Сотня из автобусного билета

В. В. БОРИСЕНКО
Московский государственный университет им. М. В. Ломоносова
e-mail: vladimir_borisen@mail.ru

Р. РАХМАТОВ
Филиал МГУ им. М. В. Ломоносова в городе Душанбе, Таджикистан
e-mail: lotrapa97@gmail.com

УДК 004.02

Ключевые слова: полные бинарные деревья, число Каталана.

Аннотация

Рассматривается шуточная задача получения числа 100 из шестизначного номера автобусного билета путём расстановки скобок и знаков арифметических операций между некоторыми цифрами номера. Алгоритм решения для фиксированного номера сводится к перебору всех полных бинарных деревьев, имеющих ровно 6 терминальных вершин, с расставленными знаками операций в нетерминальных вершинах. Для шестизначных номеров, не содержащих повторяющихся цифр и цифры 0, задача имеет решение для всех номеров, кроме единственного номера 746189. Получить сотню из номера билета 746189 невозможно.

Abstract

V. V. Borisenko, R. Rakhmatov, A hundred from a bus ticket, Fundamentalnaya i prikladnaya matematika, vol. 23 (2020), no. 3, pp. 37—47.

We consider the comic task of obtaining the number 100 from a six-digit bus ticket number by placing parentheses and symbols for arithmetic operations between some digits. The solution algorithm for a fixed number is reduced to enumerating all binary trees having exactly six terminal nodes with the signs of operations in internal nodes. For six-digit numbers that do not contain duplicate digits and the digit 0, the problem has a solution for all numbers except the only number 746189. It is impossible to obtain a hundred from this strange number 746189.

Сейчас, в пору повального увлечения смартфонами, большинство пассажиров во время поездок в транспорте развлекается, уткнувшись в экраны. Но когда смартфонов ещё не было, мы коротали время в автобусе, пытаясь составить из цифр номера автобусного билета формулу, значение которой равнялось бы 100. Переставлять цифры номера было нельзя, зато можно было расставлять скобки и знаки арифметических операций между некоторыми цифрами билета. Если между цифрами не было никакого знака, то они объединялись в двузначное или трёхзначное число. Как вариант допускалось использование унарного

Фундаментальная и прикладная математика, 2020, том 23, № 3, с. 37—47.

Рис. 1. Билет на трамвай. Смоленск, 2018 г.

минуса (т. е. операции изменения знака числа). Решение без унарного минуса считалось более красивым.

На рис. 1 изображён трамвайный билет с номером 257079. Получить сотню из него можно многими разными способами, не используя или используя унарный минус, например,

$$2+5\cdot 7+0+7\cdot 9=100, \qquad 2\cdot 5+70/(7/9)=100,$$
$$\big((-2)+5\big)\cdot 7+0+79=100, \qquad 2\cdot(-5)\cdot(7/(0-7)-9)=100.$$

Всего существует 14 решений без унарного минуса и 104 решения с унарным минусом.

Из номера 123456 сотню можно получить только двумя способами, не используя унарный минус:

$$1+(2+3+4)\cdot(5+6)=100, \qquad (1+2/3)\cdot(4+56)=100.$$

Отметим, что второй способ был найден компьютером, из людей никто до него не додумался. Используя унарный минус, из номера 123456 сотню можно получить 65 способами, вот некоторые из них:

$$(-1)\cdot 2+(3\cdot 4+5)\cdot 6=100, \qquad \big((-1)-2/3\big)\cdot((-4)-56)=100,$$
$$(-1)\cdot\big(2-3\cdot(4+5\cdot 6)\big)=100, \qquad (-1)\cdot 2-(3-4\cdot 5)\cdot 6=100,$$
$$1-(2+3+4)\cdot((-5)-6)=100, \qquad 1\cdot\big((-2)/3+4\big)\cdot 5\cdot 6=100.$$

Конечно, не из всякого номера можно получить сотню; очевидно, например, что это невозможно сделать для номера 000123 и других подобных. Тем не менее из номеров, не начинающихся с нуля и содержащих разнообразные цифры (т. е. более или менее случайных), сотня обычно получалась. К тому времени относится следующее предположение.

Гипотеза. Если номер начинается не с нуля и не содержит двух одинаковых цифр, то сотню сделать можно.

Данная работа посвящена проверке этого утверждения с помощью компьютера. Нужно перебрать все номера, удовлетворяющие условию гипотезы, и для

каждого номера перебрать всевозможные формулы, включающие четыре арифметических действия и операцию изменения знака (унарный минус).

Забегая вперёд, сразу скажем, что эта гипотеза оказалась неверной. Было найдено довольно много подобных номеров, из которых сотню сделать невозможно: 103459, 104759, 106597, 107459, 109543, ..., 980476, 980527, 980567, 984370, 984560 — всего 2240 номеров. Правда, число номеров, удовлетворяющих условию гипотезы, намного больше: оно равно 131250, так что вероятность найти такой «плохой» номер среди случайного набора невелика, всего лишь $2240/131250 \approx 0{,}017$.

Почти все подобные «плохие» номера содержат цифру ноль, поэтому был соблазн переформулировать гипотезу, совсем исключив 0 из цифр номера. Но, как оказалось, существует и номер, не содержащий нуля и повторяющихся цифр, из которого сотню сделать невозможно. Самое удивительное, что такой номер всего один! Это номер 746189 — экстремально плохой, нарушающий все закономерности. Причём совершенно невозможно понять причину, почему это так. Из соседнего номера 746188 сотню можно сделать 11 способами, например,

$$7 + (4 + 61/8) \times 8 = 100;$$

из номера 746190 — 26 способами, например,

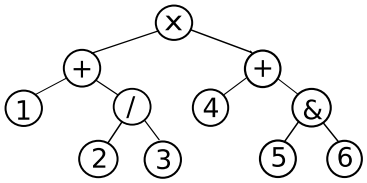
$$7 - 4 + 6 + 1 + 90 = 100.$$

Из позитивных утверждений отметим, что гипотеза верна для номеров длины 7 (а также для номеров большей длины вплоть до 10). Впрочем, это неудивительно и уже не так интересно: чем длиннее номер, тем больше разных вариантов составления формул и больше возможностей получить число 100.

Ещё одной целью при написании компьютерной программы, решающей задачу получения сотни из номера билета, была проверка возможностей компьютера в соревновании с человеком. Как и в большинстве интеллектуальных игр, здесь компьютер выигрывает у человека «в одни ворота». Для случайного номера компьютер находит решение в среднем быстрее, чем за 0,02 секунды, причём зачастую компьютер находит решения, до которых человек никогда не додумается.

Представление формул в виде бинарных деревьев

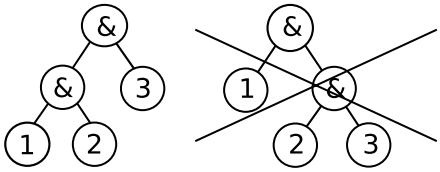
Сведём нашу задачу к рассмотрению полных бинарных деревьев. Бинарное дерево называется *полным*, если у каждого узла количество потомков равно либо 0 (такой узел называется *терминальным* узлом или *листом*), либо 2. Подобные деревья задают порядок расстановки скобок в арифметической формуле без унарного минуса. Терминальным узлам соответствуют цифры, входящие в формулу, *внутренним* (т. е. нетерминальным) узлам — операции. Например, формуле $(1 + 2/3) \times (4 + 56)$ соответствует следующее полное бинарное дерево:



Здесь знаком $\&$ обозначена операция *конкатенации*, т. е. приписывания цифры 6 справа к числу 5, в результате которой получается двузначное число $5 \,\&\, 6 = 56$. Отметим, что если арифметические операции можно расставлять произвольно во внутренних вершинах фиксированного дерева, то на операцию конкатенации $\&$ накладываются следующие два ограничения. Пусть операция $\&$ исполняется в некотором внутреннем узле v. Тогда

— левое поддерево узла v либо является листом, либо в его корне также выполняется операция $\&$;
— правое поддерево узла v является листом.

Например, на рисунке ниже левое дерево является корректным, а правое нет.



Значение в узле, в котором выполняется операция $\&$, вычисляется по формуле

$$v = v_l \cdot 10 + v_r,$$

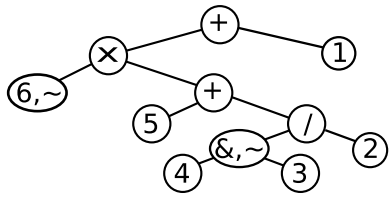
где v_l — значение в корне левого поддерева, v_r — значение в корне правого поддерева (оно должно быть листом).

Учёт унарных минусов

Операция изменения знака, т. е. унарный минус, возможна в любом узле дерева, представляющего арифметическую формулу. Если это лист, то мы просто изменяем знак цифры. Если узел внутренний, то сначала выполняется заданная в узле бинарная операция и только потом применяется унарный минус. При изображении дерева мы в каждом узле можем указывать одну или две операции, сначала бинарную операцию и затем унарный минус, если он применяется. Чтобы не путать унарный минус с бинарной операцией вычитания, условимся изображать его значком $\sim$ (тильда). Например, рассмотрим представление сотни для номера 654321, использующее унарные минусы:

$$(-6) \times (5 + (-43)/2) + 1 = 100.$$

Ему соответствует следующее дерево.



Потенциально унарный минус может стоять в любом узле дерева (за исключением узлов, у которых родительский узел содержит операцию конкатенации &). Поскольку в полном бинарном дереве с шестью листьями общее число узлов равно 11, получаем для фиксированного дерева с фиксированной расстановкой бинарных операций $2^{11} = 2048$ вариантов расстановки унарных минусов. Такое большое количество вариантов катастрофически замедляет перебор. К счастью, следующее утверждение позволяет существенно ограничить перебор.

Предложение 1. *Для дерева с фиксированной расстановкой бинарных операций достаточно перебрать только всевозможные расстановки унарных минусов в следующих узлах*:

- *в* листьях, *являющихся* левыми *сыновьями своих родителей, причём в родительском узле не должна выполняться операция конкатенации* &;
- *во* внутренних узлах с операцией конкатенации &, *являющихся* левыми *сыновьями своих родителей, причём такой узел должен быть корнем максимального поддерева, во внутренних узлах которого выполняются только операции конкатенации.*

Доказательство. Заметим, что для арифметических бинарных операций можно рассматривать лишь случаи, когда унарный минус применяется только к первому операнду, поскольку

$$(-x) + (-y) = (-x) - y, \quad (-x) - (-y) = (-x) + y,$$
$$(-x) \cdot (-y) = x \cdot y, \quad (-x)/(-y) = x/y.$$

Кроме того, следующие тождества позволяют переместить операции унарного минуса с внутренних узлов вниз по дереву вплоть до левых листьев либо левых внутренних вершин, содержащих операцию конкатенации:

$$-(x + y) = (-x) - y, \quad -(x - y) = (-x) + y,$$
$$-(x \cdot y) = (-x) \cdot y, \quad -(x/y) = (-x)/y.$$

Последняя часть предложения вытекает из того, что операция конкатенации неприменима к нижестоящим узлам, в которых выполнялась операция изменения знака. □

Из предложения 1 следует, что в дереве с шестью листьями при любой расстановке бинарных операций можно ограничиться не более чем пятью операциями унарного минуса, это сразу сокращает перебор с $2^{11} = 2048$ до $2^5 = 32$ вариантов. Причём 32 варианта встречаются очень редко, только для деревьев, соответствующих формулам вида $x_1 \circ \Big(x_2 \circ \Big(x_3 \circ \big(x_4 \circ (x_5 \circ x_6)\big)\Big)\Big)$, где символ $\circ$ обозначает любую из четырёх арифметических операций. (Здесь цифры $x_1, \ldots, x_5$ содержатся в левых листьях, операцию изменения знака можно применять только в них; цифра x_6 содержится в правом листе, поэтому унарный минус для неё не применяется.)

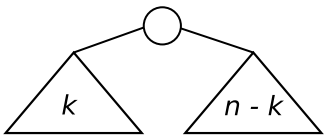
Генерация полных бинарных деревьев и число Каталана

Количество полных бинарных деревьев с n листьями равно *числу Каталана* C_{n-1}. Число Каталана C_n — одно из самых замечательных чисел в комбинаторике [1—4], оно появляется в самых разных контекстах, на первый взгляд не связанных между собой. Хорошо известна формула для числа Каталана:

$$C_n = \binom{2n}{n} / (n+1).$$

В нашем случае число полных бинарных деревьев с шестью листьями равно $C_5 = 42$. Перебор деревьев получается не очень большим даже для деревьев с семью листьями, их число равно 132.

Для генерации всех полных бинарных деревьев с n листьями воспользуемся следующим соображением. Если $n = 1$, то такое дерево единственно и состоит из одного листа. Если $n > 1$, то корень дерева имеет ровно двух сыновей. Пусть левое поддерево имеет k листьев, где $k < n$. Тогда правое поддерево имеет $n - k$ листьев. Таким образом, каждое дерево с n листьями получается соединением двух деревьев с k и $n - k$ листьями путём добавления ещё одной корневой вершины.



Алгоритм генерации деревьев легко реализуется рекурсивно. Для заданного целого числа $n > 1$ алгоритм заполняет массив всех деревьев с n листьями. При $n = 1$ такое дерево только одно и состоит из единственного узла. Для произвольного n задача сводится к той же задаче для меньших n. В цикле перебираются все числа k от 1 до $n-1$. При каждом k с помощью рекурсивного вызова алгоритма создаётся массив A деревьев с k листьями. Во вложенном цикле для каждого дерева $t_l \in A$ также путём рекурсивного вызова алгоритма

создаётся массив B деревьев с $n-k$ листьями. Деревья $t_r \in B$ перебираются в ещё одном вложенном цикле, в котором для каждой пары деревьев (t_l, t_r) создаётся дерево t с n листьями путём добавления новой корневой вершины, для которой деревья t_l и t_r будут левым и правым поддеревьями. Созданное таким образом дерево t добавляется в выходной массив.

Обозначим через $f(n)$ количество полных бинарных деревьев с n листьями. (Хорошо известно, что $f(n) = C_{n-1}$.) Из описанного алгоритма следует, что

$$f(n) = \sum_{k=1}^{n-1} f(k)f(n-k)$$

(эта формула является одним из эквивалентных определений числа Каталана). Из этой формулы, учитывая, что $f(1) = f(2) = 1$, получаем

$$\begin{aligned}
&f(3) = f(1)f(2) + f(2)f(1) = 2,\\
&f(4) = f(1)f(3) + f(2)f(2) + f(3)f(1) = 1\cdot 2 + 1\cdot 2 + 2\cdot 1 = 5,\\
&f(5) = f(1)f(4) + f(2)f(3) + f(3)f(2) + f(4)f(1) =\\
&\quad = 1\cdot 5 + 1\cdot 2 + 2\cdot 1 + 5\cdot 1 = 14,\\
&f(6) = f(1)f(5) + f(2)f(4) + f(3)f(3) + f(4)f(2) + f(5)f(1) =\\
&\quad = 1\cdot 14 + 1\cdot 5 + 2\cdot 2 + 5\cdot 1 + 14\cdot 1 = 42.
\end{aligned}$$

Оценка времени работы алгоритма

Число полных бинарных деревьев с шестью листьями невелико — всего лишь 42. Больший вклад во время работы алгоритма вносит перебор всех наборов операций. Учитывая операцию конкатенации, мы имеем пять возможных операций. Всего в полном дереве с шестью листьями 11 узлов, 6 терминальных и 5 внутренних, операции расставляются только во внутренних узлах, поэтому получаем $5^5 = 3125$ различных наборов операций. Правда, с учётом указанных выше ограничений на операцию конкатенации & не все наборы операций корректны для любого дерева. Для всех возможных расстановок операций и деревьев мы получаем 64 469 корректных деревьев (это число вычислено в ходе работы программы). Но всё равно для каждого дерева приходится рассматривать все наборы операций для определения их корректности, поэтому всего имеется $3125 \cdot 42 = 131\,250$ вариантов. Расстановки операций унарного минуса с учётом указанных выше ограничений увеличивают это число не более чем в $2^5 = 32$ раза (это лишь ограничение сверху, реально это число намного меньше). Поэтому получаем верхнюю оценку для числа вариантов при фиксированном наборе из шести цифр: не более $42 \cdot 5^5 \cdot 2^5 = 4\,200\,000$ вариантов.

Мы рассматриваем лишь наборы из шести цифр, удовлетворяющие условию сформулированной гипотезы: первая цифра должна быть отлична от 0 и каждая цифра может входить в набор не больше одного раза. Поэтому всего наборов $9 \cdot 9 \cdot 8 \cdot 7 \cdot 6 \cdot 5 = 136\,080$.

Отметим, что для любого фиксированного набора программа обычно очень быстро находит первое решение. При проверке сформулированной в начале статьи гипотезы перебор вариантов происходит только до первого решения, поэтому перебирать все 4 200 000 вариантов разных деревьев, наборов операций и расстановки унарных минусов чаще всего не приходится. На практике на не очень быстром компьютере программа тратит на проверку одного номера в среднем не больше двух сотых секунды. Больше времени тратится лишь на «плохие» номера, из которых сотню сделать невозможно, для них приходится перебирать все варианты. Но таких номеров, как оказалось, совсем немного. На проверку всех номеров длины 6, удовлетворяющих условию гипотезы, было потрачено меньше часа.

Более того, отметим, что примерно так же быстро программа проверила все номера длины 7. Таких номеров больше, но зато программа для номера длины 7 в среднем находит решение значительно быстрее, чем для номера длины 6. Объясняется это тем, что чем больше количество цифр, тем больше вариантов для получения сотни. Для длины 7 гипотеза оказалась верна: «плохих» номеров не существует, из любого номера из семи цифр, начинающегося не с нуля и не содержащего двух одинаковых цифр, сотню сделать можно. Более того, для номеров длины 7 можно даже не использовать унарный минус — сотню можно сделать лишь с помощью четырёх бинарных арифметических операций.

Программная реализация

Нами были реализованы две программы: `tick100` и `badNumbers`. Первая для фиксированного номера произвольной длины, который вводится человеком с клавиатуры, находит все решения задачи «сделать сотню из введённого номера», причём сначала она находит решения без унарного минуса и затем решения с унарным минусом. Последних всегда значительно больше, поэтому печатается лишь их ограниченное число (не больше 64). Примеры работы программы:

```
Ticket number: 123456
1+(2+3+4)*(5+6) = 100
(1+2/3)*(4+56) = 100
((-1)*(-2)/3-4)*(-5)*6 = 100
((-1)*2/3+4)*5*6 = 100
...
(-1)*2+(3*4+5)*6 = 100
(-1)*2+3*(4+5*6) = 100
(-1)*2-(3-4*5)*6 = 100
...
Solutions without unary minus: 2
Solutions with unary minus: 65
```

```
Ticket number: 777777
(7+7)*(7+7/(7*7)) = 100
(7+7)*(7+(7/7)/7) = 100
...
(777-77)/7 = 100
(-7)*((-7)-7)+(7+7)/7 = 100
...
Solutions without unary minus: 13
Solutions with unary minus: 37
```

Отметим, что в случае номера 777777 решение $(777 - 77)/7 = 100$ для человека вполне естественно, но решение $(7 + 7) * (7 + 7/(7 * 7)) = 100$ выглядит совершенно удивительным! Связано это с тем, что человек обычно пытается выразить сотню либо как сумму двух или трёх чисел, которые составляются из частей номера, либо как произведение двух чисел, причём рассматриваются лишь простые произведения, такие как $50 \cdot 2$, $25 \cdot 4$, $20 \cdot 5$, $10 \cdot 10$; иногда произведение можно подкорректировать, например, $100 = 17 \cdot 6 - 2$. Но для человека очень непривычно искать решение в виде $100 = 14 \cdot 7\frac{1}{7}$, как это делает компьютер в данном примере.

Ещё один пример демонстрирует, что бывают номера, для которых есть только решения с унарными минусами:

```
Ticket number: 102958
(-1)+(0-2)+95+8 = 100
(-1)+(0-2)-((-95)-8) = 100
(-1)+0+(-2)+95+8 = 100
...
Solutions without unary minus: 0
Solutions with unary minus: 77
```

Вторая программа `badNumbers` была написана для проверки гипотезы о том, что из любого номера длины 6, начинающегося не с нуля и не имеющего повторяющихся цифр, можно сделать сотню. Она ищет «плохие» номера, не удовлетворяющие этой гипотезе. Длина номера является параметром программы. Плохие номера, если они есть, записываются в файл `badNumbers6.txt` (или `badNumbersN.txt`, где `N` — длина номера, которая является входным параметром и задаётся в командной строке).

В другой файл `unMinusNumbers6.txt` (или `unMinusNumbersN.txt`, если длина номера `N` отлична от 6) записываются номера, для которых нет решений без использования унарного минуса; также в него записываются и плохие номера, вообще не имеющие решений (это отмечается в тексте). От программы `tick100` программа `badNumbers` отличается тем, что она находит только одно, самое первое решение для каждого номера, если оно есть. Благодаря этому программа работает достаточно быстро: все номера длины 6 перебираются за время

около часа. Вот фрагмент файла `unMinusNumbers6.txt\verb` с результатами работы программы, в котором перечислены номера, требующие унарного минуса, а также «плохие» номера, из которых сотню сделать невозможно.

```
102958
103459 cannot make 100 even with un. minus
104267
104759 cannot make 100 even with un. minus
105278
105439
...
980526
980527 cannot make 100 even with un. minus
980567 cannot make 100 even with un. minus
983470
984370 cannot make 100 even with un. minus
984560 cannot make 100 even with un. minus
985460
987240
```

Здесь для номеров 102958, 104267, 105278, 105439, 980526, 983470, 985460, 987240 есть только решения с унарными минусами, а для номеров 103459, 104759, 980527, 980567, 984370, 984560 решений вообще нет.

Все программы написаны на языке C++ с использованием стандартной библиотеки классов STL (используются классы `std::vector`, `std::string` и `std::set`). Для работы с деревьями реализованы классы `TreeNode` (узел дерева) и `Tree` (дерево). Для убыстрения работы с деревьями узлы дерева захватываются не в динамической памяти (потенциально медленный оператор new языка C++ не используется), а в пуле блоков (`class TreeNodePool`, который наследуется из класса `std::vector<TreeNode>`). Ссылки на левого и правого сына узла реализуются не как указатели (стандартное решение), а как индексы элементов в общем для всех деревьев массиве `nodePool`, содержащем элементы типа `TreeNode`.

Цифры проверяемого номера не записываются в узлы дерева. Вместо этого терминальные узлы хранят индексы элементов отдельного массива `digits`, в котором содержатся цифры проверяемого номера. Это позволяет быстро перебирать номера, меняя только содержимое этого массива и не изменяя узлов деревьев. Тот же приём используется и для задания бинарных операций во внутренних узлах деревьев: операции хранятся в отдельном массиве `operations`, узлы деревьев хранят индексы элементов этого массива. Аналогично организована и работа с унарными минусами. Большинство функций, работающих с деревьями, реализовано рекурсивно (генерация деревьев с заданным числом листьев, проверка корректности заданного набора операций, определение возможных позиций унарных минусов, вычисление значения формулы по её дереву и т. п.).

Исходные тексты программ доступны по адресу `http://mech.math.msu.su/~vvb/tick100.zip`.

Литература

[1] Ландо С. К. Лекции о производящих функциях. — М.: МЦНМО, 2007.

[2] Шень А. Программирование. Теоремы и задачи. — М.: МЦНМО, 2007.

[3] Davis T. Catalan Numbers. — http://www.geometer.org/mathcircles. — 2006.

[4] Črepinšek M., Mernik L. An efficient representation for solving Catalan number related problems // Int. J. Pure Appl. Math. — 2009. — Vol. 56, no. 4. — P. 589—604.