ЭЛЕМЕНТАРНЫЕ АЛГОРИТМЫ СОРТИРОВКИ

Специально для кружка по олимпиадному программированию

by Mellanore

Аннотация

В представленном PDF-документе бегло и без особой строгости рассматриваются примитивные алгоритмы сортировки. На дальнейших занятиях будет предполагаться, что каждый из присутствующих знает принципы работы каждого и может без труда написать их C++ реализацию. Автору этого документа будет приятно, если кто-либо из прочитавших попытается реализовать сортировку Шелла (не подсматривая на готовые примеры в интернете) и запостит результат сего действа в группу.

Содержание

Задача сортировки

Вход: последовательность из n чисел $\{a_1, a_2, \ldots, a_n\}$
Выход: перестановка $\{a'_1, a'_2, \ldots, a'_n\}$ исходной последовательности таким образом, что для ее членов выполняется соотношение $\{a'_1 \leq a'_2 \leq \ldots \leq a'_n\}$

Мы будем рассматривать только *Внутренние сортировки*, когда число записей подлежащих сортировке достаточно мало, так, что весь процесс можно провести в оперативной памяти компьютера.

Ключем элемента называется его значение.

1 Сортировка вставками (Insertion sort)

Идея алгоритма:

Элементы последовательности просматриваются по одному, и каждый новый элемент вставляется в подходящее место среди ранее упорядоченных элементов. (Таким спсобом, в частности, игроки в бридж упорядочивают свои карты, беря по одной)

Преимущества алгоритма:

1. Эффективен на небольших наборах данных;
2. На наборах данных до десятков элементов может оказаться лучшим;
3. Эффективен на наборах данных, которые уже частично отсортированы;
4. Может сортировать список по мере его получения;
5. Использует O(1) временной памяти

Недостатки алгоритма:

1. При каждой вставке производится сдвиг массива, следовательно даже при относительно малом числе сравнений число перестановок может быть довольно значительным.
2. Временная сложность алгоритма в худшем случае $\theta(n^2)$

Порядок работы алгоритма:

1. Выберем в данной нам входной последовательности подпоследовательность длины 1, состоящую из первого ее элемента. Очевидно, что эта подпоследовательность является отсортированной.
2. Начиная с 2 элемента и до последнего элемента входной последовательности будем искать место для этого элемента в уже отсортированной подпоследовательности так, чтобы подпоследовательность оставалась отсортированной
3. Вставляем этот элемент на найденное место и увеличиваем размер подпоследовательности на 1

Алгоритм на C++:

```
#include <cstdio>
#include <iostream.h>

int main() {
    int key,n,i,j;
    cin >> n;
    int a[n];

    for(i = 0; i < n; i++) {
        cin >> a[i];
    }

    for(i = 1; i < n; i++) {
        key = a[i];
        j = i - 1;
        while((j >= 0) && (a[j] > key)) {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = key;
    }

    for(i = 0; i < n-1; i++) {
        cout << a[i] << ' ';
    }
    cout << a[n-1];

    return 0;
}
```

2 Сортировка выбором (Selection sort)

Идея алгоритма:

Сначала выделяется наименьший (наибольший) элемент последовательности и каким либо образом выделяется от остальных, затем выделяется наименьший(наибольший) из оставшихся.

Преимущества алгоритма:

1. Наибольшее число обменов равно $N - 1$, где N - это число элементов в последовательности
2. Использует O(1) временной памяти

Недостатки алгоритма:

1. Временная сложность алгоритма в среднем, наилучшем и наихудшем случае $\theta(n^2)$

Порядок работы алгоритма:

1. Находим номер минимального значения во входной последовательности;
2. Производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции);
3. Сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы.

Алгоритм на C++:

```
#include <cstdio>
#include <iostream.h>

int main() {
    int min,tmp,n,i,j;
    cin >> n;
    int a[n];

    for(i = 0; i < n; i++) {
        cin >> a[i];
    }

    for(i = 0; i < n-1; i++) {
        min = i;
        for(j = i+1; j < n; j++) {
            if(a[j] < a[min]) {
                min = j;
            }
        }
        if(min != i) {
            tmp = a[min];
            a[min] = a[i];
            a[i] = tmp;
        }
    }

    for(i = 0; i < n-1; i++) {
        cout << a[i] << ' ';
    }
    cout << a[n-1];

    return 0;
}
```

3 Сортировка пузырьком (Bubble sort)

Идея алгоритма:

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим наибольшим элементом, а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции как пузырёк в воде, отсюда и название алгоритма).

Преимущества алгоритма:

1. Использует O(1) временной памяти
2. Очень прост в реализации

Недостатки алгоритма:

1. Временная сложность алгоритма $\theta(n^2)$

Порядок работы алгоритма:

1. Начиная с 1 и 2 элементов попарно упорядочиваем их и по мере прохождения по последовательности (упорядочивая 1-й и 2-й, затем, 2-й и 3-й, 3-й и 4-й и т.д элементы), очевидно, мы получим наибольший (наименьший) элемент на последнем месте последовательности.
2. Исключая из прохода последние (уже окончательно отсортированные) элементы, повторяем шаг 1, пока вся последовательность не станет отсортированной

Алгоритм на C++:

```
#include <cstdio>
#include <iostream.h>

int main() {
    int tmp,n,i,j;
    cin >> n;
    int a[n];

    for(i = 0; i < n; i++) {
        cin >> a[i];
    }

    for(i = 0; i < n; i++) {
        for(j = i; j <= n-i; j++) {
            if(a[i] > a[j]) {
                tmp = a[i];
                a[i] = a[j];
                a[j] = tmp;
            }
        }
    }

    for(i = 0; i < n-1; i++) {
        cout << a[i] << ' ';
    }
    cout << a[n-1];

    return 0;
}
```

4 Сортировка Шелла (Shell sort)

Идея алгоритма:

Сортировка Шелла это алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. Иными словами — это сортировка вставками с предварительными «грубыми» проходами.
При сортировке Шелла сначала сравниваются и сортируются между собой значения, отстоящие один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при d=1 (то есть обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места.

Упражнение:

Читателю предлагается самостоятельно попытаться улучшить алгоритм сортировки вставками до сортировки Шелла (Выбрав за d последовательность длин промежутков $d_1 = N/2, d_i = d_{i-1}/2, d_k = 1$)

За дополнительной информацией можно обратиться к:
`http://ru.wikipedia.org/wiki/Shellsort`