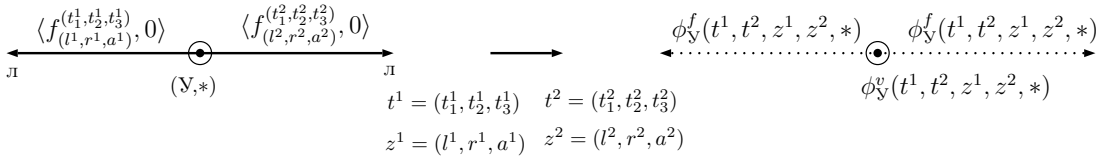


Рис. 2.82: Преобразование на удаление без переброски.

	t_2	$l \dot{:} P(*)$	$r \dot{:} P(*)$	$\phi_{\text{У}}^f$
1	-	нет	нет	$f_{(l,r,\cdot)}^{(\text{а},\cdot,\text{н})}$
2	-1,0,1	да	-	$f_{(l/P(*),r,\cdot)}^{(\text{а},t_2+1,\text{У}_{\leftarrow})}$
3	-1,0,1	-	да	$f_{(l,r/P(*),\cdot)}^{(\text{а},t_2-1,\text{У}_{\rightarrow})}$

(2.7)

Первая строчка таблицы 2.7 соответствует случаю, когда удаляемой записи нет в базе данных. Вторая и третья строчки соответствуют случаям, когда удаляемая запись есть в базе данных.

При балансе, равном 2 (-2), автомат на удаление "перебрасывает" вершину справа (слева) от корня вправо (влево), меняет тип переброшенной вершины на "в", переходит в нее для последующего удаления во второй такт. Переброска осуществляется алгоритмом вставки, то есть считаем, что вставляется запись равная перебрасываемой вершине. Например, если перебрасываем вершину слева направо и справа от корня, вершина имеет тип "у" или "в", то перебрасываемая вершина присваивается этой вершине. Иначе, создается новая вершина справа от корня, которой и присваивается перебрасываемое значение.

Тип корня меняется на "к". Если значение обоих предикатов равно нулю, то типы вершин соседних с корнем не меняется. Отдельно опишем, как меняется тип вершин соседних с корнем, если значение предиката на запросе равен одному, то есть, удаляемая запись находится рядом с корнем. Если тип вершины в которую ведет это ребро имеет тип "в", то автомат не меняет ее. Если ее тип равен "л", то он меняет ее тип на "у", если удаляемая запись есть в базе данных, то есть функция, которая принимает тип вершины, зависит от того, есть удаляемая запись в базе данных или нет. Также автомат на удаление меняет вершины с типом "у" на тип "в", если такая есть, и переходит в нее для последующего удаления.

Удаление, второй и последующие такты

Сначала разберем второй такт автомата на удаление. Как и в случае с поиском, автомат на удаление смотрит, есть ли запись, которую он удаляет, в базе данных или нет. Если есть, то в какой половине списка она находится (слева или справа от корня).

Если автомат находится в состоянии дополнительного удаления вершины, то он удаляет ее, согласуясь с правилами удаления вершины. После того, как он ее удалит или по алгоритму, он должен будет завершить функционирование, и автомат сразу переходит к удалению собственной вершины. Алгоритм удаления вершины, в которой находится автомат, будет описан ниже.

Опишем общий случай, когда автомату не нужно делать дополнительные удаления. Если удаляемой записи в базе данных нет, автомат на удаление завершает свое функционирование. Если запись есть, то возможны варианты.

Напомним, что из-за пустых запросов пришлось усложнить алгоритм автомата на удаление. А именно тем фактом, что они меняют тип корня, точно так же, как это делают автоматы на вставку. Автомат на удаление должен точно знать, этот тип сделал автомат на удаление или на вставку, когда он должен удалить вершину соседнюю с корнем. Это нужно, чтобы избежать конфликтов.

Если автомат на удаление должен удалить вершину рядом с корнем, то он должен будет понять, кто сделал тип "$У_{\leftarrow}$" или "$У_{\rightarrow}$" в корне. Если это сделал он, значит он сможет удалить вершину, соседнюю с корнем, иначе это сделал автомат на вставку, и в этом случае он не будет ее удалять. Автомат на удаление определяет этот факт, смотря на ребро, ведущее из корня в вершину, которое он должен удалить. Если ребро было актуальным, то его параметр t_3 должен был измениться на "$В_{\leftarrow}$" или "$В_{\rightarrow}$". Если же это ребро было не актуальным, а затем стало актуальным, с типом "$В_{\leftarrow}$" или "$В_{\rightarrow}$", то это опять признак того, что пришел автомат на вставку. В других случаях этот тип сделал автомат на удаление.

Перейдем к вариантам, когда автомат на удаление должен найти вершину. Пусть это вершина, которая находится слева от корня и соседняя с ним. Случай, когда эта правая вершина разбирается аналогично. В этом случае автомат меняет тип вершины слева от корня на "в", переходит в нее и меняет свое состояние на "удаление вершины". Возможен интересный случай, когда эта вершина на следующий такт уже будет находиться не рядом с корнем. Это могло произойти, если другой автомат на удаление применил преобразование переброски.

Если автомат продолжает функционирование, то он смотрит тип корня. Если он равен "$к_{\leftarrow}$", то он завершает свое функционирование, так как автомат на вставку вставит на место этой вершины новую запись. Так же если автомат видит, что баланс стал равен -2, то он также завершает функционирование, так как следующий за ним автомат на преобразование базы данных уберет эту вершину. В других ситуациях, автомат на удаление смотрит тип вершины слева. Если он равен "в", то он остается в этой вершине, и на следующий такт повторяет тот же алгоритм, что и в предыдущий такт. Случай удаления вершины не вблизи корня отличается только тем, что не смотрится его тип и баланс.

Если вершину, которую должен удалить автомат, находится не рядом с корнем, то автомат переходит в состояние поиска этой вершины. Алгоритм поиска совпадает с уже описанном выше для автомата на поиск. Отличие состоит в том, что, когда

автомат на удаление находит вершину, которую нужно удалить, он меняет ее тип на "в" и переходит в нее. Далее смотрит, не будет ли удаляться вершина слева (справа) от него, если он двигался влево (вправо). Он понимает это так же по типу вершины. Если тип вершины слева (справа) равен "в", то значит она будет удаляться иначе нет. Если вершина будет удаляться, то автомат на удаление ждет такт, иначе он удаляет эту вершину. Начиная со следующего такта автомат повторяет алгоритм предыдущего такта. Ниже будет доказано, что максимальное количество подряд идущих удалений — три, поэтому автомат максимально может стоять два такта.

Отметим, что когда автомат удаляет вершину, вместе с ней он удаляет и ребро, которое ведет из этой вершины.

Отдельно стоит описать поведение автомата на вставку, когда, находясь в корне, автомат не может двигаться в сторону вершины, которую он должен удалить из-за вершины с типом "в". Если такая ситуация возникает, то автомат следит за типом корня. Допустим, автомат на вставку должен был двигаться влево от корня. Тогда, если он видит, что во время простоя в вершине, тип корня стал равен "к$_{\leftarrow}$", а вершина слева все еще имеет тип "в", то он переходит в специальное состояние, готовясь к следующему такту. Если предикат слева от корня будет равен 1 на запросе, то это не та запись, которую он должен удалить. Он должен ее воспринимать как обычную запись и двигаться дальше к удалению собственной. В других случаях автомат на удаление действует стандартным образом.

Нужно сделать небольшое замечание в алгоритме удаления листовой вершины графа (самой левой или правой вершины графа), которое не меняет его смысл, но позволяет избежать конфликтов. При удалении самой левой или самой правой вершины графа, автомат удаляет только вершину и не удаляет ребро в нее ведущее. Автоматы же понимают, что если они видят ребро, которое не ведет в вершину, то значит они находятся в последней вершине графа. Такое необычное поведение нужно, чтобы избежать конфликтов изменения предпоследней вершины. Так как автомат на удаление может изменить ее тип на "в" в тот же такт, что и удаление последнего ребра в графе. Это не влияет на общее описание алгоритма, поэтому будем считать, что автомат может сразу удалить последнее ребро, подразумевая под этим алгоритм описанный выше.

Доказательство, что полученный ПДИГ, удовлетворяет условию теоремы

Итак, осталось доказать, что приведенный выше алгоритм автоматов работает корректно и с заявленной сложностью. Для этого докажем несколько вспомогательных лемм.

Лемма 5. *Актуальное ребро определяется в каждый такт функционирования ПДИГ — корректно.*

Доказательство. Эта лемма утверждает, что автомат может корректно определить актуальное ребро. Другими словами, если из корня выходит два ребра с типом t_1 равным "а", то значения t_2, l, r у них совпадают, кроме того эти значения соответствуют ИГ в данный так.

В случае пустой базы данных возможен случай, когда ребер нет, или есть одно ребро, ведущее в вершину с типом "у". В обоих случаях автомат не будет узнавать актуальное ребро, он понимает, что в данный так ИГ пуст и будет выполнять, описанный выше алгоритм в этих случаях. В частности, автоматы на поиск и удаление завершают свое функционирование, а автомат на вставку вставляет свою запись вместо вершины с типом "у" и меняет предикат, ведущей к ней на актуальный.

Если база данных состоит из одной записи, то она могла возникнуть двумя способами. В первом — из пустой базы данных, добавлением новой записи, а во втором — удалением из двухэлементной базы данных одного элемента. Первый вариант уже разобран выше, в этом случае в ИГ всегда есть ровно одно актуальное ребро.

Рассмотрим теперь второй случай. Он совпадает и с общим случаем, поэтому объединим их в один.

Сначала разберем вариант, когда пустых запросов нет, то есть в каждый такт времени к базе данных поступает либо запрос на поиск, либо на вставку, либо на удаление. Заметим, что после первого такта автомата всегда существует как минимум одно актуальное ребро, поэтому если пустых запросов нет, то актуальное ребро будет. Кроме этого, если автоматы делают преобразования во второй и последующие такты, то предикаты, которые они создают, имеют тип t_1 равный "н", то есть не актуальный. Поэтому не возникнет ситуации, когда после первого такта автомата возникнет два актуальных ребра с разными параметрами.

Осталось рассмотреть вариант, когда есть пустые запросы. Здесь так же возможны подслучаи.

За автоматом на поиск поступил один или несколько пустых запросов. Автомат на поиск в первый такт своего функционирования создал одно или два актуальных ребра.

Допустим он создал одно актуальное ребро, а другое ребро не изменял. Такие ситуации описаны выше в алгоритме. Они возникают по нескольким причинам. Первая из них, если тип корня равен "к$_{\leftarrow}$" или "к$_{\rightarrow}$". Эта означает, что перед автоматом на поиск первый не пустой запрос был на вставку. Если тип корня равен "к$_{\leftarrow}$", то вершина справа от корня обязательно имеет тип "л" или "у". В этом можно убедиться, рассмотрев все преобразования на вставку. Таким образом, ребро справа от корня и вершина справа от него изменяться не будут до следующего запроса. Действительно, автоматы на поиск и удаление, которые поступили раньше, уже не будут менять эту вершину и ребро. Следовательно, неважно, какое количество пустых запросов поступило к базе данных, все равно это актуальное ребро останется до следующего запроса.

Второй вариант, когда одна из вершин слева или справа от корня имела тип

”в”. Заметим, что в этом случае другая вершина обязательно имеет тип ”л” или ”у” в силу алгоритма автоматов. В этой ситуации опять же автомат на поиск сделает актуальным ребро ведущее в вершину с типом ”л” или ”у”. Даже если до этого был автомат на удаление вершины с типом ”л”, соседней с корнем, то этот автомат на удаление не изменит ребра, в нее ведущее, а только поменяет тип вершины с ”л” на ”у”. Следовательно, это ребро останется актуальным до следующего не пустого запроса.

Рассмотрим теперь случай, когда автомат на поиск создал сразу два актуальных ребра. Это означает, что тип вершин слева и справа от корня был ”л” или ”у”, а так же тип корня ”к”. Аналогично разобранному выше случаю с одним актуальным ребром, на последующие такты может поменяться только тип вершин с ”л” на ”у”, а оба ребра останутся актуальными до следующего не пустого запроса.

Разберем случай, когда за автоматом на вставку поступили один или несколько пустых запросов. В отличии от поиска автомат на вставку во второй такт своего функционирования может изменить ребро выходящее из корня на не актуальное. Поэтому, если до этого это было единственное актуальное ребро, и он его изменил на не актуальное, то оба ребра, выходящих из корня, будут иметь тип t_1 равный ”н”, то есть не актуальным. Но тип корня не поменялся и остался равным ”к$_{\leftarrow}$” или ”к$_{\rightarrow}$”. ”Стрелка” и будет индикатором актуальности ребра в этом случае, поэтому это вошло в определение актуального ребра. Если тип корня ”к$_{\leftarrow}$”, то актуальным считаем левое ребро иначе правое. Изменяться эти ребра больше не будут до следующего не пустого запроса.

Осталось рассмотреть случаи, когда за автоматом на удаление поступают пустые запросы. Этот случай аналогичен разобранному выше случаю со вставкой. В определении преобразования автомата на удаление в самом начале была оговорка, что автомат на удаление точно так же, как и автомат на вставку, меняет тип корня. Смена типа корня позволит понять, какое ребро актуальное, даже если его тип t_1 равен ”н”. Лемма доказана. □

Лемма 6. *Справедливы следующие два утверждения. ПДИГ функционирует бесконфликтно. Если автомат на удаление применяет преобразование переброски, то он обязательно перебросит существующую запись из базы данных.*

Доказательство. Рассмотрим первый такт произвольного автомата. Покажем, что преобразование, которое он применит не будет конфликтовать с другими автоматами. Алгоритмы всех автоматов в первый такт уже были описаны. Так же было пояснено, почему автомат меняет в некоторых случаях только одно ребро, выходящее из корня, а не два. Автомат делает это как раз для того, чтобы избежать конфликтов. Осталось только показать, что автомат сможет поменять хотя бы одно ребро. Рассмотрим все случаи, когда он не смог бы это сделать, и докажем, что таких ситуаций не возникнет во время функционирования. Таких вариантов три: вершина слева и справа от корня имеет тип ”в”; вершина слева от корня имеет тип ”в” и корень имеет

тип "к$_{\rightarrow}$"; вершина справа от корня имеет тип "в" и корень имеет тип "к$_{\leftarrow}$". Докажем, что ни один их этих случаем не возможен.

Рассмотрим первый случай, когда вершина слева и справа от корня имеет тип "в". Заметим, что автоматы на поиск и пустые запросы не меняют типы вершин. Также видно, что автоматы на вставку не создают новых вершин с типом "у" и "в", а наоборот могут поменять тип "у" на "л" или "в" на "л". Алгоритм удаления устроен так, что если одна из соседних с корнем вершин имеет тип "в", а другая тип "л", то автомат на удаление либо оставит ей тип "л", либо сделает ей тип "у". Осталось рассмотреть последний случай, когда в первый такт своего функционирования автомат на удаление меняет тип "у" на "в", другая вершина имела тип "в", и она не исчезает на следующий такт. Для этого нужно более детально посмотреть, когда появляется тип "в". Она появляется минимум на следующий такт за удалением, которое сделало тип "у". Вершина с типом "в" могла не удалиться по причине того, что рядом с ней была еще одна вершина с типом "в". Докажем, что это невозможно, то есть вершина с типом "в" данной ситуации обязательно успела бы удалится. Ниже будет доказано, что соседних вершин с типом "в" может быть не больше трех, следовательно, любая вершина с типом "в" исчезает не более, чем за три такта. Под исчезает понимаем два варианта: вершина удаляется или на ее место вставляется другая запись. Вершина с типом "в" рядом с корнем появляется минимум за два такта. Поэтому на третий такт она обязательно удалится.

Рассмотрим второй (третий) случай, когда вершина слева (справа) от корня имеет тип "в", а корень имеет тип "к$_{\rightarrow}$" ("к$_{\leftarrow}$"). Оба варианта симметричны, поэтому разберем случай, когда вершина слева от корня имеет тип "в", а корень имеет тип "к$_{\rightarrow}$". Допустим такая ситуация возможна. Понятно, что одновременно за один такт такого не могло произойти. Тогда нужно разобрать два варианта. Сначала тип вершины слева от корня стал равен "в", а затем тип корня стал равен "к$_{\rightarrow}$", и, наоборот, сначала тип корня стал равен "к$_{\rightarrow}$", а затем вершина слева от корня стала иметь тип "в".

Пусть сначала тип корня стал равен "к$_{\rightarrow}$". Тогда, чтобы тип вершины слева от корня стал равен "в", должен придти запрос на удаление. В другом случае он не примет этот тип. Если поступит запрос на удаление, то он поменяет тип корня на "к".

Рассмотрим теперь случай, когда вершина слева от корня имеет тип "в". В этом случае автомат на вставку сможет только сделать тип корня равным "к$_{\rightarrow}$". Но как уже было разобрано выше вершина с типом "в" удаляется за один такт, поэтому в этом случае не получится конфигурации, когда вершина слева имеет тип "в", а корень имеет тип "к$_{\rightarrow}$".

Итак, разобраны все случаи, когда автомат не сможет поменять ребро в первый такт своего функционирования. Доказано, что таких конфигураций не возникнет, поэтому любой автомат в первый такт своего времени не образует конфликтов с другими автоматами. Рассмотрим преобразования, начиная со второго такта его

функционирования. Если это автомат на поиск, то он не образует конфликтов, так как только перемещается по ИГ, с учетом того, что в алгоритме поиска запрещается переходить в вершину, которая возможно будет удаляться. Автомат на вставку делает преобразование только во второй такт функционирования. При этом он сообщает автоматам, куда он собирается вставить запись. Поэтому автомат на вставку не образует конфликтов.

Осталось рассмотреть автомат на удаление. Автомат на удаление может изменить тип вершин. Но это не вызывает конфликтов. Этот случай уже подробно разобран в самом алгоритме удаления. Аналогично автомату на поиск, перемещение по графу автомата на удаление так же не конфликтует с другими автоматами. Преобразование, которое удаляет вершину, так же подробно разобрано в алгоритме удаления. Таки образом, доказано, что автоматы не конфликтуют.

Докажем теперь второе утверждение леммы 6. Для этого нужно доказать, что не возникнет конфигураций, когда автомат на удаление должен будет перебросить вершину типа "в" или "у" через корень. Не ограничивая общности рассуждения, нужно перебросить вершину слева от корня вправо. В этом случае баланс должен быть равен -2, чтобы случилась переброска слева направо.

Рассмотрим первый случай. Автомат должен перебросить вершину с типом "у" при балансе -2. Вершина с типом "у" появляется только после первого такта. В этом случае баланс стал равен не меньше 0 , так как удаление в левой половине увеличивает баланс на единицу. Баланс не мог бы стать -1, так как до этого он был бы равен -2, а при таком балансе автомат применил бы переброску и вершина с типом "у" не появилась бы. Если баланс стал равен 0 (1), то должно придти как минимум 2 (3) удаления, чтобы баланс стал равен -2. За два удаления вершина с типа "у" уже точно поменяет на тип "в", а этот случай уже был разобран выше и доказано, что не бывает ситуаций, когда вершина с типом "в" перебрасывается.

Осталось доказать факт, что не бывает трех подряд вершин на удаление. Этот факт следует из алгоритма переброски. В одну сторону графа может пойти не более трех подряд удалений, после чего обязательно произойдет переброска. Действительно, переброска происходит при балансе 2 или -2. Каждое удаление вправой (влевой) части уменьшает (увеличивает) баланс на 1. Следовательно, если все запросы на удаление поступают в одну сторону, то максимальное количество равняется 3, так как $-2 = 1 - 3$ и $2 = -1 + 3$. Во время переброски, автомат удаляется от группы удалений на 2 ребра. Причем баланс становиться равен -1 (1), а это значит, что в эту же сторону можно послать только один запрос на удаление, после чего произойдет переброска, и опять будет удаление на 2 вершины. Следовательно, группы более чем из трех удалений образоваться не может. Лемма доказана. □

Лемма 7. *Баланс после любого запроса будет принадлежать множеству* $\{-2, -1, 0, 1, 2\}$. *Причем после преобразования на вставку баланс принадлежит множеству* $\{-1, 0, 1\}$.

Доказательство. Данная лемма предполагает, что все преобразования корректны. Этот факт доказан в лемме 6.

Лемму будем доказывать индукцией по тактам для любого потока запроса. Если такт равен 0,1 то в базе данных не более одной записи и утверждение леммы очевидно.

Рассмотрим произвольный такт. Из алгоритма на поиск следует, что автоматы на поиск не меняют баланс, пустые запросы так же его не меняют, поэтому осталось рассмотреть случай, когда к базе данных поступил запрос на вставку или на удаление.

Нетрудно заметить, что если нет запросов на удаление, а есть только запросы на вставку, то баланс будет принадлежать множеству $\{-1, 0, 1\}$. Это свойство написано в алгоритме вставки.

Допустим, пришел первый запрос на удаление. В момент его поступления баланс принадлежал множеству $\{-1, 0, 1\}$, причем тип вершин слева и справа от корня равнялся "л". После удаления возможно два варианта. Либо баланс остался во множестве $\{-1, 0, 1\}$, либо стал равен 2 или -2, что удовлетворяет предположению индукции.

Пусть поступил очередной запрос на удаление. Разберем случай, когда баланс принадлежит множеству $\{-1, 0, 1\}$. В этом случае после преобразования на удаление баланс будет принадлежать множеству $\{-2, -1, 0, 1, 2\}$, что удовлетворяет предположению индукции. Если баланс был равен -2 или 2, то автомат на удаление в любом случае применяет преобразование переброски, которое делает баланс равным 0. Далее, если ему нужно будет удалить запись, то баланс станет равным либо 1, либо -1. Если запись удалять не нужно, то баланс станет равным 0. То есть предположение индукции выполняется и в этом случае. Лемма доказана. □

Лемма 8. *Автомат на поиск работает корректно. То есть автомат на поиск x, выдает ответ $\emptyset$ если искомой записи нет и $\{x\}$, если запись есть.*

Доказательство. Автомат на поиск после первого такта узнает есть искомая запись в базе данных или нет. Если ее нет, то выдает в качестве ответа $\emptyset$. Заметим, что из леммы 5 следует, что это правильный ответ. Если запись есть, то он начинает ее искать. В алгоритме поиска уже было детально описано, почему вставка не сможет помешать поиску найти свою запись. Если автомат на поиск оказывается в одной вершине с автоматом на удаление этой же вершины, то он все равно выдаст в ответ $\{x\}$, так как автомат на удаление в этот такт только присвоит этой вершине тип "в" или "у". □

Оценка сложности поиска является нетрудным следствием вышеперечисленных лемм. Главное здесь, что баланс принадлежит множеству $\{-2, -1, 0, 1, 2\}$. Если в базе данных n записей, то в одной половине при таком балансе не больше, чем $]n/2[+2$. Автомат на поиск мог затормозить свое передвижение не более, чем на три такта. Следовательно, если автомат на поиск подождет три такта, то он не догонит ни одну

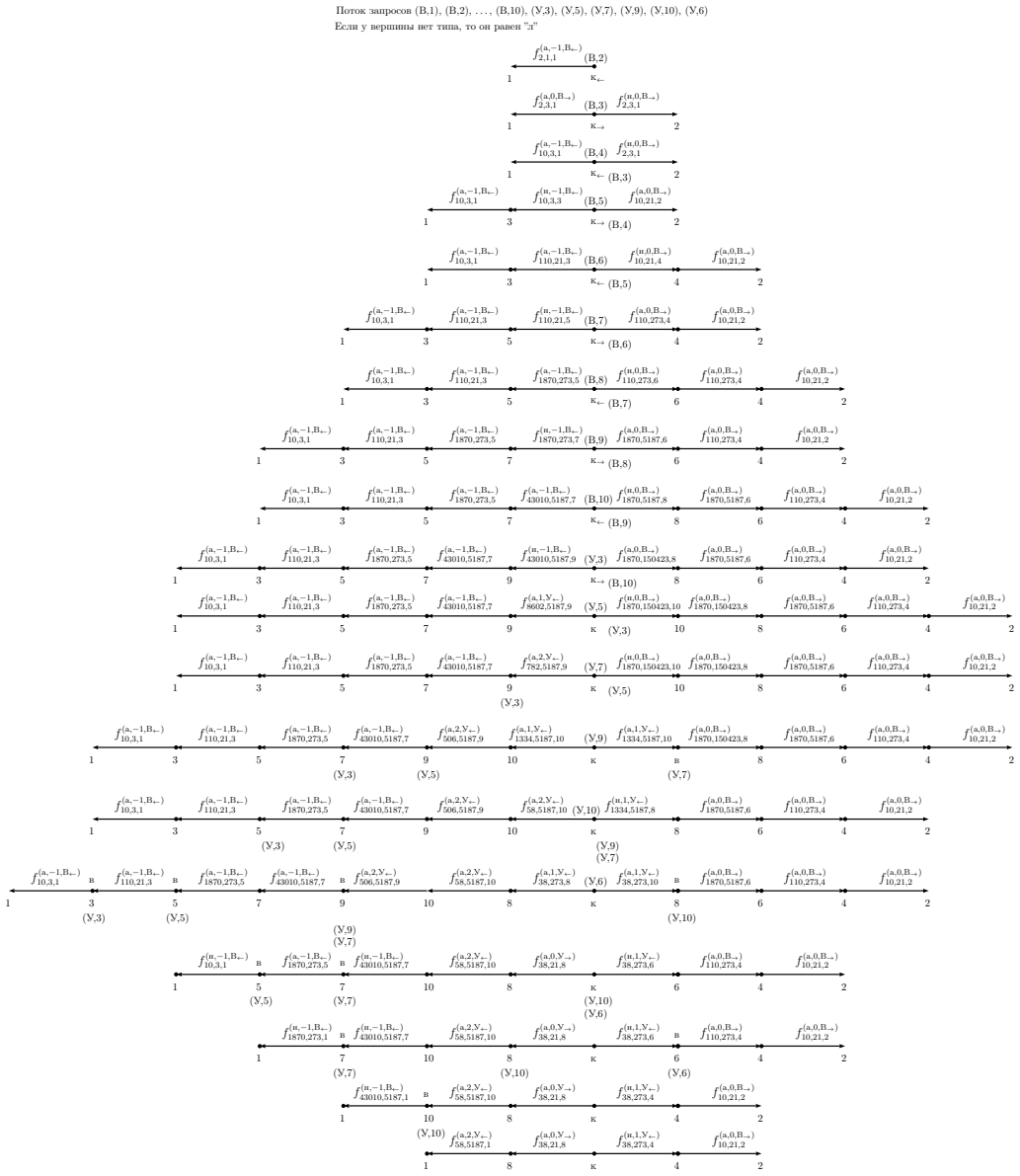


Рис. 2.83: Пример функционирования ПДИГ.

другую группу удалений и не будет больше простаивать. Поэтому количество тактов, необходимое на выдачу ответа, не превосходит $]n/2[+5$. На рисунке 2.83 приведен пример функционирования ПДИГ.

Теорема доказана.

□

Глава 3

Нижние оценки ПДИГ

1 Нижняя оценка для ПДИГ со степенью ветвления один

Теорема 7. *Для любого натурального R, не существует конечного ПДИГ типа $(1, R)$, решающего ДЗПИО для любого потока запросов.*

Доказательство. Ограничение на количество инцидентных ребер означает, что ИГ может представлять из себя объединение разных компонент связности, каждая из которых состоит либо из одной вершины, либо из двух вершин и ребра их соединяющего. Других компонент связности быть не может, так как из вершины не может выходить более одного ребра. Пример графа изображен на рисунке 3.1.

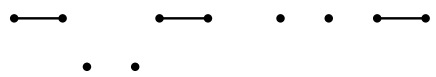


Рис. 3.1: Примера графа с $N = 1$.

Заметим, что автомат видит только вершины, между которыми есть путь длины не больше R, а это, в свою очередь, означает, что данные вершины находятся в одной компоненте связности. Следовательно, автомат не сможет переместиться из одной компоненты связности в другую, во время функционирования.

В одной вершине ИГ не может находиться более одной записи, следовательно, в одной компоненте связности не может находиться более двух записей. По определению *ПДИГ конечный*, если для произвольного потока запросов H и произвольного такта времени i существует такая константа C, $C < \infty$, что для потока запросов $H', H'(1) = H(1), \ldots H'(i) = H(i), H'(i+1) = \Lambda, \ldots, H'(i+C) = \Lambda$, автомат обслуживающий запрос $H(1)$ завершит свое функционирование.

Рассмотрим в качестве H поток, состоящий из одного запроса на вставку $H(1) = (\mathrm{B}, 1)$. По определению конечного ПДИГ, существует такая константа C_1, $C_1 < \infty$,

что для потока запросов $H^1, H^1(1) = H(1), H^1(2) = \Lambda, \ldots, H^1(1 + C_1) = \Lambda$ автомат, обслуживающий запрос $H(1)$, завершит свое функционирование.

Рассмотрим теперь поток H_2, $H_2(1) = H^1(1), \ldots, H_2(1 + C_1) = H^1(1 + C_1), H_2(2 + C_1) = (\text{В}, 2)$. Опять по определению конечного ПДИГ существует такая константа C_2, $C_2 < \infty$, что для потока запросов $H^2, H^2(1) = H_2(1), \ldots H^2(2 + C_1) = H_2(2 + C_1), H^2(3+C_1) = \Lambda, \ldots, H^2(2+C_1+C_2) = \Lambda$, автомат, обслуживающий запрос $H_2(2+C_1)$, завершит свое функционирование. Видно, что для потока запросов H_2 автоматы, обслуживающие запросы на вставку 1 и 2, завершат свое функционирование в такт не позже, чем $2 + C_1 + C_2$.

Аналогичным образом рассмотрим поток $H_3, H_3(1) = H^2(1), \ldots, H_3(2+C_1+C_2) = H^2(2+C_1+C_2), H_3(3+C_1+C_2) = (\text{В}, 3)$. По определению конечного ПДИГ, существует такая константа C_3, $C_3 < \infty$, что для потока запросов $H^3, H^3(1) = H_3(1), \ldots H^3(3 + C_1+C_2) = H_2(3+C_1+C_2), H^2(3+C_1+C_2) = \Lambda, \ldots, H^2(3+C_1+C_2+C_3) = \Lambda$, автомат обслуживающий запрос $H_2(3 + C_1 + C_2)$ завершит свое функционирование. Видно, что для потока запросов H_3 автоматы, обслуживающие запросы на вставку 1 и 2, завершат свое функционирование в такт не позже чем $3 + C_1 + C_2 + C_3$.

Итак, предъявим поток H_0, на котором ПДИГ будет работать не корректно. Пусть $C = 3 + C_1 + C_2 + C_3$, тогда $H_0(1) = H^3(1), \ldots, H_0(C) = H^3(C), H_0(C + 1) = (\text{П}, 1), H_0(C + 2) = (\text{П}, 2), H_0(C + 3) = (\text{П}, 3)$.

Очевидно, что автомат на поиск не сможет найти одну из трех записей, поэтому он не будет работать корректно на потоке запросов H_0. Действительно, в такт C база данных состоит из трех записей (1, 2 и 3). При этом в такт C ни один автомат не функционирует. Так как есть всего две вершины, то одной из трех записей нет как минимум одной в ИГ, а так как ни один автомат не функционирует, то она не может появиться в нем (без еще одного запроса на вставку). Следовательно, один из поисковых запросов не сможет найти запись, что является не корректностью работы ПДИГ. Теорема доказана. □

2 Нижняя оценка для ПДИГ со степенью ветвления два

Будем говорить, что не существует ПДИГ над базовым множеством F, который решает ДЗПИО для любого потока H так, подразумевая, что не существует базового множества преобразований $\mathcal{R}$ и ИГ над F, для которого ПДИГ решает ДЗПИО.

Справедлива следующая теорема.

Теорема 8. *Для любой функции $L, L : \mathbb{N} \to \mathbb{N}$ и для любого множества $T, |T| < \infty$ не существует конечного, селекторного ПДИГ типа (2, 1) над множеством $F(T)$, который решает логическую ДЗПИО со сложностью L.*

Доказательство. ПДИГ типа (2,1) означает, что ИГ, который будет преобразовывать автомат, выглядит следующим образом. Граф будет представлять из себя цепочку ребер или круг, как изображено на рисунке 3.2. Теорема утверждает, что нет автомата A, множества типов T и базового множества преобразований, использующих селекторные функции такого, что удается решать логическую ДЗПИО, используя такой граф над множеством предикатов $F(T)$.

Данная теорема справедлива для логической ДЗПИО. Следовательно, она справедлива для ДЗПИО. ИГ не обязан содержать сами записи, так как рассматривается логическая ДЗПИО. Далее под записью z в доказательстве будем понимать либо саму запись z, либо предикат $f_z^t, t \in T$. В доказательстве не будет существенен факт, что ИГ содержит саму запись z. Достаточно, чтобы он содержал ребро f_z^t. Например, фразу "найти запись z" можно понимать как "найти предикатное ребро $f_z^t, t \in T$, которое принимает значение один".

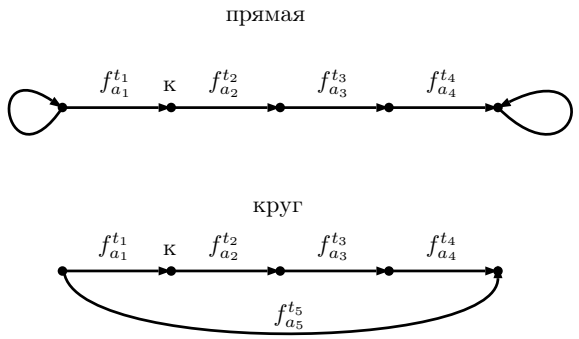


Рис. 3.2: Примеры ИГ, которые может построить ПДИГ типа (2,1).

Идея доказательства будет следующей. Для любого фиксированного алгоритма автомата, базового множества $F(T)$ и функции L можно привести поток запросов H, на котором не будет выполнено свойство, что количество тактов на поисковый запрос не превосходит $L(|V(i,H)|)$, где i — такт поступления поискового запроса, $|V(i,H)|$ — объем базы данных в такт i.

Искомый поток H будет устроен следующим образом. Сначала будут идти запросы на вставку $1, 2, \ldots, M$, причем $M > 4 * 4^{L(3)} + 5$. Далее поток будет содержать удаление всей базы данных кроме трех записей в определенном порядке. Следующим запросом будет поиск записи, от корня до которой расстояние не меньше $[M/2] - 1$. Доказательство существования такой записи является основным в данной теореме.

Также будет доказано, что если расстояние от корня до записи есть K ребер и K достаточно большое число, то минимальное количество тактов, которое автомат должен затратить на поиск этой записи будет не меньше $\log_2 \sqrt{\frac{K}{2}}$. После математических выкладок получим, что автомат, обслуживающий запрос на поиск записи, которая находится на расстоянии не меньшем $[M/2] - 1$ от корня, должен будет за-

тратить больше $L(3)$ тактов, а по условии должен был затратить не больше $L(3)$ тактов, что и докажет теорему.

Итак, приступим к доказательству. Сначала будут доказаны вспомогательные утверждения, а затем будет приведен поток запросов, описанный выше.

Докажем следующую лемму.

Лемма 9. *В условиях теоремы 8, если база данных V содержит M, $M \geq 3$ записей ($|V| = M$), то существует запись из V, путь до которой от корня содержит не меньше $[M/2] - 1$ ребер, где через $[x]$ обозначена целая часть снизу (наибольшее целое число не превосходящее x).*

Доказательство. Заметим, что ИГ должен представлять из себя связный граф, так как автомат начинает свое функционирование в корне и перемещается только по ребрам, которые он видит из текущей вершины. Поэтому, если ИГ содержит компоненты связности, которые не содержат корня, то автомат в них не сможет зайти. Следовательно, все компоненты связности, которые не включают в себя корень, можно не рассматривать. Поэтому далее будем считать, что ИГ содержит только одну компоненту связности.

Очевидно, что ИГ должен содержать не меньше $M-1$ ребра, так как существуют M различных вершин, которые должны образовывать связный граф. Кроме этого, ИГ представляет из себя либо цепочку, либо круг, как изображено на рисунке 3.2.

Рассмотрим случай цепочки. Пусть слева от корня находиться l записей, а справа r записей, где $l, r \geq 0$ и $l + r = M$. По принципу Дирихле либо l, либо r не меньше чем $[M/2]$. Без ограничения общности, пусть $r \geq [M/2]$. Выберем самую дальнюю справа от корня запись. Расстояние до нее будет не меньше $[M/2] - 1$ ребра, так как граф на $[M/2]$ вершинах должен содержать не меньше, чем $[M/2] - 1$ ребро.

В случае, когда ИГ представляет из себя круг, можно свести к случаю цепочки. Для этого удалим в круге 1 ребро и получим случай цепочки, для которой уже доказано. □

Докажем теперь лемму о минимальном количестве тактов, необходимых автомату, чтобы дойти до вершины, находящейся на расстоянии в K ребер от текущей вершины автомата. Заметим только, что основная цель следующей леммы состоит в том, чтобы показать, что количество тактов будет не константной функцией, а функцией от K. Поэтому для простоты оценка в лемме будет достаточно грубой. Также заметим, что нет смысла рассматривать маленькие значения K, так как они будут большими при доказательстве теоремы 8.

Лемма 10. *В условиях теоремы 8 автомат на поиск записи, находящейся от корня на расстоянии в $K, K \geq 100$ ребер, затратит не меньше $]\log_2 \sqrt{\frac{K}{2}}[$ тактов.*

Доказательство. Лемма утверждает, что автомату на поиск записи, находящейся на расстоянии в K ребер от корня, потребуется не меньше $]\log_2\sqrt{\frac{K}{2}}[$ тактов, чтобы ее найти.

Рассмотрим тривиальный случай, когда на ИГ нет ни одного автомата. Действительно, если на ИГ в момент поступления запроса на поиск не было ни одного другого автомата, то очевидно, что автомату на поиск потребовалось не меньше $K-1 >]\log_2\sqrt{\frac{K}{2}}[$ тактов, чтобы найти запись, так как расстояние уменьшиться не могло, а автомат, в силу радиуса видимости 1, может за 1 такт пройти только 1 ребро.

Пусть на ИГ функционировали другие автоматы во время поступления рассматриваемого автомата. Рассмотрим, каким образом автоматы с радиусом видимости 1 могли сократить расстояние, чтобы автомат на поиск мог дойти до искомой записи за меньшее количество тактов.

Первый способ – это поменять искомую запись с записью, которая на одно ребро ближе к корню. В этом случае расстояние до корня уменьшиться ровно на 1 ребро. Поменять с записью, которая ближе к корню больше, чем на 1 ребро, автомат не может в силу радиуса видимости 1.

Если какой-либо автомат удалит два ребра и не создаст новое, то в этом случае непрерывного пути к искомой записи либо не останется тогда, когда ИГ – цепочка, либо останется путь, который не изменится в случае круга. В случае круга такое преобразование возможно только один раз, так как после этого преобразования круг превратится в цепочку. А цепочка может превратиться в круг только тогда, когда ее длина будет не больше двух ребер, так как автомат должен соединить крайние вершины цепочки, что возможно только в случае, когда ребер не больше двух. В любом случае данное преобразование не сокращает расстояние до искомой записи, а, следовательно, не поможет рассматриваемому автомату найти ее за меньшее количество тактов.

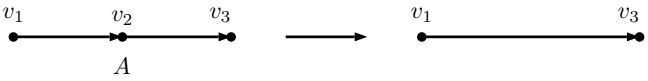


Рис. 3.3: Пример сокращения расстояния.

Рассмотрим последнее преобразование, которое может сократить расстояние от корня до искомой записи. Это преобразование изображено на рисунке 3.3. Данное преобразование заменят два ребра на одно. После его применения расстояние от корня к искомой записи сократиться на одно ребро.

Заметим, что других преобразований, сокращающих расстояние, нет, так как ПДИГ типа (2,1), а, следовательно, после преобразования ИГ должен удовлетворять свойству, что степень инцидентности любой вершины не больше двух. Поэтому

один автомат не может сократить расстояние больше, чем на одно ребро, так как радиус видимости у всех автоматов равен единице.

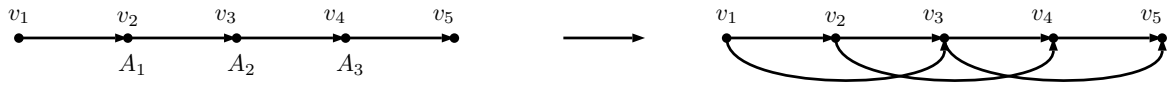


Рис. 3.4: Пример сокращения расстояния несколькими автоматами. В правой части нарисованы все возможные ребра. Видно, что расстояние в 4 ребра из v_1 в v_5 нельзя будет преодолеть быстрее чем за 2 такта.

Итак, рассмотрим преобразование, изображенное на рисунке 3.3. Посмотрим, насколько может сократиться расстояние, если несколько автоматов одновременно применят это преобразование. На рисунке 3.4 изображен пример, когда расстояние от вершины v_1 до вершины v_5 автоматы сокращают с помощью рассматриваемого преобразования. Видно, что расстояние в четыре ребра может сократиться максимум до двух ребер. Не трудно заметить, что если расстояние между вершинами будет p ребер, то расстояние не может стать не меньше $[p/2]$ ребер после применения преобразования несколькими автоматами, изображенного на рисунке 3.3.

Итак, за один такт расстояние от корня до искомой записи может сократиться максимально на $[K/2]+1$ ребро, так как возможно применение преобразования, меняющее записи местами, которое сократит расстояние еще на одно ребро.

Доказан следующий факт. Рассматривается расстояние от автомата до записи, которую он ищет. Пусть расстояние от вершины, в которой находится этот автомат, до искомой записи равно x. Тогда в следующий такт расстояние от автомата до искомой записи не меньше, чем $[x/2]+2$. Действительно, плюс один к оценке выше мог добавить сам автомат, перейдя в вершину по направлению к искомой записи.

Для доказательства леммы 10 осталось провести вычислительные выкладки. Будут проведены достаточно грубые оценки, так как главное в данной лемме – количество тактов зависит от K и не является константой.

Очевидно, что автомат дойдет до записи, когда расстояние до нее будет равно 0. В первый такт функционирования расстояние было не меньше K. Как было доказано выше, во второй такт функционирования, расстояние не меньше $K-([K/2]+2) \geq [K/2]-2$. В третий такт – не меньше $[K/2]-2-(([K/2]-2)/2+2) \geq [K/4]-6$. Пусть для i-го такта верно, что расстояние не меньше $[K/2^{i-1}]-2^{i-1}-2$. База индукции доказана для $i=2,3$. Докажем, что это верно для $i+1$. Для $i+1$ такта расстояние не может быть меньше $[K/2^{i-1}]-2^{i-1}-(([K/2^{i-1}]-2^{i-1})/2+2) \geq [K/2^i]-2^i-2$.

Найдем большое i, $i>1$, при котором $[K/2^i]-2^i-2>0$. Умножим для простоты и левую и правую часть неравенства на 2^i. Получаем, $K>2^{2i}+2^{i+1}$. Если $i<\log_2\sqrt{\frac{K}{2}}$, то

$2^{2i} + 2^{i+1} < 2^{2\log_2 \sqrt{\frac{K}{2}}} + 2^{\log_2 \sqrt{\frac{K}{2}}+1} < K/2 + 2\sqrt{K} < K.$

Следовательно, автомат затратит не менее $\log_2 \sqrt{\frac{K}{2}}$ тактов на поиск записи, что и требовалось доказать. □

Докажем теперь основную лемму, необходимую для доказательства теоремы 8.

Лемма 11 (Основная)**.** *В условиях теоремы 8 пусть база данных состоит из M записей. Для любого конечного ПДИГ типа (2,1) существует такой поток запросов H, что есть такт i, на котором мощность базы данных будет равна 3 ($|V(i, H) = 3|$), и в этот такт существует предикат f_z^t, $z \notin \{H(1), \ldots, H(i)\}$, расстояние от корня до которого не меньше $[M/2] - 1$ ребер.*

Доказательство. Другими словами лемма 11 утверждает, что для любого ПДИГ из условия теоремы 8 можно подобрать такой поток запросов H, что через некоторое количество тактов мощность базы данных будет равна 3, и будет существовать запись, расстояние от корня до которой будет не меньше $[M/2] - 1$, удаление которой не было в H.

Важно в данной лемме, что рассматривается конечный и селекторный ПДИГ. Так как ПДИГ конечный, то очевидно, можно найти такой такт, что ни один автомат не будет функционировать. Зафиксируем этот такт и будем посылать искомую последовательность, запросов на удаление начиная с него. Главный смысл состоит в том, что автомат на удаление не может удалить, чужую запись, так как в этом случае ее нельзя будет восстановить.

Итак, приступим к доказательству. Прежде всего, поймем, когда запись точно не может быть удалена. Всем ребрам ИГ приписаны предикаты из множества $F(T)$. Рассмотрим произвольный автомат, обслуживающий запрос z. Очевидно , что перемещаясь по ИГ, он будет видеть значение 1, только на тех ребрах, которым сопоставлены предикаты вида $f_z^t, t \in T$. На всех остальных ребрах он будет видеть значение 0.

Допустим автомат может удалить произвольную вершину. Покажем, что при этом ПДИГ не будет корректно функционировать.

Рассмотрим поток запросов. В первый такт идет удаление z, а в последующих идет поиск $M - 1$ записи всех элементов, кроме z. Если автомат, обслуживающий z, удалит вершину с записью не равной z, то автомат на поиск записи, соответствующей данной вершине, завершится неправильно, следовательно, ПДИГ будет работать некорректно. Действительно, как было показано выше, ни одного автомата, содержащего удаленную чужую запись, не функционирует (конечный ПДИГ). Поэтому, после удаления чужой записи, она никак не сможет восстановиться, откуда получаем, что ПДИГ будет работать некорректно.

Итак, показано, что если не было автомата на удаление какой либо записи, то вершина, содержащую эту запись, не может быть удалена.

Рассмотрим следующий, не трудный, но важный факт. Автомат может понять, есть ли искомая запись в какой-либо вершине, только в том случае, если он увидит предикат равный 1. Если во время своего функционирования он видит предикаты только равные 0, то он не знает есть ли искомая запись в базе данных или нет.

Рассмотрим последовательность запросов на удаление, в которой нет повторяющихся запросов (удаляются различные элементы). Для такой последовательности запросов, если автомат видит, во время своего функционирования, только предикаты равные 0, то запись, которую он удаляет не может быть удалена. Действительно, запись может быть удалена только в том случае, если хоть один из автоматов увидит предикат равный 1, соответствующей этой записи (предикат с параметром равной записи). Кроме автомата, удаляющего данную запись, никто не может увидеть значение равное 1 на таких предикатах, так как другие автоматы удаляют другие записи.

Из предыдущих выкладок следует, что если в потоке запросов все автоматы на удаление, во время своего функционирования будут видеть предикаты равные только 0, то ни одна запись не будет удалена. Покажем, что такую последовательность запросов можно привести.

Идея построения такой последовательности состоит в следующем. Каждый такт, кроме первого, своего функционирования автомат может увидеть не более одного нового ребра. В первый такт автомат может увидеть два ребра. Это следствие того, что радиус видимости автомата равен 1, а любой вершине в ИГ инцидентно не более двух ребер (ПДИГ типа (2,1)). Заметим, что новый предикат, который увидит автомат, будет равен 1 только на одном запросе. Поэтому, чтобы автомат увидел значение 0, достаточно не использовать этот запрос.

Для простоты объяснения, не ограничивая общности рассуждения, будем считать, что ни одного автомата не функционирует, и что поток запросов H, начиная с первого такта, будет содержать запросы на удаление, а не пустые запросы, чтобы дождаться, пока ни один автомат не будет функционировать.

Последовательность запросов на удаление состоит из $M-3$ запроса. Каждому автомату сопоставим множество запросов $B^i, 1 \leq i \leq M-3$, где i номер автомата в искомой последовательности удалений. В нулевой такт это множество состоит их всех элементов базы данных. Обозначать номер такта будем индексом снизу, то есть для всех $i, 1 \leq i \leq M-3 : |B_0^i| \geq M$.

Рассмотрим первый такт. В первый такт начинает функционировать только первый автомат. Пусть он увидит два предиката f_{a1}^{t1}, f_{a2}^{t2}. Он должен увидеть значение 0 на обоих предикатах, чтобы это гарантировать, из множества B_0^1 уберем два элемента a_1 и a_2. Получим множество элементов $B_1^1 = B_0^1 \setminus \{a_1, a_2\}$, которые может удалять первый автомат, так, чтобы во время функционирования видеть только нулевые значения предикатов.

Так как другие автоматы из последовательности еще не начали свое функционирование, то для всех $i, 2 \leq i \leq M-3 : B_0^i = B_1^i$. Следовательно в первый такт

$|B_1^1| \geq M-2, 2 \leq i \leq M-3 : |B_1^i| \geq M$.

Рассмотрим второй такт. Как было доказано выше первый автомат может увидеть не более одного нового ребра. Пусть предикат, который ему соответствует f_a^t. Важно, что на ребре, которое он уже видел не может появиться новый аргумент, так как используются только селекторные преобразования. Как и на первом такте, уберем этот элемент из B_1^1, получим что $B_2^1 = B_1^1 \setminus \{a\}$, $|B_1^2| \geq M-3$. Второй автомат во второй такт, аналогично первому автомату в первый такт, может увидеть не более двух предикатов, поэтому $|B_2^2| \geq M-2$. Мощность множества оставшихся автоматов не изменится, так как они не начали еще свое функционирование.

Итак, рассмотрим $M-3$ такт. Каждый такт, начиная со второго, мощность множества B^1 могла уменьшаться не более, чем на 1. То есть

$$|B_{M-3}^1| \geq |B_{M-2}^1| - 1 \geq |B_{M-1}^1| - 2 \geq \cdots \geq |B_2^1| - (M-5) \geq M-3-(M-5) = 2.$$

Отсюда следует, что $|B_{M-3}^2| \geq 3$, $|B_{M-3}^3| \geq 4, \ldots, |B_{M-3}^{M-3}| \geq M-2$.

Еще раз напомним смысл данных множеств. Множество B_{M-3}^1 состоит не менее чем из двух элементов. Если первый автомат удалял любую запись из множества B_{M-3}^1, то он видел бы, во время своего функционирования, только нулевые значения предикатов. Аналогично для остальных множеств.

Пусть первый запрос на удаление был $z_1 \in B_{M-3}^1$, то есть $H(1) = (\text{У}, z_1)$. Автомат, который его обслуживает до $M-3$ такта, обязательно видит нулевые предикаты. Второй запрос на удаление z_2, выбираем из множества $B_{M-3}^2 \setminus \{z_1\}$, мощность которого не меньше двух, так как $|B_{M-3}^2| \geq 3$. Аналогично для остальных тактов $j, j < M-3$, $H(j) = (\text{У}, z_j)$, где $z_j \in B_{M-3}^j \setminus \{z_1, z_2, \ldots, z_{j-1}\}$, $|B_{M-3}^j \setminus \{z_1, z_2, \ldots, z_{j-1}\}| \geq j+1-(j-1) \geq 2$. Итак, для потока запросов H, в $M-2$ такт не была удалена ни одна запись, так как все автоматы на удаление видели во время своего функционирования нулевые предикаты. По лемме 9 существует запись, расстояние от корня до которой не меньше $[M/2]-1$ ребра. Пусть это запись z. Заметим, что была свобода выбора из двух записей для каждого автомата. Например, если $z_1 = z$, то в качестве $H(1)$ выберем $z_1^{'} \in B_{M-3}^1 \setminus \{z\}$. Если $z_2 = z$, то $H(2)$ будет равен $z_2^{'} \in B_{M-3}^2 \setminus \{z_1, z\}$, $|B_{M-3}^2 \setminus \{z_1, z\}| \geq 1$. Аналогично для остальных тактов $j, j < M-3, z_j^{'} \in B_{M-3}^j \setminus \{z_1, z_2, \ldots, z_{j-1}, z\}$, $|B_{M-3}^j \setminus \{z_1, z_2, \ldots, z_{j-1}, z\}| \geq j+1-j \geq 1$.

Построенный поток H удовлетворяет условию леммы 11, так как в такт объем базы данных равен $|V(M-3), H| = 3$ и существует запись $z, (\text{У}, z) \notin H$, расстояние от корня до которой не меньше $[M/2]-1$ ребра. □

Вернемся к доказательству теоремы 8. Зафиксируем произвольную функцию L и произвольный конечный ПДИГ типа (2,1). Для доказательства нужно предъявить поток запросов H, в котором существует такт i и поисковый запрос, что время на его обработку будет больше, чем $L(|V(i, H)|)$.

Как уже было сказано в идее доказательства, сначала в потоке запросов H будут идти запросы на вставку. $H(1) = (\text{В},1), \ldots, H(M) = (\text{В},\text{М})$, где $M > 4 * 4^{L(3)} +$

5. Далее ставим столько пустых запросов, чтобы на графе не осталось ни одного работающего автомата. Номер такта, когда это случиться, обозначим M'. Применяя лемму 10 можно предъявить поток запросов $H^{'}$, такой, что будет существовать такт t, для которого мощность базы данных будет равна 3, и будет существовать запись z, расстояние от корня до которой будет не меньше $[M/2]-1$, удаление которой не было в $H^{'}$.

Пусть $H(M'+1)=H^{'}(1),\ldots,H(M'+t)=H^{'}(t),H(M'+t+1)=$ (П,z). Покажем, что время обработки запроса $H(M'+t+1)$ больше, чем $L(|V(M'+t+1,H)|)$, а тем самым докажем теорему.

Действительно, по лемме 10, $|V(M'+t+1,H)|=3$, расстояние от корня до z не меньше $[M/2]-1>2*4^{L(3)}+2$. Будем считать, что $M>202$, чтобы выполнилось условие леммы 10. По лемме 10 автомат затратит на поиск z не меньше, чем $\log_2\sqrt{\frac{[M/2]-1}{2}}>\log_2\sqrt{4^{L(3)}}=L(3)$, а должен был затратить не больше $L(|V(M'+t+1,H)|=L(3)$ тактов, противоречие.

Теорема 8 доказана.

□

Заключение

В диссертации получены следующие основные результаты.

Разработана математическая модель динамических баз данных. Доказана применимость предлагаемой модели для решения задач, возникающих в динамических базах данных. На основе этой модели получены бесконечно распараллеливаемые структуры данных. Получены различные верхние и нижние оценки для решения ДЗПИО и логической ДЗПИО для любого потока запросов.

Построен конечный, селекторный ПДИГ типа (8,4), решающий ДЗПИО для любого потока запросов с логарифмической сложностью.

Доказано, что существует конечный, селекторный ПДИГ типа (2,2), решающий ДЗПИО для любого потока запросов. Также доказано, что для любого натурального R не существует конечного ПДИГ типа $(1, R)$, который решает ДЗПИО для любого потока запросов, но существует конечный, не селекторный ПДИГ типа (1,1), решающий логическую ДЗПИО.

Показано, что не существует конечного, селекторного ПДИГ типа (2,1), решающего ДЗПИО для любого потока запросов, но существует бесконечный ПДИГ типа (1,1), решающий эту же задачу.

Предъявлен минимально возможный по степени ветвления конечный, не селекторный ПДИГ с радиусом видимости один, решающий ДЗПИО для любого потока запросов.

В качестве перспектив дальнейшей разработки темы диссертации можно предложить применять полученную модель динамических баз данных для решения других динамических задач информационного поиска, например, для решения динамической задачи двумерного интервального поиска.

Литература

[1] Гасанов Э.Э., Кудрявцев В. Б. Теория хранения и поиска информации // М.: ФИЗМАТЛИТ, 2002.

[2] Кудрявцев В.Б., Алешин С.В., Подколзин А.С. Введение в теорию автоматов // М.: Наука, 1985.

[3] Codd E. F. A Relation Model of Data for Large Shared Data Banks // Comm. ACM 13, № 6, ACM, New York, London, Amsterdam, June 1970, 377–387.

[4] Решетников В. Н. Алгебраическая теория информационного поиска // Программирование. — 1979. — № 3. — С. 68–74.

[5] Dumey A. Indexing for Rapid Random Access Memory Systems // Computers and Automation. (1956) 4, № 12, 6–9.

[6] Ершов А. П. О программировании арифметических операторов // ДАН СССР. — 1958. — Т. 118. — С. 427–430.

[7] Bentley J. L., Friedman J. H., Data structures for range searching // Comput. Surveys (1979), 11 397–409.

[8] Кнут Д., Искусство программирования для ЭВМ. Т. 3. Сортировка и поиск. — М.: Мир, 1978.

[9] Ульман Дж. Основы систем баз данных,пер. с англ. — М.: Мир, 1983.

[10] Hagerup T., Kammer F. Succinct Choice Dictionaries // CoRR abs/1604.06058 (2016).

[11] Ruzic M. Uniform deterministic dictionaries // ACM Trans. Algorithms 4 (2008).

[12] Препарата Ф., Шеймос М. Вычислительная геометрия: Введение, М.: Мир, 1989.

[13] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford Introduction to Algorithms (3rd ed.) (2009) [1990]. MIT Press and McGraw-Hill.

[14] Donald E. Knuth Sorting and Searching, volume 3 of The Art of Computer Programming // Addison-Wesley, 1973.

[15] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ulman The Design and Analysis of Computer Algorithms // Addison-Wesley, 1974.

[16] Comer D. The Ubiquitous B-tree // ACM Computing Surveys, 11(2):121-137, 1979.

[17] Leo J. Guibas, Robert Sedgewick A Dichromatic Framework for Balanced Trees // In Proceedings of the 19th Annual Symposium on Foundations of Computer Science, pages 8-21. IEEE Computer Society, 1978.

[18] E.A. Arjomandi A Study of Parallelism in Graph Theory // PhD thesis, PhD thesis, Dept. Computer Science, University of Toronto, 1975.

[19] F.Y. Chin, J. Lam, and I. Chen Efficient parallel algorithms for some graph problems // Comm. ACM, 25(9):659–665, 1982.

[20] O. Berkman , B. Schieber, and U. Vishkin Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values // Journal of Algorithms, 14(3):344–370, 1993.

[21] Park H., Park K. Parallel algorithms for red-black trees // Theoretical Computer Science 262 (2001) 415-435.

[22] Wang B-F, Chen G-H Cost-optimal parallel algorithms for constructing 2-3 trees // Journal Of Parallel And Distributed Computing 11 (1991) 257-261.

[23] Gafni E., Naor J., Ragde P. On Separating the EREW and CREW PRAM Models // Theoretical Computer Science 68 (1989) 343-346.

[24] Snir M. On parallel searching // SIAM J. Computing 14 (1985) 688-708.

Публикации автора по теме диссертации

[25] Плетнев А.А. Моделирование динамических баз данных// Интеллектуальные системы. — 2014. Т. 17, Вып. 1—4. — С. 75-79.

[26] Плетнев А.А. Информационно-графовая модель динамических баз данных и ее применение// Интеллектуальные системы. Теория и приложения. — 2014. Т. 18, Вып. 1. — С. 111-140.

[27] Плетнев А.А. Динамическая база данных, допускающая параллельную обработку произвольных потоков запросов// Интеллектуальные системы. Теория и приложения. — 2015. Т. 19, Вып. 1. — С. 117–142.

[28] Плетнев А.А. Логарифмическая по сложности параллельная обработка автоматами произвольных потоков запросов в динамической базе данных// Интеллектуальные системы. Теория и приложения. — 2015. Т. 19, Вып. 1. — С. 171–212.

[29] Плетнев А.А. Нижняя оценка на область видимости автомата, обрабатывающего произвольный поток запросов к динамической базе данных// Интеллектуальные системы. Теория и приложения. — 2015. Т. 19, Вып. 4. — С. 117-151.

[30] Плетнев А. А. Минимально возможный по степени ветвления информационный граф с радиусом видимости один, обрабатывающий произвольный поток запросов к динамической базе данных // Интеллектуальные системы. Теория и приложения.Т. 20, Вып. 1, 2016, — С. 223-254.

[31] Плетнев А.А. Моделирование динамических баз данных // Материалы X Международной конференции "Интеллектуальные системы и компьютерные науки"(5-10 декабря 2011 года). М.: 2011. С. 72-75.

[32] Плетнев А.А. Решение динамической задачи поиска идентичных объектов // Материалы XI Международного семинара "Дискретная математика и ее приложения"(Москва, 18-23 июня 2012 г.). М.: 2012. С. 366-368.

[33] Плетнев А.А. Динамическая база данных, допускающая параллельную обработку произвольных потоков запросов с логарифмической сложностью// Интеллектуальные системы и компьютерные науки. — 2015.

[34] Плетнев А.А. О сложности параллельного решения динамической задачи поиска идентичных объектов // Тезисы докладов Республиканской научной конференции с участием зарубежных ученых "Современные методы математической физики и их приложения Ташкент, 15-17.04.2015. — Т. 1. — Ташкент, 2015. — С. 167—168.