Серия «Математика»
2017. Т. 22. С. 3—17

Онлайн-доступ к журналу:
http://mathizv.isu.ru

ИЗВЕСТИЯ
Иркутского
государственного
университета

УДК 519.688
MSC 68P10
DOI https://doi.org/10.26516/1997-7670.2017.22.3

Использование полных последовательностей для сортировки натуральных чисел

Ю. Н. Артамонов
ФГБНУ «Госметодцентр»

Аннотация. Полные последовательности определяются как бесконечные последовательности натуральных чисел, с помощью которых можно представить любое другое натуральное число. Наиболее сильный результат, позволяющий судить о полноте любой последовательности, был получен Д. Брауном. В статье ставится задача представления в виде суммы элементов полной последовательности всех натуральных чисел до некоторого предела (такие начальные участки полных последовательностей названы порождающими последовательностями). Тогда возникает задача нахождения для заданного предела N порождающих последовательностей минимальной длины. В статье предложены алгоритмы генерации порождающих последовательностей минимальной длины. Предложен класс алгоритмов генерации порождающих последовательностей, содержащих в себе заданную порождающую последовательность меньшей длины, что позволяет вводить регулярные алгоритмы генерации полных последовательностей. Предложенные регулярные алгоритмы генерации полных последовательностей использованы при разработке алгоритма сортировки натуральных чисел без их сравнения, являющегося развитием алгоритма поразрядной сортировки с интерпретацией разрядов как элементов подходящей полной последовательности. В статье продемонстрированы подходы адаптации работы данного алгоритма для сортировки конкретной сортируемой последовательности.

Ключевые слова: полные последовательности, алгоритмы сортировки за линейное время, поразрядная сортировка, нетрадиционные системы счисления, алгоритмы поиска и хранения числовых данных.

1. Общая постановка задачи

Начнем рассмотрение со следующей задачи [1; 4; 5]:

«Имеется 2009 мешочков с 1, 2, 3,..., 2008 и 2009 монетами. Каждый день разрешается взять из одного или нескольких мешочков по одина-

ковому числу монет. За какое минимальное число дней можно взять все монеты?»

Известно следующее решение этой задачи: представим все числа, соответствующие монетам в мешках, в двоичной системе счисления. Задача будем решена, если из всех разрядов этих чисел путем вынимания нужного количества монет убрать все единицы. Тогда ясно, что верхнее ограничение количества дней, за которое это можно сделать, равняется количеству разрядов самого большого числа в совокупности: $\lceil \log_2(2009) \rceil = 11$. Это доставляет следующее решение:

$$V_1 = [1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1].$$

Такая запись будет означать, что в первый день необходимо вынуть из всех мешков, из которых это возможно, 1024 монеты, во второй день - 512 монет и так далее. Таким образом, все монеты будут вынуты за 11 дней. В дальнейшем будет показано, что это действительно минимальное количество дней в этой задаче. Однако возможны и другие минимальные решения. Например, если на каждом шаге из оставшихся непустых мешков находить и вынимать среднее (или медианное) значение монет, округляя его до ближайшего большего целого, то получаем такое решение:

$$V_2 = [1005, 502, 251, 126, 63, 31, 16, 8, 4, 2, 1],$$

Также возможны и другие варианты, например:

$$V_3 = [1010, 508, 255, 128, 64, 32, 16, 8, 4, 2, 1],$$

$$V_4 = [1000, 502, 253, 127, 64, 32, 16, 8, 4, 2, 1].$$

Заметим, что подобные решения связаны с так называемыми *полными последовательностями* (complete sequence) [8]:

Определение 1. *Неубывающая последовательность натуральных чисел*

$$V = [v_1, v_2, v_3, \ldots]$$

называется полной, если каждое натуральное число можно представить в виде суммы некоторых чисел из V:

$$\forall n \in \mathbb{N} : n = \sum_{i=1}^{\infty} e_i \cdot v_i, \forall i : e_i \in \{0, 1\}.$$

Важной теоремой, позволяющей судить о полноте любой последовательности, является критерий Д.Брауна [8]:

Теорема 1. *Для того, чтобы неубывающая последовательность $V = [v_1, v_2, v_3, \ldots]$ была полной, необходимо и достаточно, чтобы:*

1) $v_1 = 1$;

2) $\forall k > 1 : v_k - 1 \leq v_1 + v_2 + \ldots + v_{k-1}$.

Как следует из определения 1 и теоремы 1, элементы полной последовательности записываются в V в неубывающем порядке. Однако в задаче про мешки удобнее использовать начальный участок полной последовательности, записанный в обратном порядке (такой порядок соответствует порядку получения и использования чисел последовательности). В дальнейшем всегда будем использовать такой обратный порядок, если специально не оговаривается обратное.

Хорошо известно, что последовательность степеней двойки является полной, что легко следует из теоремы 1 и соотношения $\forall l : 2^l - 1 = 2^{l-1} + 2^{l-2} + \ldots + 1$. В задаче с мешками для представления всех чисел от 1 до 2009 в виде суммы чисел последовательности V_1 таким образом потребуется 11 чисел, что фактически эквивалентно представлению каждого числа от 1 до 2009 в двоичной системе счисления с максимальной степенью основания системы счисления не более 10 ($2^{10} = 1024$).

Заметим, что последовательность V_1 позволяет представить в виде суммы своих элементов все числа вплоть до 2047, последовательность V_3 — вплоть до 2028, и только последовательности V_2, V_4 — являются минимальными в том смысле, что максимальное число, которое с помощью них можно представить равно выбранному нами в задаче числу мешков 2009 (поскольку сумма членов этих последовательностей равна 2009). В связи с этим введем следующее определение:

Определение 2. *Для заданного натурального числа n назовем конечную последовательность натуральных чисел* $B = [b_1, b_2, \ldots, b_m], b_i \geq b_{i+1}$ *порождающей, если любое натуральное число* $k < n$ *можно представить в виде суммы элементов этой последовательности:* $\forall k < n : k = \sum_{i=1}^{m} e_i \cdot b_i, e_i \in \{0, 1\}$, *причем соблюдается условие:* $n = \sum_{i=1}^{m} b_i$.

Исходя из данного определения, первые m элементов любой полной последовательности, записанные в обратном порядке, составляют порождающую последовательность для числа $n = \sum_{i=1}^{m} v_i$. Например, последовательность V_1 является порождающей для числа 2047.

Таким образом, можно рассмотреть задачу построения алгоритмов формирования порождающих последовательностей минимальной длины и общую задачу построения регулярных (без перечисления, подбора) алгоритмов формирования порождающих последовательностей.

Ниже рассмотрены некоторые алгоритмы формирования порождающих последовательностей. Доказанные свойства этих алгоритмов дают возможность строить новые алгоритмы сортировки целочисленных последовательностей без сравнения чисел, похожие на класс алгоритмов поразрядной сортировки [2; 3; 6; 7].

2. Основные алгоритмы, леммы и теоремы

Лемма 1. *Для любого натурального числа n последовательность чисел b_i, получаемая по алгоритму* ***BinAlgorithm*** *является порождающей последовательностью длины $m = \lceil \log_2 n \rceil$, если n не является степенью двойки, или длины $m = \lceil \log_2 n \rceil + 1$ в противном случае.*

1: **Функция** BinAlgorithm(n)
2: $a \leftarrow n, i \leftarrow 1$
3: **Пока** $a \neq 0$ **выполнять**
4: $B[i] \leftarrow \lceil \frac{a}{2} \rceil$
5: $a \leftarrow a - B[i]$
6: $i \leftarrow i + 1$
7: **Конец цикла**
8: **return** B ▷ Возвращаем порождающую последовательность
9: **Конец функции**

Доказательство. Вначале докажем, что последовательность чисел b_i, получаемая по алгоритму **BinAlgorithm**, является порождающей последовательностью.

Примем обозначения $a_1 = n, b_i = \lceil \frac{a_i}{2} \rceil, a_{i+1} = a_i - b_i$ (b_i соответствует i-му элементу массива B, a_i соответствует переменной a на i-м шаге алгоритма **BinAlgorithm**).

Для того, чтобы доказать, что $B = [b_1, b_2, \ldots, b_m]$ порождающая последовательность для n, рассмотрим возрастающую последовательность чисел $T_1 = [1, 2, 3, \ldots, n]$.

На первом шаге алгоритма **BinAlgorithm** в качестве элемента порождающей последовательности выбирается число $b_1 = \lceil \frac{n}{2} \rceil$. Представим последовательность T_1 следующим образом:

$$n \equiv 1 \mod 2 \Rightarrow T_1 = [1, 2, 3, \ldots, \lfloor \frac{n}{2} \rfloor - 1, \lfloor \frac{n}{2} \rfloor, \lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil + 1, \ldots, n],$$

$$n \equiv 0 \mod 2 \Rightarrow T_1 = [1, 2, 3, \ldots, \lceil \frac{n}{2} \rceil - 1, \lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil + 1, \ldots, n].$$

Вычтем полученное число b_1 из всех чисел последовательности T_1, из которых это возможно, получив тем самым последовательность T_2:

$$n \equiv 1 \mod 2 \Rightarrow T_2 = [1, 2, 3, \ldots, \lfloor \frac{n}{2} \rfloor - 1, \lfloor \frac{n}{2} \rfloor, 0, 1, \ldots, n - \lceil \frac{n}{2} \rceil]$$

$$n \equiv 0 \mod 2 \Rightarrow T_2 = [1, 2, 3, \ldots, \lceil \frac{n}{2} \rceil - 1, 0, 1, \ldots, n - \lceil \frac{n}{2} \rceil]$$

Заметим, что

$$n = \lfloor \frac{n}{2} \rfloor + \lceil \frac{n}{2} \rceil \Rightarrow n - \lceil \frac{n}{2} \rceil = \lfloor \frac{n}{2} \rfloor$$

Как видно из последовательностей T_2, при любой четности числа n полученное число $a_2 = n - \lceil \frac{n}{2} \rceil = \lfloor \frac{n}{2} \rfloor$ остается максимальным в последовательности T_2.

Продолжим действовать подобным образом для всех последующих шагов получения элементов порождающей последовательности $b_i, i \geq 2$, вычитая из всех элементов $t \in T_i$ элемент b_i, если $t \geq b_i$, получая тем самым последовательность T_{i+1}. Тогда на каждом шаге алгоритма **BinAlgorithm** числа последовательности T_i, не меньшие очередного b_i, будут уменьшаться. В виду рассуждений, аналогичных для T_2, на каждом шаге i число $a_i = a_{i-1} - \lceil \frac{a_{i-1}}{2} \rceil = \lfloor \frac{a_{i-1}}{2} \rfloor$ будет оставаться максимальным в последовательности T_i. Как только некоторое число $k < n$ из T_1 на i-м шаге в последовательности T_{i+1} становится равным нулю, это эквивалентно разложению его в виде суммы $k = \sum_{j=1}^{i} e_j \cdot b_j, e_j \in \{0, 1\}$. Число n при таком разложении будет включать в себя все числа b_i. Действительно, имеем:

$$n = a_1 = a_2 + b_1, \ldots, a_i = a_{i+1} + b_i, \ldots, a_m = a_{m+1} + b_m, a_{m+1} = 0$$

Подставляя данные выражения последовательно друг в друга, получим $n = b_1 + b_2 + \ldots + b_k + \ldots + b_m$.

Теперь докажем утверждение леммы относительно длины порождающей последовательности.

Если число n является степенью двойки, то доказательство легко проводится по индукции. Непосредственно для $n = 2^0 = 1$ убеждаемся, что алгоритму требуется один шаг. Предполагаем, что для $n = 2^l$ требуется $l + 1$ шаг. Поскольку на каждом шаге алгоритма число a_{i+1} будет уменьшаться ровно в два раза относительно a_i, то ясно, что для $n = 2^{l+1}$ уже потребуется $l + 2$ шага.

Для доказательства того, что длина порождающей последовательности для фиксированного числа n, не являющегося степенью двойки, равна $\lceil \log_2 n \rceil$, рассмотрим две последовательности:

$$T_1' = [1, 2, 3, \ldots, 2^{\lceil \log_2 n \rceil - 1}], T_1'' = [1, 2, 3, \ldots, 2^{\lceil \log_2 n \rceil}].$$

Из доказанного следует, что для последовательности T_1' количество шагов алгоритма **BinAlgorithm** равно $m' = \lceil \log_2 n \rceil$, для последовательности T_1'' число шагов равно $m'' = \lceil \log_2 n \rceil + 1$. Если число n не является степенью двойки, то имеем $2^{\lceil \log_2 n \rceil - 1} < n < 2^{\lceil \log_2 n \rceil}$. Поскольку число n не является степенью двойки, то хотя бы для одного шага алгоритма оставшееся число $a_i = \lfloor \frac{a_{i-1}}{2} \rfloor$ уменьшается более чем в два раза (это справедливо для любого $a_{i-1} \equiv 1 \mod 2$). Значит число шагов m не менее чем m', но меньше чем m''. Но числа m'' и m' отличаются на единицу, значит $m = m'$.

□

Лемма 2. *Порождающая последовательность чисел b_i, получаемая по алгоритму* ***BinAlgorithm*** *для заданного числа n имеет минимальную длину.*

Доказательство. Пусть $B = [b_1, b_2, \ldots, b_m]$ — порождающая последовательность для заданного числа n, полученная по алгоритму **BinAlgo rithm**. Примем систему двоичного представления всех чисел

$$k = \sum_{i=1}^{m} e_i \cdot b_i \leq n, e_i \in \{0, 1\} :$$

будем представлять каждое число k в виде m-разрядного двоичного числа, $m-i+1$ разряд числа k будет равен e_i. В лемме 1 было показано, что такое представление может быть найдено для любого $k \leq n$.

Прежде всего заметим, что если $n = 2^l - 1$, то, поскольку справедливо соотношение $2^l - 1 = 2^{l-1} + 2^{l-2} + \ldots + 1$, все элементы порождающей последовательности B будут вида $\forall i : b_i = 2^{l-1}, 2^{l-2}, \ldots, 2, 1$. Причем такая B является единственной порождающей последовательностью длины l, поскольку введенное двоичное представление любого числа $k \leq n$ в этом случае совпадает с представлением этого числа k в двоичной позиционной системе счисления с помощью l разрядов, а такое представление единственно. Кроме этого, для представления числа $n = 2^l = 2^{l-1} + 2^{l-2} + \ldots + 1 + 1$ минимально требуется уже $(l + 1)$ двоичных разрядов, чему соответствует порождающая последовательность $\forall i : b_i = 2^{l-1}, 2^{l-2}, \ldots, 2, 1, 1$.

Если теперь $2^l \leq n \leq 2^{l+1} - 1$, то из доказанного следует, что для такого n порождающая последовательность не может иметь длину меньше $(l + 1)$, иначе число 2^l можно было бы представить такой последовательностью, что невозможно из доказанного. В тоже время для представления числа $2^{l+1} - 1$ также достаточно $(l + 1)$ разрядов. Значит минимальная длина порождающей последовательности для n из указанного диапазона равна $(l + 1)$. Как следует из леммы 1, для числа, являющегося степенью двойки $n = 2^l$, порождающая последовательность имеет длину $(l + 1)$. Также из леммы 1 следует, что $\forall n = 2^l + i, 1 \leq i < 2^l : \lceil \log_2(2^l + i) \rceil = l + 1$. □

Теорема 2. *Для любого натурального числа n и любого вещественного положительного числа c, такого, что $\lceil \frac{n}{c} \rceil \leq \frac{n+1}{2}$ последовательность чисел b_i, получаемая по алгоритму* ***CAlgorithm*** *является порождающей последовательностью длины $m \leq \lfloor \log_{\frac{c}{c-1}} n \rfloor + 1$.*

1: ***Функция*** CALGORITHM*(n, c)*
2: $a \leftarrow n, i \leftarrow 1$
3: ***Пока*** $a \neq 0$ ***выполнять***
4: $B[i] \leftarrow \lceil \frac{a}{c} \rceil$

5: $a \leftarrow a - B[i]$
6: $i \leftarrow i + 1$
7: ***Конец цикла***
8: ***return*** B ▷ *Возвращаем порождающую последовательность*
9: ***Конец функции***

Доказательство. Как видно, алгоритм **BinAlgorithm** из леммы 1 является частным случаем алгоритма **CAlgorithm** и получается из **CAlgorithm** при $c = 2$. Будем использовать аналогичные обозначения: элемент b_i соответствует i-му элементу массива B, a_i соответствует значению переменной a на i-м шаге алгоритма **CAlgorithm**. Также примем $a_1 = n, b_i = \lceil \frac{a_i}{c} \rceil, a_{i+1} = a_i - b_i$ Рассмотрим начальную последовательность чисел:

$$T_1 = [1, 2, 3, \ldots, \lceil \frac{n}{c} \rceil - 1, \lceil \frac{n}{c} \rceil, \lceil \frac{n}{c} \rceil + 1, \ldots, n]$$

Согласно алгоритму **CAlgorithm** на первом шаге в качестве элемента порождающей последовательности принимают $b_1 = \lceil \frac{n}{c} \rceil$. Действуя аналогично доказательству леммы 1, преобразуем последовательность T_1 к виду T_2, вычитая b_1 из всех $t \in T_1, t \geq b_1$:

$$T_2 = [1, 2, 3, \lceil \frac{n}{c} \rceil - 1, 0, 1, \ldots, n - \lceil \frac{n}{c} \rceil]$$

Аналогично лемме 1, рассмотрим условие, при котором число $n - \lceil \frac{n}{c} \rceil$ остается максимальным на первом шаге (а следовательно, как показано в лемме 1, и на всех последующих):

$$\lceil \frac{n}{c} \rceil - 1 \leq n - \lceil \frac{n}{c} \rceil \Leftrightarrow \lceil \frac{n}{c} \rceil \leq \frac{n+1}{2}$$

Таким образом, при выполнении условия $\lceil \frac{n}{c} \rceil \leq \frac{n+1}{2}$ каждое получаемое число a_i будет оставаться максимальным числом в последовательности T_i, и значит последовательность чисел b_i, получаемая по алгоритму **CAlgorithm**, будет порождающей последовательностью.

Для оценки длины порождающей последовательности рассмотрим:

$$a_{i+1} = a_i - b_i = a_i - \lceil \frac{a_i}{c} \rceil = \lfloor a_i - \frac{a_i}{c} \rfloor = \lfloor a_i \cdot \frac{(c-1)}{c} \rfloor$$

Таким образом, на каждом шаге последующее a_{i+1} уменьшается не менее чем в $\frac{c}{c-1}$ раз, следовательно, не более чем через $\lfloor \log_{\frac{c}{c-1}} n \rfloor$ шагов $a_{m-1} = 1$, и потребуется еще один шаг для уменьшения оставшегося числа до нуля $a_m = 0$. □

Теорема 3. *Пусть для заданных n и c, таких, что $\lceil \frac{n}{c} \rceil \leq \frac{n+1}{2}$, по алгоритму **CAlgorithm** получена порождающая последовательность*

$B = [b_1, b_2, \ldots, b_m]$, тогда всегда существует не менее одного, но не более двух натуральных чисел $n' > n$, для которых порождающая последовательность по алгоритму ***CAlgorithm*** *будет $B' = [b_0, b'_1, b'_2 \ldots, b'_m]$ такая, что $\forall i = 1..m : b_i = b'_i$.*

Доказательство. Для того, чтобы в порождающей последовательности B' полностью повторилась B, положим $b_0 = \lceil \frac{n'}{c} \rceil, n = n' - b_0$. Имеем:

$$n = n' - b_0 = n' - \lceil \frac{n'}{c} \rceil = \lfloor n' \cdot \frac{c-1}{c} \rfloor, n - \lfloor n' \cdot \frac{c-1}{c} \rfloor = 0$$

$$\lceil n - n' \cdot \frac{c-1}{c} \rceil = 0 \tag{2.1}$$

Все решения n' уравнения (2.1) для любых c могут быть найдены из неравенств:

$$-1 < n - n' \cdot \frac{c-1}{c} \leq 0 \Rightarrow \frac{nc}{c-1} \leq n' < \frac{(n+1) \cdot c}{c-1}$$

С учетом того, что n' - целое число:

$$\lceil \frac{nc}{c-1} \rceil \leq n' \leq \lceil \frac{(n+1) \cdot c}{c-1} - 1 \rceil \Leftrightarrow \lceil \frac{nc}{c-1} \rceil \leq n' \leq \lceil \frac{nc+1}{c-1} \rceil$$

Числа $\lceil \frac{nc}{c-1} \rceil$, $\lceil \frac{nc+1}{c-1} \rceil = \lceil \frac{nc}{c-1} + \frac{1}{c-1} \rceil$ могут быть либо равны друг другу, либо отличаться не более чем на 1. Соответственно, получаются следующие возможные решения для n':

$$n'_1 = \lceil \frac{nc}{c-1} \rceil, n'_2 = \lceil \frac{nc+1}{c-1} \rceil$$

□

Следствие 1. *Выбрав произвольное натуральное число n, а также число c из условия $\lceil \frac{n}{c} \rceil \leq \frac{n+1}{2}$, можно сформировать последовательность $A = [a_1, a_2, \ldots, a_k, \ldots, a_{m+1}], a_{m+1} = n, a_{k-1} = a_k - \lceil \frac{a_k}{c} \rceil, a_2 = 1, a_1 = 0$ с последующим добавлением в нее элементов до бесконечности:*

$$n' = \lceil \frac{nc}{c-1} \rceil \vee n' = \lceil \frac{nc+1}{c-1} \rceil, A' = [a_1, a_2, \ldots, a_k, \ldots, a_{m+1}, n']$$

$$n'' = \lceil \frac{n'c}{c-1} \rceil \vee n'' = \lceil \frac{n'c+1}{c-1} \rceil, A'' = [a_1, a_2, \ldots, a_k, \ldots, a_{m+1}, n', n'']$$

$$\ldots$$

$$n^{(k)} = \lceil \frac{n^{(k-1)}c}{c-1} \rceil \vee n^{(k)} = \lceil \frac{n^{(k-1)}c+1}{c-1} \rceil,$$

$$A^{(k)} = [a_1, a_2, \ldots, a_k, \ldots, a_{m+1}, n', n'', \ldots, n^{(k)}]$$

и т.д. Тогда последовательность B' первых разностей от $A^{(k)}$ будет полной последовательностью: $B' = [b'_1, b'_2, \ldots, b'_{m+k}, \ldots], \forall i : b'_i = a_{i+1} - a_i$.

3. Алгоритм сортировки

Как показано при доказательстве лемм 1, 2, порождающие и полные последовательности можно использовать для задания на множестве натуральных чисел своеобразной двоичной системы счисления, по аналогии с системой счисления на основе чисел Фибоначчи. Это в свою очередь позволяет использовать порождающие последовательности для упорядочения натуральных чисел. Рассмотрим один вариант подобного алгоритма.

Пусть задана произвольная последовательность натуральных чисел $Q=[q_1,q_2,\ldots,q_k]$, числа могут повторяться, при этом пусть n=max(Q) — максимальное из чисел последовательности Q. Пусть далее $S=[s_1,s_2,\ldots,s_m], \forall i : s_i > s_{i+1}$ - некоторая порождающая последовательность, полученная по алгоритму **CAlgorithm** (теорема 2) для числа n. Исходя из доказанного, каждое число последовательности Q можно представить в виде суммы некоторых чисел последовательности S. Тогда последовательность S можно использовать для разделения и сортировки чисел последовательности Q по алгоритму **CS-sort**, представленному ниже.

```
1:  Функция DIVISOR(Q, a, x, y)   ▷ Первая вспомогательная функция
2:      i ← x, j ← y, pr ← False
3:      Пока i < j выполнять
4:          Если (Q[i] ≥ a) тогда
5:              pr ← True
6:              Пока (i < j)&(Q[j] ≥ a) выполнять
7:                  j ← (j − 1)
8:              Конец цикла
9:              Q[i], Q[j] ← Q[j], Q[i]
10:         Конец условия
11:         i ← i + 1
12:     Конец цикла
13:     return [j, pr]
14: Конец функции
15: Функция ITER-SORT(Q, S, a, x, y)        ▷ Вторая вспомогательная
    функция
16:     Если x >= y тогда
17:         return True
18:     Иначе если S = []
19:         return True
20:     Иначе
21:         [z, pr] ← Divisor (Q, (a + S[0]), x, y)
22:         Если pr тогда
23:             Iter-sort(Q, S[1 :], (a + S[0]), z, y)
```

Iter-sort$(Q, S[1:], a, x, z-1)$
Иначе
Iter-sort$(Q, S[1:], a, x, y)$
Конец условия
Конец условия
Конец функции
Функция Cs-sort(Q, S) ▷ Главная функция
Iter-sort$(Q, S, 0, 0, length(Q)-1)$
Конец функции

В представленном алгоритме используются две вспомогательные функции. Функция Divisor(Q, a, x, y) разделяет последовательность Q внутри индексов x, y относительно числа a, перемещая все числа, меньшие a влево относительно чисел, больших либо равных a, внутри диапазона $[x, y]$. Функция Divisor в качестве результата возвращает список из новой границы раздела j, такого, что все числа из диапазона $[x, j)$ меньше a, все числа из диапазона $[j, y]$ больше либо равны a, и логического признака, указывающего, что разделение чисел относительно числа a имело место. Функция Iter-sort(Q, S, a, x, y) осуществляет разделение последовательности Q относительно каждого элемента порождающей последовательности S. Принято обозначение $S[1:]$ — возвращает хвост последовательности S (без головного элемента). В качестве аргумента a функции Iter-sort выступает сумма элементов последовательности S до текущей границы, для чего вызывается функция Divisor. В функции Iter-sort осуществляется рекурсивный вызов относительно новой границы раздела z, причем для левой границы раздела от x до $z-1$ значение a не меняется, для правой границы раздела от z до y разделение происходит относительно a плюс головной элемент текущей подпоследовательности S. Окончательно, главная функция Cs-sort просто вызывает функцию Iter-sort при $a = 0$, $x = 0$ — в качестве левой границы берется первый элемент последовательности Q, $y = length(Q) - 1$ — в качестве правой границы берется последний элемент последовательности Q при нумерации с нуля.

Для оценки вычислительной сложности предложенного алгоритма заметим, что для каждого элемента порождающей последовательности S в сумме всех вызовов функции Divisor осуществляется проход по всей длине последовательности Q. В целом потребуется $length(Q) \cdot length(S)$ шагов алгоритма. Причем порождающая последовательность S минимальной длины дает минимальное число шагов алгоритма (при параметре $c = 2$). С учетом доказанных утверждений (теорема 2) число шагов $F(Q)$ алгоритма может быть оценено $F(Q) \sim C \cdot k \cdot (\lfloor \log_{\frac{c}{c-1}} \max(Q) \rfloor + 1)$, где C — некоторая константа, k — длина последовательности Q, c — выбранный параметр для получения порождающей последовательности S. Таким образом, при фиксации разрядной сетки предложенный

алгоритм относится к классу алгоритмов с оценкой вычислительной сложности $O(k)$ без сравнения чисел. Если $k = 2^l, k = 2^l - 1, c = 2$, то порождающая последовательность, как следует из лемм 1, 2, представляет собой последовательность убывающих степеней 2, поэтому алгоритм фактически совпадает с поразрядной MSD сортировкой с основанием системы счисления 2 [3; 6]. Однако, в отличие от MSD, предложенный алгоритм сортировки не является устойчивым, поскольку в функции Divisor элементы большие либо равные a меняются местами с первым встретившимся с конца элементом, меньшим a, что не гарантирует сохранение взаимного расположения равных элементов. С другой стороны, алгоритм можно рассматривать и как вариацию алгоритма быстрой сортировки Хоара, где в качестве опорных элементов выбираются подходящие суммы элементов порождающей последовательности.

Анализ алгоритма **CS-sort** показывает, что для некоторых элементов порождающей последовательности $S = [s_1, s_2, \ldots, s_m], \forall i : s_i > s_{i+1}$ может не происходить разделение сортируемой последовательности $Q = [q_1, q_2, \ldots, q_k]$. Например, если $\min(Q) > s_1$, то все элементы Q будут больше s_1, и граница разделения j в функции Divisor будет равна нижней границе x. Таким образом, элемент s_1 не разделит сортируемую последовательность, но обеспечит дополнительные затраты времени на проход по последовательности Q от x до y в функции Divisor. Также возможна и противоположная ситуация: если $\max(Q) < s_1$, то все элементы Q будут меньше s_1, и граница разделения в функции Divisor будет равна верхней границе y. Поэтому элемент s_1 опять же не разделит сортируемую последовательность, но даст дополнительные затраты времени. Тогда верными оказываются следующие два утверждения:

1) Если $\min(Q) > s_1$, то для сортировки Q можно использовать последовательность $[\sum_{i=1}^{p} s_i, s_{p+1}, \ldots, s_m]$, где p определяется из условия $\sum_{i=1}^{p} s_i \leq \min(Q)$.

2) Если $\max(Q) < s_1$, то для сортировки Q можно использовать последовательность $[s_p, \ldots, s_m], \forall i = 1 \ldots (p-1) : s_i > \max(Q), s_p \leq \max(Q)$.

Рассмотрим ряд примеров на возникновение таких ситуаций.

Пример 1. Пусть требуется отсортировать последовательность $Q = [49, 43, 46, 48, 41, 47, 38, 35, 39, 37]$. Для данной последовательности алгоритм **BinAlgorithm** даст порождающую последовательность $S_1 = [25, 12, 6, 3, 2, 1], length(S_1) = 6$. Для сравнения поразрядная сортировка на основе двоичного представления будет осуществляться по последовательности $S_2 = [32, 16, 8, 4, 2, 1], length(S_2) = 6$. Как видно, обе последовательности удовлетворяют условию первого утверждения $\min(Q) > s_1$, однако для обеих последовательностей сокращение невозможно, так как

$\sum_{i=1}^{p} s_i \le \min(Q)$ только при $p = 1$. В тоже время сокращение S_1 было бы уже возможно, если бы в последовательности Q отсутствовало число 35. Этот факт связан с тем, что в алгоритме **BinAlgorithm** порождающая последовательность S_1 имеет ограничение $\max(Q) = \sum_{i=1}^{m} s_i = 49$, тогда как S_2 представляет все числа до $2^6 - 1$ включительно (и является порождающей последовательностью по алгоритму **BinAlgorithm** для числа $2^6 - 1$). Поэтому начальная цепочка в S_2 будет состоять из больших чисел, чем в S_1. Если использовать алгоритм **CAlgorithm** при $c > 2$, то очевидно, что распределение чисел в порождающей последовательности будет еще более равномерное. В тоже время при некоторых $c > 2$ длина порождающей последовательности может не увеличиться. Например, при $c = 2.2$ получаем порождающую последовательность $S_3 = [23, 12, 7, 4, 2, 1], length(S_3) = 6$, для которой уже возможно сокращение до $S'_3 = [35, 7, 4, 2, 1]$. Отметим также, что с учетом рассмотренных условий по сокращению длины последовательности в ряде случаев порождающие последовательности большей длины алгоритма **CAlgorithm** при $c > 2$ могут не уступать порождающим последовательностям по алгоритму **BinAlgorithm**. Например, при $c = 2.5$ получаем порождающую последовательность $S_4 = [20, 12, 7, 4, 3, 2, 1], length(S_4) = 7$, которую в нашем случае можно сократить до $S'_4 = [32, 7, 4, 3, 2, 1], length(S'_4) = 6$.

Заметим также, что при выполнении условия $\min(Q) > s_1$ иногда удобнее осуществить преобразование шкалы и вместо последовательности Q сортировать последовательность $Q', \forall q'_i \in Q' : q'_i = q_i - \min(Q) + 1$. В нашем примере это дает $Q' = [15, 9, 12, 14, 7, 13, 4, 1, 5, 3]$ с порождающей последовательностью $S' = [8, 4, 2, 1], length(S') = 4$. При этом на практике вычитание можно не выполнять в явном виде, а только сравнивать в функции Divisor вместо q_i элементы q'_i с элементами последовательности S', передавая в Divisor элемент $\min(Q)$. Также поиск очередных $\min(Q), \max(Q)$ внутри интервала $[x, y]$ можно совместить с разделением чисел внутри этого интервала в функции Divisor.

Выясним условия, при которых эти преобразования действительно приводят к сокращению порождающей последовательности. В условиях первого утверждения для этого как минимум должно выполняться неравенство $s_1 + s_2 \le \min(Q)$, что соответствует следующему неравенству:

$$\lceil \frac{\max(Q)}{c} \rceil + \lceil \frac{\lfloor \max(Q) \cdot \frac{c-1}{c} \rfloor}{c} \rceil \le \min(Q), \tag{3.1}$$

$$\min(Q) \ge 2 + \max(Q) \cdot \frac{2c - 1}{c^2} \tag{3.2}$$

Последнее неравенство (3.2) представляет собой оценку сверху для (3.1), в действительности сокращение порождающей последовательности возможно и при меньших значениях $\min(Q)$ (например, в рассматриваемой последовательности Q при $c = 2$ оценка (3.2) дает

$\min(Q) \geq 39$, однако, как было показано, сокращение уже возможно при $\min(Q) = 37$). На практике при $c = 2$ сокращения можно ожидать, если $\min(Q) \geq \frac{3}{4}\max(Q)$.

Для условий сокращения на основе преобразования шкалы при $c = 2$ должно выполняться неравенство:

$$\lfloor \log_2(\max(Q)) \rfloor - \lfloor \log_2(\max(Q) - \min(Q) + 1) \rfloor > 1 \qquad (3.3)$$

Пример 2. Пусть теперь требуется отсортировать последовательность $Q = [49, 43, 46, 48, 41, 47, 38, 1000, 39, 37]$. Тогда порождающая последовательность по алгоритму **BinAlgorithm** будет $S_1 = [500, 250, 125, 63, 31, 16, 8, 4, 2, 1], length(S_1) = 10$. Однако ясно, что после использования первого элемента $S_1[1] = 500$ мы получим промежуточную последовательность $Q' = [49, 43, 46, 48, 41, 47, 38, 37, 39, 1000]$, в которой будет требоваться отсортировать подпоследовательность $Q'[1 : 9] = [49, 43, 46, 48, 41, 47, 38, 37, 39]$ с помощью порождающей последовательности $S_1' = [250, 125, 63, 31, 16, 8, 4, 2, 1]$. Здесь будут выполняться условия $\max(Q) < s_1$ и вместо S_1' можно использовать $S_1'' = [31, 16, 8, 4, 2, 1]$, $length(S_1'') = 6$, или порождающую последовательность из предыдущего примера $[25, 12, 6, 3, 2, 1]$, которую можно теперь сократить до $[37, 6, 3, 2, 1]$, или с использованием преобразования шкалы до $[7, 3, 2, 1]$.

Отметим, что кроме описанных подходов, можно использовать частичное разделение последовательности Q с порождающей последовательностью $S = [s_1, s_2, \ldots, s_m]$. Например, при $c = 2$, как следует из (3.2), сортировка на начальной цепочке порождающей последовательности S из двух элементов $[s_1, s_2]$ производит разделение последовательности Q на четыре части: $\forall i_1 : 1 \leq q_{i_1} < s_2$, $\forall i_2 : s_2 \leq q_{i_2} < s_1$, $\forall i_3 : s_1 \leq q_{i_3} < s_1 + s_2$, $\forall i_4 : s_1 + s_2 \geq q_{i_4}$. После этого каждую такую часть можно сортировать отдельно с возможными сокращениями получающихся для них порождающих последовательностей.

Из рассмотренных примеров следует, что использование порождающих последовательностей для сортировки натуральных чисел обеспечивает большую гибкость и лучше может быть адаптировано под конкретную сортируемую последовательность.

Список литературы

1. Артамонов Ю. Н. Групповой выбор с использованием матричных норм / Ю. Н. Артамонов // Изв. Иркут. гос. ун-та. Сер. Математика. — 2016. – Т. 18. – С. 3–20.
2. Кнут Д. Э. Искусство программирования. Т. 3. Сортировка и поиск / Д. Э. Кнут. — 2-е изд. — М. : Вильямс,2007. – С. 274–387.

3. Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. М. Штайн. — 2-е изд. — М. : Вильямс, 2005. – С. 220–238.
4. Научный форум dxdy. Модификация старой задачи про мешки с монетами [Электронный ресурс]. — URL: http://dxdy.ru/topic111132-30.html (дата обращения: 02.11.2017).
5. Национальный открытый университет «ИНТУИТ» [Электронный ресурс]. — URL: http://www.diofant.ru/problem/383/ (дата обращения: 02.11.2017).
6. Седжвик Р. Фундаментальные алгоритмы на C++. Анализ. Структуры данных. Сортировка. Поиск / Р. Седжвик. — Киев : ДиаСофт, 2001. — С. 401–438.
7. Goodrich M. 4.5 Bucket-Sort and Radix-Sort / M. Goodrich, T. Michael, R. Tamassia // Algorithm Design: Foundations, Analysis, and Internet Examples. — John Wiley, 2002. — P. 241–243.
8. Honsberger R. Mathematical Gems III / R. Honsberger. — Washington, DC : Math. Assoc. Amer., 1985. — P. 123–128.

Артамонов Юрий Николаевич, кандидат технических наук, ФГБНУ «Государственный научно-методический центр», 115093, Россия, г. Москва, ул. Люсиновская, 51, тел.: (499) 706-81-25
(e-mail: junaart@mail.ru)

Y. N. Artamonov

Using the Complete Sequences for Sorting Natural Numbers

Abstract. Complete sequences are defined as infinite sequences of natural numbers, with the help of which it is possible to represent any other natural number. The most significant result, allowing to judge the completeness of any sequence, was obtained by D. Brown. The article poses the problem of representing the complete sequence of all natural numbers up to a certain limit as a sum of elements (such initial segments of complete sequences are called generating sequences). Then there is the problem of finding generating sequences of minimal length for a given limit N. The article proposes algorithms for generation of generating sequences of minimal length. A class of algorithms for generation of generating sequences containing a given generating sequence of shorter length is proposed, it allows us to introduce regular algorithms of generating complete sequences. The proposed regular algorithms for generating complete sequences are used in the development of the algorithm for sorting natural numbers without comparing them, which is the development of the radix sorting algorithm with interpretation of the bits as elements of a suitable complete sequence. The article demonstrates approaches of adapting the work of this algorithm for sorting a specific sorted sequence.

Keywords: complete sequences, sorting algorithms for linear time, radix sort, non-traditional number systems, algorithms for searching and storing of numerical data.

References

1. Artamonov Y.N. Group choice using matrix norms. *Izv. Irkutsk. Gos. Univ. Ser. Mat.*, 2016, vol. 18, pp. 3-20. (in Russian).
2. Knuth D.E. *Iskusstvo programmirovanija, vol. 3. Sortirovka i poisk* [The art of computer programming, volume 3. Sorting and searching]. Vil'jams, 2007, pp. 274-387.

3. Cormen T., Leiserson C., Rivest R., Stein C. *Algoritmy: postroenie i analiz* [Algorithms: construction and analysis.] Vil'jams, 2005, pp. 220-238.
4. Nauchnyj forum dxdy. Modifikacija staroj zadachi pro meshki s monetami (jelektronnyj resurs) [The scientific forum dxdy. Modification of the old problem of bags with coins (electronic resource)] Available at: http://dxdy.ru/topic111132-30.html (accessed 2 November 2017).
5. Nacional'nyj otkrytyj universitet «INTUIT», proekt diofant.ru (jelektronnyj resurs)[National Open University « INTUIT », project diofant.ru (electronic resource)] Available at: http://www.diofant.ru/problem/383/ (accessed 2 November 2017).
6. Sedjvik R. *Fundamental'nye algoritmy na C++. Analiz. Struktury dannyh. Sortirovka. Poisk* [Fundamental algorithms in C++. Analysis. Data structures. Sorting. Search.] DiaSoft, 2001, pp. 401-438.
7. Goodrich M., Michael T., Tamassia, R. 4.5 Bucket-Sort and Radix-Sort. *Algorithm Design: Foundations, Analysis, and Internet Examples.* John Wiley, 2002, pp. 241-243.
8. Honsberger R. Mathematical Gems III. Washington, DC: Math. Assoc. Amer., 1985, pp. 123-128.

Artamonov Yuri Nikolaevich, Candidate of Sciences (Technic), Federal State budget scientific institution "State Scientific-Methodological Centre", 51, Lyusinovskaya st., Moscow, Russian Federation, 115093, tel.: (499) 706-81-25 (e-mail: junaart@mail.ru)