Сортировка

Введение

Наиболее частыми операциями при работе с данными являются «поиск» и «сортировка». При этом алгоритмы решения этих задач существенно зависят

Метод сортировки — это алгоритм для упорядочивания элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

Если мы имеем **числовой массив**, то **ключ элемента** – это сам элемент. В этом случае сортировка массива – это упорядочивание массива (перестановка его элементов) таким образом, чтобы получилась неубывающая или невозрастающая последовательность.

Если каждый элемент исходного массива представляет собой слова, составленные из букв русского алфавита, то ключ, по которому может быть упорядочен массив, связывается с порядковым номером буквы в алфавите. По этому принципу упорядочиваются слова в словарях – из двух слов первым помещается то слово, ключ которого меньше. Здесь принимается, что отсутствие буквы (т. е. пустая строка) имеет меньший ключ, чем любая другая буква. Так слово "студент" помещается в словаре перед словом "студентка".

Введение

Если слова состоят из букв разных алфавитов и цифр, как, например, имена файлов и папок, то любая цифра имеет меньший ключ, чем любая буква, а любая латинская буква имеет меньший ключ по сравнению с любой русской буквой. При сортировке таких имен в качестве ключа используется **код символа** в некоторой **кодовой таблице**. Этот принцип упорядочивания еще называют **лексикографическим порядком** или **расширенным алфавитом**.

Разработкой различных алгоритмов сортировки информации, хранящейся в оперативной памяти компьютера или на его жестком диске, программисты занимаются уже давно. Интерес к этой проблеме обусловлен тем, что по мнению специалистов **25% всего времени** обработки информации расходуется на сортировку данных.

Термины

Если элементы массива связаны отношениями $a_0 < a_1 < \ldots < a_{n-1}$, то говорят, что массив упорядочен по **возрастанию**. Такая упорядоченность предполагает, что в массиве нет одинаковых элементов.

Если элементы массива связаны отношениями $a_0 \le a_1 \le \ldots \le a_{n-1}$, то говорят, что массив упорядочен по **неубыванию**. Такая упорядоченность не исключает наличие в массиве одинаковых элементов.

Если элементы массива связаны отношениями $a_0 > a_1 > \ldots > a_{n-1}$, то говорят, что массив упорядочен по **убыванию**. Такая упорядоченность предполагает, что в массиве нет одинаковых элементов.

Если элементы массива связаны отношениями $a_0 \ge a_1 \ge \ldots \ge a_{n-1}$, то говорят, что массив упорядочен по **невозрастанию**. Такая упорядоченность не исключает наличие в массиве одинаковых элементов.

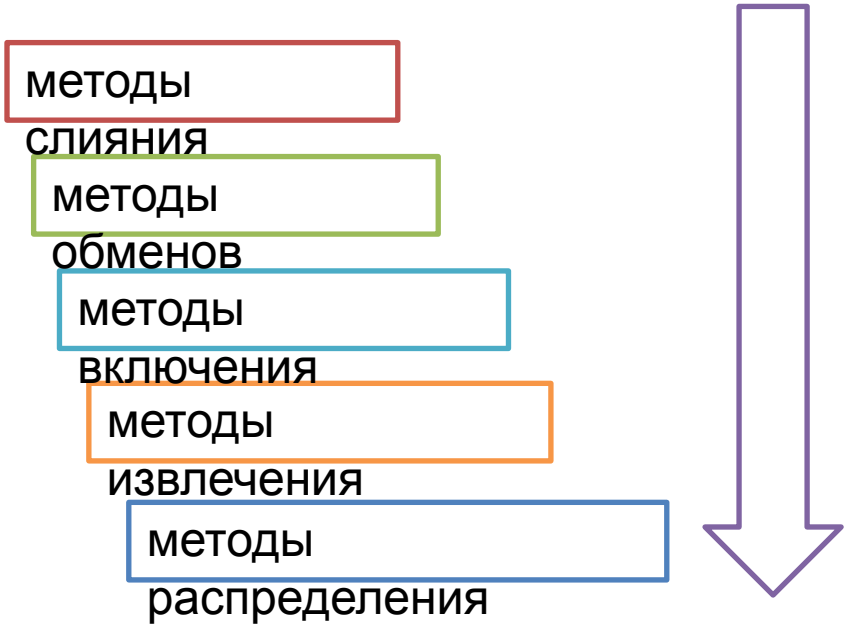
Классификация

Методы сортировки классифицируются по времени их работы. Хорошей мерой эффективности может быть число сравнений ключей - *C* и число пересылок элементов - *P*. Эти числа являются функциями *C(n), P(n)* от числа сортируемых элементов *n*. **Быстрые** (но сложные) алгоритмы сортировки требуют (при $n \to \infty$) порядка ***n log n*** сравнений, **прямые** (простые) методы - $\mathbf{n^2}$.

Прямые методы коротки, просто программируются, быстрые, усложненные, методы требуют меньшего числа операций, но эти операции обычно сами более сложны, чем операции прямых методов, поэтому для достаточно малых *n* $(n \leq 50)$ прямые методы работают быстрее. Значительное преимущество быстрых методов (в *n/log(n)* раз) начинает проявляться при $n \geq 100$.

Классификация

Все методы сортировки можно разделить на **пять групп**



Концепция методов

Общая концепция **методов извлечения** заключается в следующем: из исходного массива извлекается минимальный элемент и меняется местами с первым элементом массива, затем извлекается минимальный элемент из части массива, начиная со второго элемента, и меняется местами со вторым элементом и т. д. Последний раз минимальный элемент выбирается из двух последних элементов массива. В результате получится массив, упорядоченный по **неубыванию**.

Различные методы извлечения отличаются объектом извлечения (минимальный или максимальный элемент) и, соответственно, объектами перестановки (первый или последний элемент), а также условием окончания процесса сортировки.

Концепция методов

Идея **методов включения** состоит в том, что сначала первый элемент массива рассматривается как упорядоченный массив и в этот массив включается следующий элемент исходного массива так, чтобы получился упорядоченный по неубыванию массив из двух элементов. Затем в полученный упорядоченный массив включается третий элемент массива так, чтобы опять-таки получился упорядоченный массив. Процесс продолжается до тех пор, пока не будет включен последний элемент.
Различные алгоритмы включения отличаются способами выбора элемента для включения, способами определения места включения и методами самого включения.

Концепция методов

Идея **методов обменов** состоит в следующем: в исходном массиве выбирается пара элементов, и они сравниваются между собой. Если их положение не удовлетворяет требованию упорядоченности, то элементы переставляются. Затем выбирается следующая пара элементов и так до тех пор, пока не получим упорядоченный массив.
Различные алгоритмы обменов отличаются способами выбора пары элементов для сравнения и перестановки, а также условиями окончания процесса сортировки.

Метод слияния применяется в том случае, когда имеются два (или больше) упорядоченных массива и требуется соединить исходные массивы в один общий упорядоченный массив.

Метод распределения употребим в тех случаях, когда в исходном массиве имеется заданное, известное заранее, количество различных ключей (значений). Например, имеется список студентов с оценками по пятибалльной системе, полученными на экзамене. Нам известно заранее, что оценки могут быть 5, 4, 3 и 2. Поэтому для упорядочения массива по невозрастанию можно сначала выбрать все записи с оценками 5, затем с оценками 4, потом с оценками 3 и, наконец, с оценками 2.

Схемы работы методов

Сортировка пузырьком

Сортировка пузырьком – простейший алгоритм сортировки, применяемый чисто для учебных целей. Практического применения этому алгоритму нет, так как он не эффективен, особенно если необходимо отсортировать массив большого размера. К плюсам сортировки пузырьком относится простота реализации алгоритма.

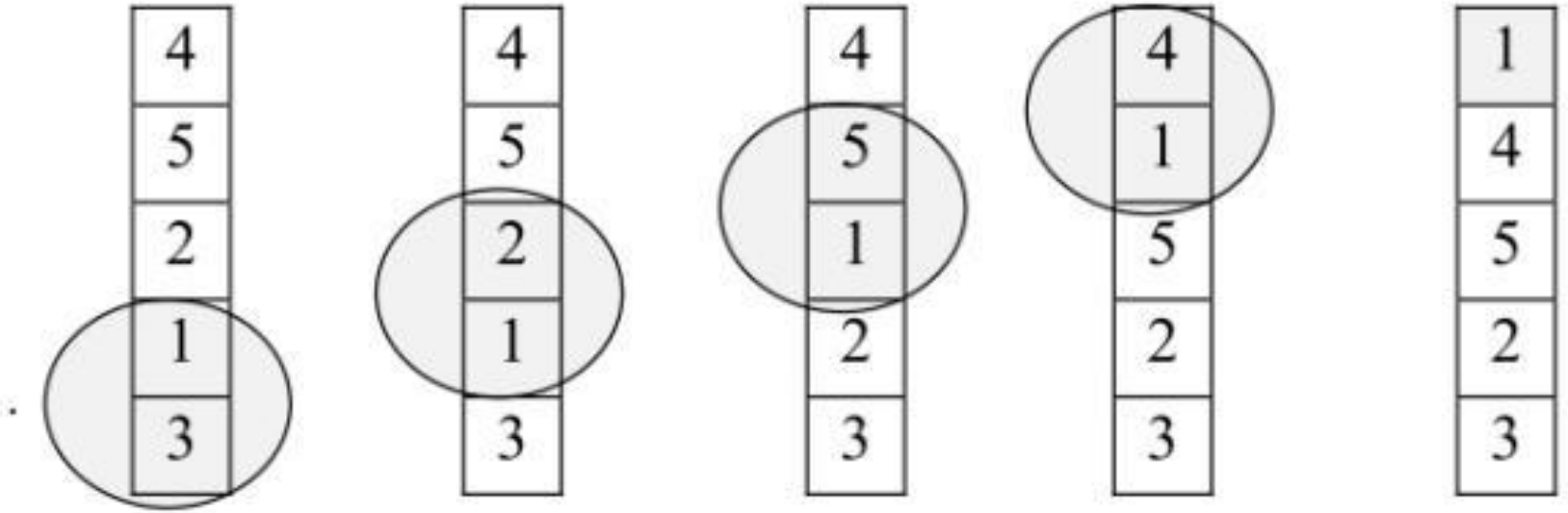
Алгоритм сортировки пузырьком сводится к повторению проходов по элементам сортируемого массива. Проход по элементам массива выполняет внутренний цикл. За каждый проход сравниваются два соседних элемента, и если порядок неверный элементы меняются местами. Внешний цикл будет работать до тех пор, пока массив не будет отсортирован.

Таким образом, внешний цикл контролирует количество срабатываний внутреннего цикла. Когда при очередном проходе по элементам массива не будет совершено ни одной перестановки, то массив будет считаться отсортированным.

Схемы работы методов

Сортировка пузырьком

Сортировка пузырьком. Название этого метода произошло от известного физического явления - пузырек воздуха в воде поднимается вверх. В этом методе сначала поднимается "наверх" (к началу массива) самый "легкий" элемент (минимальный), затем следующий и т.д.



Схемы работы методов

Сортировка пузырьком

№ итерации	Элементы массива							Перестановки
исх. массив	3	3	7	1	2	5	0	
0	3	3						false
1		3	7					false
2			**1**	**7**				7>1, true
3				**2**	**7**			7>2, true
4					**5**	**7**		7>5, true
5						**0**	**7**	7>0, true

Схемы работы методов

Сортировка пузырьком

тек.массив	**3**	**3**	**1**	**2**	**5**	**0**	**7**	
0	3	3						false
1		**1**	**3**					3>1, true
2			**2**	**3**				3>2, true
3				**0**	**3**			3>0, true
4					3	5		false
5						5	7	false

Схемы работы методов

Сортировка пузырьком

тек.массив	**3**	**1**	**2**	**0**	**3**	**5**	**7**	
0	**1**	**3**						3>1, true
1		**2**	**3**					3>2, true
2			**0**	**3**				3>0, true
3				3	3			false
4					3	5		false
5						5	7	false

Схемы работы методов

Сортировка пузырьком

тек.массив	**1**	**2**	**0**	**3**	**3**	**5**	**7**	
	1	2						false
		0	**2**					2>0, true
			2	3				false
				3	3			false
					3	5		false
						5	7	false

Схемы работы методов

Сортировка пузырьком

тек.массив	**1**	**0**	**2**	**3**	**3**	**5**	**7**	
	0	**1**						1>0, true
		1	2					false
			2	3				false
				3	3			false
					3	5		false
						5	7	false
конечный массив	**0**	**1**	**2**	**3**	**3**	**5**	**7**	
Конец сортировки								

Схемы работы методов

Сортировка пузырьком

```
#include < stdio.h >
const N = 10;
void main()
{
int i, j, A[N], c;
// ввод массива A
        for ( i = 0; i < N-1; i ++ )
                for ( j = N-2; j >= i; j -- )
                        if ( A[j] > A[j+1] )
                        {
                                c = A[j]; A[j] = A[j+1];
                                A[j+1] = c;
                        }
        printf("\n Отсортированный массив:\n”);
        for ( i = 0; i < N; i ++ )
                printf("%d ", A[i]);
}
```

Схемы работы методов

Сортировка пузырьком

Для того чтобы отсортировать массив хватило пяти запусков внутреннего цикла **for**. Запустившись, цикл **for** срабатывал **6 раз**, так как элементов в массиве **7**, то итераций (повторений) цикла **for** должно быть на одно меньше. На каждой итерации сравниваются два соседних элемента массива. Если текущий элемент массива больше следующего, то меняем их местами. Таким
образом, пока массив не будет отсортирован, будет запускаться внутренний цикл и выполняться операция сравнения. Обратите внимание на то, что за каждое полное выполнение цикла **for** как минимум один элемент массива находит своё место. В худшем случае, понадобится **n-2** запуска внутреннего цикла, где **n** – количество элементов массива. Это говорит о том, что сортировка пузырьком крайне неэффективна для больших массивов.

Схемы работы методов

Сортировка выбором

Пожалуй, самый простой алгоритм сортировок – это сортировка выбором. Судя по названию сортировки, необходимо что-то выбирать (максимальный или минимальный элементы массива). Алгоритм сортировки выбором находит в исходном массиве максимальный или минимальный элементы, в зависимости от того как необходимо сортировать массив, по возрастанию или по убыванию. Если массив должен быть отсортирован по возрастанию, то из исходного массива необходимо выбирать минимальные элементы. Если
же массив необходимо отсортировать по убыванию, то выбирать следует максимальные элементы.

Допустим необходимо отсортировать массив по возрастанию. В исходном массиве находим минимальный элемент, меняем его местами с первым элементом массива. Уже, из всех элементов массива один элемент стоит на своём месте. Теперь будем рассматривать не отсортированную часть массива, то есть все элементы массива, кроме первого. В неотсортированной части массива опять ищем минимальный элемент. Найденный минимальный элемент меняем местами со вторым элементом массива и т. д.

Схемы работы методов

Сортировка выбором

исходный массив: 3 3 7 1 2 5 0

1)Итак, находим минимальный элемент в массиве. 0 – минимальный элемент

2)Меняем местами минимальный и первый элементы массива.

Текущий массив: 0 3 7 1 2 5 3

3) Находим минимальный элемент в неотсортированной части массива. 1 минимальный элемент

4) Меняем местами минимальный и первый элементы массива.

Текущий массив: 0 1 7 3 2 5 3

5) min = 2

6) Текущий массив: 0 1 2 3 7 5 3

7)min = 3

8) Текущий массив: 0 1 2 3 7 5 3 в массиве ничего не поменялось, так как 3 стоит на своём месте

9) min = 3

10) **Конечный вид массива: 0 1 2 3 3 5 7** – массив отсортирован.

Схемы работы методов

Сортировка выбором

```
#include <stdio.h>
const N = 10;
void main()
{
        int i, j, nMin, A[N], c;
// ввод массива A
        for ( i = 0; i < N-1; i ++ )
        {
                nMin = A[i];
                for ( j = i+1; j < N-1; j ++ )
                        if ( A[j] < A[nMin] )
                        nMin = j;
                if ( nMin != i )
                {
                         c = A[i]; A[i] = A[nMin];
                        A[nMin] = c;
                 }
        }
printf("\n Отсортированный массив:\n”);
for ( i = 0; i < N; i ++ )
printf("%d ", A[i]);
}
```

Схемы работы методов

Сортировка вставками

Сортировка вставками — достаточно простой алгоритм. Как в и любом другом алгоритме сортировки, с увеличением размера сортируемого массива увеличивается и время сортировки. Основным преимуществом алгоритма
сортировки вставками является возможность сортировать массив по мере его получения. То есть имея часть массива, можно начинать его

Сортируемый массив можно разделить на две части — **отсортированная** часть и **неотсортированная**. В начале сортировки первый элемент массива
считается отсортированным, все остальные — не отсортированные. Начиная
со второго элемента массива и заканчивая последним, алгоритм вставляет
неотсортированный элемент массива в нужную позицию в отсортированной
части массива. Таким образом, за один шаг сортировки отсортированная часть массива увеличивается на один элемент, а неотсортированная часть

Схемы работы методов

Сортировка вставками

исходный массив: 3 3 7 1 2 5 0

шаг	отсортированная часть массива							тек.элемент	вставка
1	3							3	false
2	3	3						7	false
3	3	3	7					1	true
4	1	3	3	7				2	true
5	1	2	3	3	7			5	true
6	1	2	3	3	5	7		0	true
-	0	1	2	3	3	5	7	-	-

Схемы работы методов

Сортировка вставками

На каждом шаге сортировки сравнивается текущий элемент со всеми элементами в отсортированной части. На первом шаге сравнивается тройка с тройкой, они равны поэтому не меняем их местами. На втором шаге сравниваем 7 с двумя тройками, 7 >3 а так как сортировка по возрастанию, то опять элементы массива остаются на своих местах. На третьем шаге единица сравнивается с тремя элементами и все они больше единицы, значит единицу вставляем на первое место, в начало массива. На четвёртом шаге текущий элемент — 2 сравниваем с элементами 1, 3, 3, 7. Получается, что 1 < 2 < 3 и 7 поэтому двойку вставляем между единицей и тройкой. Пятый и шестой шаги выполняются точно также. В итоге на шестом шагу мы получаем отсортированный по возрастанию массив.

Схемы работы методов

Сортировка вставками

```
void InsertionSort(int n, int mass[])
{
   int newElement, location;

   for (int i = 1; i < n; i++)
   {
      newElement = mass[i];
      location = i - 1;
      while(location >= 0 && mass[location] > newElement)
      {
         mass[location+1] = mass[location];
         location = location - 1;
      }
      mass[location+1] = newElement;
   }
}
```

Схемы работы методов

Сортировка Шелла

В 1959 году Дональд Шелл опубликовал усовершенствованный алгоритм сортировки вставками, который впоследствии получил его имя. Идея данного метода заключается в сравнение разделенных на группы элементов некоторой последовательности.

Сначала, сравниваются элементы находящиеся друг от друга на расстоянии d (первое d обычно равно n/2, где n — количество всех элементов). Постепенно расстояние между элементами уменьшается, и на d=1 сортировка происходит последний раз.

Схемы работы методов

Сортировка Шелла

```
/* shellsort: сортируются v[0] ... v[n-1] в возрастающем порядке */
void shellsort (int v[], int n)
{
   int gap, i, j, temp;
   for (gap = n/2; gap > 0; gap /= 2)
      for (i = gap; i < n; i++)
         for (j = i - gap; j >= 0 && v[j] > v[j + gap]; j -= gap) {
            temp = v[j];
            v[j] = v[j + gap];
            v[j + gap] = temp;
         }
}
```

Схемы работы методов

Метод подсчета

Идея метода заключается в следующем: в отсортированной последовательности, элемент, занимающий позицию с номером **K+1**, превышает ровно **K** элементов, поэтому в процессе сортировки методом подсчета на каждом **i**-ом проходе мы попарно сравниваем **i**-й элемент со всеми элементами массива. Если установлено, что **mass[i] > mass[j]**, то увеличиваем счетчик **K** на единицу (в начале K = 0). По окончании текущего прохода счетчик **K** указывает количество элементов, меньших, чем **mass[i]**, поэтому элемент **mass[i]** занимает в отсортированной последовательности позицию **K + 1** (sortedMass[k + 1] = mass[i]).

Внимание! Рассматриваемый алгоритм можно использовать только для последовательностей, которые не содержат одинаковых элементов!

Схемы работы методов

Метод подсчета

```
//сортировка методом подсчета
void methodOfCalculation(int n, int mass[], int sortedMass[])
{
   int k;
   for (int i = 0; i < n; i++)
   {
      k = 0;
      for (int j = 0; j < n; j++)
      {
         if (mass[i] > mass[j])
            k++;
      }
      sortedMass[k] = mass[i];
   }
}
```