Лекция 10

Простые сортировки

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

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

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;
- выбора;
- обмена;
- подсчета;
- разделения;
- слияния.

Сортировка включением (вставками)

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

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

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 есть

$$C = (N - 1) + (N - 2) + \ldots + 1 = N * (N - 1)/2$$

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.

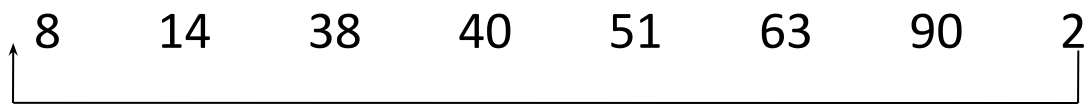
Улучшение метода пузырька

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

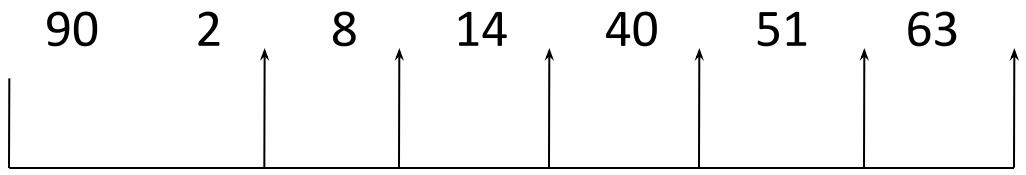
Один из способов улучшения алгоритма сортировки пузырьком состоит в том, чтобы запомнить, производился ли на очередном проходе какой-либо обмен.

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.

Сортировка методом подсчета

При сортировке подсчетом каждый элемент поочередно сравнивается со всеми остальными и подсчитывается количество элементов, которые меньше его. Это число (+1) определяет позицию элемента в отсортированной последовательности при условии, что все элементы различны.

Простейшая реализация этого метода требует дополнительного массива, в котором накапливаются отсортированные элементы. Это связано с тем, что в данном методе входная последовательность не сокращается по мере обработки элементов.

Алгоритм (на одном массиве)

Условно разделить массив A на отсортированную и несортированную части. Пусть i – номер первого элемента в несортированной части массива.

```
i := 1;
пока i < N выполнять
    r:= 1;
    // Посчитать количество элементов в массиве, меньших
    // i-го элемента и записать это число в переменную r
    цикл по j от 1 до N с шагом 1 выполнять
        если A[i] > A[j]
        то r := r + 1;
    конец цикла
    если r ≤ i      // i-й элемент стоит на своем месте,
    то i := i + 1 // увеличить сортированную часть на 1 элемент
    иначе
        начало
        // вычислить позицию, куда нужно поставить i-й элемент:
            пока A[r] = A[i] выполнять r:= r+1
            конец пока
        // поменять его местами с тем элементом,
    // который в этой позиции находится:
            Обмен (r, i)
```