Основы программирования

Лекция 2

Алгоритмы на массивах:

- поиск,
- простые сортировки
- перестановки

Задача поиска

Объекты в общем случае будем рассматривать как записи произвольной природы, однако имеющие в своей структуре один и тот же *ключ* — поле, содержащее значение, которое сравнивается в процессе поиска с искомым ключом. В более общем случае ключ можно рассматривать как числовую функцию, которая строит значение ключа на основании сколь угодно сложного анализа всех данных, представленных в записи.

Далее при рассмотрении методов поиска и сортировки мы для простоты будем отождествлять записи с их ключами.

Последовательный поиск

Начинаем просмотр с первого элемента массива, продвигаясь дальше до тех пор, пока не будет найден нужный элемент или пока не будут просмотрены все элементы массива.

```
#include <stdio.h>
#define N 100

int main()
{
   int x;
   int a[N];
   int  i = 0;
   ...                    // Ввод массива и x
   while ((i < N) && (a[i] != x))
       i++;
   if (i < N)
        printf("%d", i);  // элемент найден
   else
        printf("No");     // элемент не найден
    return 0;
}
```

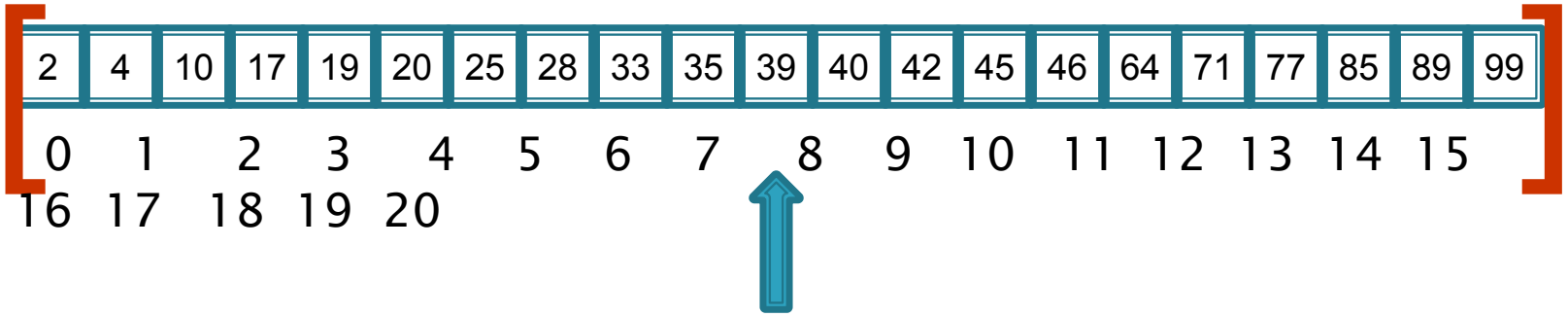
Бинарный поиск в массиве

Условие применения:

массив должен быть отсортированным.

Идея:

массив на каждом шаге делится пополам и выбирается та его часть, в которой должен находиться искомый элемент.



X = 33

Бинарный поиск - программа

```
#include <stdio.h>
#define N 100
int main()
{
   int x;
   int a[N];
   int left = 0;
   int right = N - 1;
   int middle;
   // Ввод массива и x
   do
   {
      middle = (left + right) / 2;
      if (x == a[middle])
         break;
      else
         if (a[middle] < x)
             left = middle + 1;
         else right = middle - 1;
      } while (left <= right);
   if (left <= right)
       printf("%d", middle);  /* элемент найден */
   else
       printf("No");           /* элемент не найден */
   return 0;
}
```

Максимальное число сравнений равно $log_2 N$.

Задача сортировки

Задача сортировки состоит в том, чтобы упорядочить N объектов $a_1, \ldots, a_N$:
переставить их в такой последовательности $a_{p1}, \ldots, a_{pN}$,
чтобы их ключи расположились в неубывающем порядке

$$k_{p1} \le k_{p2} \le \ldots \le k_{pN}.$$

Свойство устойчивости сортировки

Сортировка называется *устойчивой*,

если она удовлетворяет условию, согласно которому записи с одинаковыми ключами остаются в прежнем порядке:

$$k_{Pi} = k_{Pj} \; u \; i < j, \; mo \; p_i < p_j$$

При устойчивой сортировке относительный порядок элементов с одинаковыми ключами не меняется.

Виды сортировок

Методы сортировки обычно разделяют на две категории:

- *внутреннюю* сортировку массивов и
- *внешнюю* — сортировку файлов.

Методы сортировки можно разбить на несколько основных классов в зависимости от лежащего в их основе приема сортировки:

- включения (вставки);
- выбора;
- обмена;
- подсчета;
- разделения;
- слияния.

Сортировка включением

Разделим условно все элементы массива на две последовательности: *входную* $a_i, \dots, a_N$ и *готовую* последовательность $a_1, \dots, a_{i-1}$ элементы которой уже отсортированы.

В алгоритмах, основанных на методе включения, на каждом i-м шаге i-й элемент входной последовательности вставляется в подходящее место готовой последовательности.

Сортировка простыми включениями наиболее очевидна.

Пусть $2 < i < N$, $a_1, \dots, a_{i-1}$ уже отсортированы:

$$a_1 \le a_2 \le \dots \le a_{i-1}.$$

Будем сравнивать по очереди a_i с a_{i-1}, a_{i-2}, ... до тех пор, пока не обнаружим, что элемент a_i следует вставить между a_j и a_{j+1} ($0 \le j \le i - 1$) элементами.

После этого или в процессе поиска подвинем записи $a_{j+1}, \dots, a_{i-1}$ на одно место вправо и переместим запись a_i в позицию $j + 1$.

Пример

Процесс сортировки включениями покажем на примере последовательности, состоящей из восьми ключей:

```
i = 1    40 | 51 8   38  90  14  2  63
i = 2    40 51 | 8   38  90  14  2  63
i = 3    8  40  51 | 38  90  14  2  63
i = 4    8  38  40  51 | 90  14  2  63
i = 5    8  38  40  51  90 | 14  2  63
i = 6    8  14  38  40  51  90 | 2  63
i = 7    2  8   14  38  40  51  90 | 63
i = 8    2  8   14  38  40  51  63  90 |
```

Алгоритм

замечание: массив нумеруется с 1до N

Условно разделить массив A на отсортированную и несортированную части. К сортированной части сначала относится только первый элемент.

```
цикл по i от 2 до N с шагом 1 выполнять
    // i – номер первого элемента в несортированной части массива
    x ← A[i];
    j ← i – 1;
// Все элементы из отсортированной части, большие,
// чем x, сдвинуть на одну позицию вправо:
    пока j > 0 и A[j] > x выполнять
        A[j+1] ← A[j];
        j ← j – 1;
    конец пока
// Элемент x поставить на свое место по порядку:
    A[j+1] ← x;
конец цикла
```

Анализ алгоритма

На *i*-м шаге максимально возможное число сравнений C_i во внутреннем цикле равно *i* -1;

если предположить, что все перестановки *N* ключей равновероятны, число сравнений в среднем равно *i*/2.

Для M_i, количества пересылок на *i*-м шаге, максимальное $M_i = C_i + 2$.

Всего шагов *N* - 1.

Следовательно, количество сравнений и пересылок в худшем и лучшем случаях:

$$C_{\max} = 1 + 2 + \ldots + N - 1 = \frac{N \cdot (N-1)}{2},$$

$$C_{\min} = N - 1,$$

$$M_{\min} = 2 \cdot (N-1),$$

$$M_{\max} = \frac{N \cdot (N-1)}{2} + 2 \cdot (N-1) \approx \frac{N \cdot (N+3)}{2}.$$

Сортировка бинарными включениями

Для нахождения места для *i*-го элемента можно использовать метод бинарного поиска элемента в отсортированном массиве, в котором на *i*-ом шаге выполняется ~ $log_2 i$ сравнений.

Поэтому всего будет произведено ~ $N \cdot log_2 N$ сравнений.

Но количество пересылок в этом методе не изменится

Сортировка простым выбором

Методы сортировки посредством выбора основаны на идее многократного выбора.

На *i*-м шаге выбирается наименьший элемент из входной последовательности a_i, ..., a_n и меняется местами с a_i-м.

Таким образом, после шага *i* на первом месте во входной последовательности будет находиться наименьший элемент.

Затем этот элемент перемещается из входной в готовую последовательность.

Процесс выбора наименьшего элемента из входной последовательности повторяется до тех пор, пока в ней останется только один элемент.

Пример

Проиллюстрируем этот метод на той же последовательности

| 40 51 8 38 90 14 2 63.

На первом шаге находим наименьший элемент 2, обмениваем его с первым элементом 40 и перемещаем в готовую последовательность:

2 | 51 8 38 90 14 40 63

2 8 | 51 38 90 14 40 63

2 8 14 | 38 90 51 40 63

2 8 14 38 | 90 51 40 63

2 8 14 38 40 | 51 90 63

2 8 14 38 40 51 | 90 63

2 8 14 38 40 51 63 | 90

Обсуждение

Данный метод в некотором смысле противоположен сортировке простыми включениями.

При сортировке простым выбором рассматриваются все элементы входной последовательности и для фиксированного места из нее выбирается наименьший элемент.

При этом не возникает необходимости "сдвига" участка массива, поскольку выбранный элемент вставляется всегда в конец готовой последовательности. Вытесняемый же элемент достаточно переставить на освободившееся место в несортированной входной части.

Алгоритм

Условно разделить массив A на отсортированную и несортированную части. Сначала весь массив — это несортированная часть.

цикл по i **от** 1 **до** N–1 **с шагом** 1 **выполнять**

// i – номер первого элемента в несортированной части массива

r ← i;

// Найти минимальный элемент в несортированной части массива:

цикл по j **от i+1 до** N **с шагом** 1 **выполнять**

если A[j] < A[r]

то r ← j;

конец цикла

// Найденный минимальный элемент поменять местами с

// первым элементом несортированной части:

если i ≠ r

то Обмен (i, r);

// Он будет последним элементом новой сортированной части массива A

конец цикла

Анализ

Число C_i сравнений на i-м шаге не зависит от начального порядка элементов.

На первом шаге первый элемент сравнивается с остальными $N - 1$ элементами, на втором шаге число сравнений будет — $N - 2$ и т. д.

Поэтому число сравнений :

$$C = (N – 1) + (N – 2) + ... + 1 = N * (N - 1)/2$$

Максимальное число пересылок $M_{max} = N - 1$, так как на каждом проходе выполняется обмен найденного минимального элемента с i-м.

Вероятность того, что i-й элемент уже стоит на месте, невелика, поэтому средняя оценка M близка к максимальной.

Анализ, продолжение

Мы видим, что число сравнений в методе выбора всегда равно максимальному числу сравнений в методе простых включений, в то время как число перемещений, наоборот, минимально.

Если вспомнить, что сравниваются ключи позиций, а перемещаются записи целиком, то метод выбора, экономящий число перемещений может на практике оказаться предпочтительней.

Сортировка простым обменом

Метод основан на принципе сравнения и обмена пар соседних элементов.

На первом шаге сравним последний и предпоследний элементы, если они не упорядочены, поменяем их местами.

Далее проделаем то же со вторым и третьим элементами от конца массива, третьим и четвертым и т. д. до первого и второго с начала массива.

При выполнении этой последовательности операций меньшие элементы в каждой паре продвинутся влево, наименьший займет первое место в массиве.

Повторим этот же процесс от *N*-го до 2-го элемента, потом от *N*-го до 3-го и т. д.

i-й проход по массиву приводит к «всплыванию» наименьшего элемента из входной последовательности на *i-е* место в готовую последовательность.

Пример

Процесс сортировки обменами покажем на примере все той же последовательности, состоящей из восьми ключей:

i = 0	40	51	8	38	90	14	2	63
i = 1	2	40	51	8	38	90	14	63
i = 2	2	8	40	51	14	38	90	63
i = 3	2	8	14	40	51	38	63	90
i = 4	2	8	14	38	40	51	63	90
i = 5	2	8	14	38	40	51	63	90
i = 6	2	8	14	38	40	51	63	90
i = 7	2	8	14	38	40	51	63	90

Алгоритм (метод пузырька)

```
цикл по i от 2 до N с шагом 1 выполнять
   // проход от конца массива к началу:
  цикл по j от N до i с шагом -1 выполнять
        // если два рядом стоящих элемента нарушают
    // порядок по возрастанию, то их поменять местами.
        если  A[j] < A[j-1]
     тоОбмен(j, j-1);
    конец цикла
конец цикла
```

Анализ

Количество сравнений C_i на i – м проходе равно $N - i$, что приводит к уже известному выражению для C:

$$C = (N - 1) + (N - 2) + \ ... \ + 1 = N \cdot (N - 1)/2$$

Минимальное количество пересылок $M_{min} = 0$, если массив уже упорядочен,

максимальное $M_{max} = C$, если массив упорядочен по убыванию.

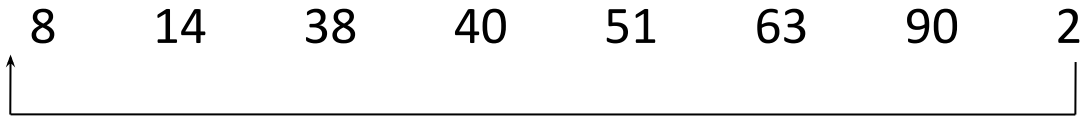
Улучшение метода пузырька

1) Нередко случается, что последние проходы сортировки простым обменом работают «вхолостую», так как элементы уже упорядочены.

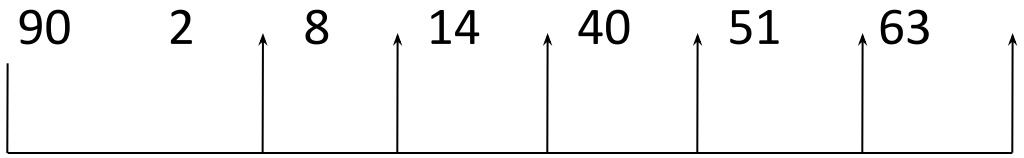
Один из способов улучшения алгоритма сортировки пузырьком состоит в том, чтобы запомнить, производился ли на очередном проходе какой-либо обмен.

Если ни одного обмена не было, то алгоритм может закончить работу.

2) Асимметрия метода: один неправильно расположенный «пузырек» на «тяжелом» конце почти отсортированного массива «всплывет» на место за один проход:



Неправильно расположенный «камень» на «легком» конце будет «опускаться» на правильное место только только по одному шажку на каждом проходе:



Шейкер-сортировка (алгоритм)

Пусть F — логическая переменная, принимающая истинное значение, если во время прохода по массиву были обмены двух рядом стоящих элементов, *left* – левая граница несортированной части массива, а right – ее правая граница.

```
left ← 1; right ← N;
F ← истина;
пока F выполнять
     F ← ложь;
     i ← left;
     //Проход по массиву от начала к концу:
     пока i < right выполнять
        если A[i] > A[i + 1] то
   // переставить два рядом стоящих элемента, нарушающие порядок:
            начало
           Обмен (i, i+1);
           F ← истина;
            конец
        i ← i + 1;
     конец пока
     // Сдвинуть правую границу влево на одну позицию :
     right ← right – 1;
```

Шейкер-сортировка (продолжение алгоритма)

```
// Если были обмены во время предыдущего прохода,
если F то  // совершить проход по массиву от конца к началу:
  начало
     F ← ложь;
     i ← right;
     пока i > left выполнять
        если A[i] < A[i – 1] то
        // переставить рядом стоящие элементы,  нарушающие
порядок:
            начало
               Обмен (i, i–1);
               F ← истина;
            конец
        i ← i – 1;
      конец пока
   конец
   // Сдвинуть левую границу вправо на одну позицию:
   left ← left + 1;
конец пока   // Цикл повторять до тех пор, пока F не
          //останется равной значению ложь.
```

Анализ

$C_{min} = N - 1$.

Кнут показал, что среднее число сравнений пропорционально

$$N^2 - N.$$

Но все предложенные улучшения не влияют на число обменов.

В самом деле, каждый обмен уменьшает число инверсий в массиве на 1, следовательно, при любом алгоритме, основанном на обмене пар соседних элементов, число необходимых перестановок одинаково и равно числу инверсий в массиве.

Сортировка обменом и ее улучшенная сортировка хуже, чем сортировка включениями и выбором.

Шейкер-сортировку выгодно использовать тогда, когда массив почти упорядочен.

Перестановки

Перестановкой порядка N называется расположение N различных объектов в ряд в некотором порядке.

Например, для трех объектов — a, b и c — существует шесть перестановок:

abc, acb, bac, bca. cab, cba.

Для множества из N элементов можно построить $N!$ различных перестановок: первую позицию можно занять N способами, вторую — (N – 1) способом, так как один элемент уже занят, и т. д. На последнее место можно поставить только один оставшийся элемент.

Следовательно, общее количество вариантов расстановки равно

$$N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot 1 = N!$$

Далее будем рассматривать перестановки элементов множества

{1, 2, 3, … , N}

Инверсии

Пусть даны базовое множество из N элементов 1,2, 3,..., N и его перестановка

$$a_1, a_2, ..., a_{N-1}, a_N$$

Пара (a_i, a_j) называется *инверсией* (*инверсионной парой*) перестановки, если $i < j$ и при $a_i > a_j$.

Например, перестановка 4, 1, 3, 2 имеет четыре инверсии:
(4, 1), (3, 2), (4, 3) и (4, 2).

Единственной перестановкой, не содержащей инверсий, является упорядоченная перестановка 1, 2, 3, ... , N.

Таким образом, каждая инверсия — это пара элементов перестановки, нарушающих ее упорядоченность.

Таблицей инверсий перестановки $a_1, a_2, \ldots, a_N$ называется последовательность чисел $b_1, b_2, \ldots, b_N$, где b_j есть число элементов перестановки, больших j и расположенных левее j (т. е. количество инверсий вида (x, j), при $x > j$).

Например,
для перестановки **5 9 1 8 2 6 4 7 3**
таблица инверсий: **2 3 6 4 0 2 2 1 0**.

Свойство элементов таблицы инверсий:

$$b_N = 0,$$
$$\ldots$$
$$0 \le b_i \le N - i,$$
$$\ldots$$
$$0 \le b_1 \le N - 1.$$

Утверждение

Таблица инверсий единственным образом определяет соответствующую ей перестановку.

Построение перестановки по таблице инверсий

1 способ: проход по таблице инверсий справа налево

Создается заготовка перестановки из одного максимального числа. На каждом шаге в нее вставляется следующий по величине элемент с учетом того, сколько элементов, больших него, должно стоять перед ним.

Пример:

Таблица инверсий: 2 3 6 4 0 2 2 1 0

1. 9
2. 9 8
3. 9 8 7
4. 9 8 6 7
5. 5 9 8 6 7
6. 5 9 8 6 4 7
7. 5 9 8 6 4 7 3
8. 5 9 8 2 6 4 7 3
9. 5 9 1 8 2 6 4 7 3

Алгоритм П1:

построение перестановки по таблице инверсий справа налево

Вход:

$N > 0$ - количество элементов перестановки,

$b_1, b_2 \ldots, b_N$ – таблица инверсий,

$0 \le b_j \le N - j$.

$P \leftarrow$ пустая последовательность;

цикл по j от N вниз до 1

вставить число j в P после b_j элементов;

конец цикла

Выход:

P − перестановка, соответствующая данной таблице инверсий

Построение перестановки по таблице инверсий

2 способ: проход по таблице инверсий слева направо

Создается заготовка пустой перестановки длины N.

На каждом шаге для каждого элемента перестановки, начиная с 1, отсчитывается в ней столько пустых ячеек, какое число записано в соответствующей позиции в таблице инверсий. В следующее за ними пустое место вставляется этот элемент.

Пример:

Таблица инверсий: **2 3 6 4 0 2 2 1 0**

Перестановка:

5	9	1	8	2	6	4	7	3

Алгоритм П2:

построение перестановки по таблице инверсий слева направо

Вход:

$N > 0$ - количество элементов перестановки,

$b_1, b_2 \ldots, b_N$ – таблица инверсий,

$0 \le b_j \le N - j$.

$P \leftarrow$ последовательность из N пустых элементов;

цикл по i **от** 1 **до** N **с шагом** 1 **выполнять**

пропустить b_i *пустых мест в* P;

поместить i *на следующее пустое место;*

конец цикла

Выход:

P – перестановка, соответствующая данной таблице инверсий

Инверсионный метод поиска всех перестановок

Таблица инверсий однозначно определяет перестановку и каждая перестановка имеет только одну таблицу инверсий.

Следовательно, если мы сумеем перебрать все таблицы инверсий, то с помощью алгоритмов П1 или П2 сможем по ним восстановить все перестановки.

Рассмотрим таблицу инверсий как *N*-значное число в такой необычной «системе счисления»: количество цифр, которое можно использовать в *i*-м разряде (с конца, начиная с 0) равно *i*.

Возьмем «минимальную» таблицу и будем последовательно прибавлять к ней, как к числу, единицу, пользуясь, например, алгоритмом сложения с переносом для многоразрядных чисел, модифицированным для нашей «системы счисления».

Генерация таблиц инверсии

4	3	2	1	0	
0	0	0	0	0	Шаг 0
0	0	0	1	0	Шаг 1
0	0	1	0	0	Шаг 2
0	0	1	1	0	Шаг 3
0	0	2	0	0	Шаг 4
0	0	2	1	0	Шаг 5
0	1	0	0	0	Шаг 6
0	1	0	1	0	Шаг 7
0	1	1	0	0	Шаг 8
0	1	1	1	0	Шаг 9
0	1	2	0	0	Шаг 10
…	…	…	…	…	…
4	3	2	1	0	Шаг 119

Алгоритм И1:
нахождение следующей таблицы инверсий

Пусть B = b_1, b_2, ..., b_N – таблица инверсий, построенная на предыдущем шаге.

Тогда следующая таблица инверсий получается из нее прибавлением к ней единицы:

```
i ← N;
flag ←  истина;
пока flag выполнять
     x ← b_i + 1;
    если x > N – i
    то
       начало
     b_i ← 0;
     i ←  i –1;
        конец
    иначе
       начало
     b_i ← x;
     flag ← ложь;
        конец
конец пока
```

Алгоритм Дейкстры: поиск следующей по алфавиту перестановки

Пусть даны две перестановки

$b = b_1, b_2 \ldots, b_N$ и

$c = c_1, c_2 \ldots, c_N$

набора 1, 2, ..., N.

Говорят, что перестановка b предшествует перестановке c в алфавитном (лексикографическом) порядке, если для минимального значения k, такого что $b_k \neq c_k$, справедливо $b_k < c_k$.

Например, перестановка **1 2 3 4 5** предшествует перестановке **1 2 4 5 3** (здесь $k = 3$).

Первой перестановкой в алфавитном порядке является

перестановка **1,2,3, ..., *N***,

а последней — ***N*,*N*-1,N-2,...,1**

Идея алгоритма Дейкстры:

определить каким-либо образом функцию, которая по заданной перестановке выдает непосредственно следующую за ней в алфавитном порядке, и применять ее последовательно к собственным результатам начиная с самой первой перестановки, пока не будет получена последняя.

Например, для перестановки

1 4 6 2 9 5 8 7 3

следующей по алфавиту является перестановка

1 4 6 2 9 7 3 5 8.

Алгоритм Дейкстры:
генерация следующей по алфавиту перестановки

Вход: $N > 0$ — количество элементов;
$a_1, a_2, \ldots, a_{N-1}, a_N$ – предыдущая перестановка.

Шаг 1. Просматривая перестановку, начиная с последнего элемента, найдем такой номер i, что $a_{i+1} > \ldots > a_N$ и $a_i < a_{i+1}$. Если такого i нет, то последовательность упорядочена по убыванию и следующей перестановки нет: конец алгоритма.

Шаг 2. Найти в «хвосте» $a_{i+1}, \ldots, a_N$ элемент a_j, такой что $i+1 \le j \le N$, a_j есть наименьшее значение, удовлетворяющее условию $a_j > a_i$. После этого поменять местами a_i и a_j.

Шаг 3. Упорядочить «хвост» $a_{i+1}, \ldots, a_N$ по возрастанию. Для этого достаточно его инвертировать (обернуть в обратном порядке).

Выход: следующая по алфавиту перестановка за данной.

Пример построения следующей по алфавиту перестановки

Для перестановки

1 4 6 2 9 5 8 7 3

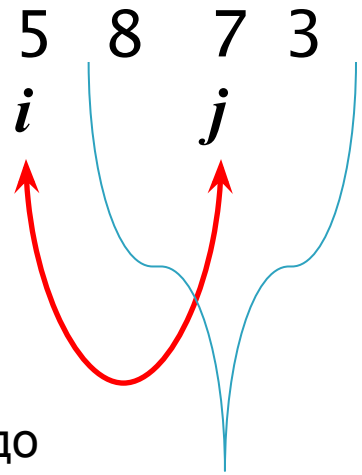
Найти следующую по алфавиту.

Шаг 1: Найти номер *i* элемента от конца, нарушающего порядок по возрастанию

Шаг 2:

Найти элемент из хвоста с номером *j* – минимальный, больший *i*-го

Поменять их местами



Шаг 3:

Перестроить хвост от (i+1)-го элемента до конца по возрастанию (обернуть)