Сортировка
Базовые алгоритмы

Сортировка

Под сортировкой понимают процесс перестановки объектов в определённом порядке. Сортировка – одна из самых частых операций, причём не только в программировании, но и в повседневной жизни: объекты сортируются в телефонных справочниках, в каталогах, в словарях, даже дома на улицах отсортированы по номерам. Основная цель сортировки – облегчить поиск определённого элемента. Очевидно, что сортировка – весьма важная операция. Важность этой операции привела к появлению множества различных алгоритмов, решающих одну и ту же задачу. При этом алгоритмы сортировки отличаются друг от друга требованиями к памяти, временем работы, способом доступа к данным и многими другими параметрами.

Прежде чем перейти непосредственно к изучению алгоритмов, надо договориться о некоторых базовых определениях и понятиях. Будем считать, что сортируемые элементы состоят из двух частей: непосредственно данных и ключа. Для алгоритмов сортировки важен только ключ, по которому выполняется сортировка. Например, у нас есть следующая информация о студентах: фамилия, имя, год рождения. Если необходимо упорядочить студентов по алфавиту, ключом будет выступать фамилия, если же надо отсортировать студентов по возрасту, ключом будет год рождения.

Структура и способ хранения обрабатываемых данных всегда сильно влияет на структуру алгоритма. В случае сортировки это влияние настолько сильно, что выделяют два типа сортировки: внутренняя (для данных, хранящихся в ОЗУ, к которым возможен произвольный доступ) и внешняя (для данных, хранящихся на внешних устройствах с произвольным доступом). Чтобы понять разницу в подходах к внутренней и внешней сортировке, можно воспользоваться аналогией с упорядочиванием колоды игральных карт. Хранение в ОЗУ соответствует раскладыванию карт на столе так, чтобы каждая карта была непосредственно доступна. Хранение на устройствах с последовательным доступом похоже на расположение всех карт в виде одной стопки так, что доступ возможен только к самой верхней карте.

Помимо деления на внутренние и внешние сортировки принято классифицировать по устойчивости. Метод сортировки называется устойчивым, если он сохраняет относительный порядок элементов с равными ключами. Например, если студенты из рассмотренного ранее примера уже были отсортированы по возрасту, то их сортировка по алфавиту устойчивым алгоритмом приведёт к тому, что однофамильцы останутся упорядоченными по возрасту. Устойчивость является полезным свойством, если элементы уже упорядочены по некоторым ключам.

Базовые алгоритмы внутренней сортировки

Главное требование для алгоритмов внутренней сортировки – экономное использование памяти. То есть все перестановки должны выполняться непосредственно в сортируемом массиве без дополнительного расхода памяти. Существуют эффективные алгоритмы, требующие порядка $N * \log(N)$ перестановок, но мы сначала рассмотрим несколько простых и очевидных методов,

требующих порядка $O(N^2)$ перестановок. Есть несколько причин, по которым необходимо рассмотреть эти медленные алгоритмы, прежде чем переходить к более эффективным алгоритмам:

- Простые методы особенно хороши для понимания базовых принципов;
- Соответствующие программы легко понять, кроме того они довольно просты и занимают мало места в памяти;
- Быстрые алгоритмы требуют меньшего количества операций, но эти операции более сложные, за счёт этого простые алгоритмы выгоднее для массивов малых размеров.

Сортировка вставками

В методе сортировки вставками все элементы мысленно разделяются на две последовательности: приёмник (все элементы приёмника упорядочены) и источник (элементы в источнике не упорядочены). На каждом шаге алгоритма в последовательности-источнике берётся очередной элемент и ставится в правильную позицию в последовательности-приёмнике. При поиске правильной позиции для очередного элемента выполняется просеивание этого элемента по последовательности-приёмнику справа налево. На каждом шаге элемент, предварительно сохранённый в локальную переменную (например x), сравнивается с расположенным левее элементом A_j. Затем, в зависимости от результата сравнения, либо A_j передвигается вправо в A_{j-1} и j уменьшается на единицу, либо найдена правильная позиция для элемента, сохранённого в x. Заметим, что есть два условия, которые могут вызвать прекращение процесса просеивания:

- У элемента A_j ключ оказался меньше, чем ключ x
- Обнаружен левый конец последовательности-приёмника

Листинг 1

```
procedure InsertionSort;
var
x: integer;
i,j: integer;
begin
  for i:=2 to N do
  begin
    x:=a[i];
    j:=i;
    while (j>1) and (x<a[j-1]) do
    begin
      a[j]:=a[j-1];
      j:=j-1;
    end;
    a[j]:=x;
  end;
end;
```

Заметив, что последовательность-приёмник уже упорядочена, можно немного ускорить алгоритм. Для этого достаточно использовать более быстрый способ нахождения позиции для вставки. Очевидный выбор – алгоритм деления пополам, в котором проверяется середина последовательности-приемника, затем продолжаются деления пополам, пока не будет найдена точка вставки. Такой модифицированный алгоритм называется сортировкой двоичными вставками:

Листинг 2

```
procedure BinaryInsertionSort;
var
x: integer;
i,j,L,R,m: integer;
begin
  for i:=2 to N do
  begin
    x:=a[i];
    L:=1;
    R:=i;
    while L<R do
    begin
      m:=(L+R) div 2;
      if a[m]<x then
        L:=m+1
      else
        R:=m;
    end;
    for j:=i downto R+1 do
      a[j]:=a[j-1];
    a[R]:=x;
  end;
end;
```

Позиция вставки найдена, когда $L = R$. Для этого интервал длиной k надо делить пополам $\log(k)$ раз. Использование двоичного поиска сокращает число сравнений, но никак не влияет на число перестановок. На самом деле это улучшение не принципиально. так как пересылки более затратны, чем сравнения ключей, а количество пересылок по-прежнему имеет порядок $O(N^2)$.

Сортировка выбором

В алгоритме сортировки выбором, как и в алгоритме сортировки вставками, сортируемый массив делится на две последовательности: приёмник (в котором элементы уже упорядочены) и источник (где хранятся ещё не упорядоченные элементы). В отличие от сортировки вставками, где берётся очередной элемент из источника и ищется позиция в приёмнике, алгоритм сортировки выбором выполняет поиск минимального элемента в последовательности-источнике и ставит найденный элемент в конец последовательности-приёмника.

Листинг 3

```
procedure SelectionSort;
var
x: integer;
i,j,k: integer;
begin
  for i:=1 to N-1 do
  begin
    k:=i;
    x:=a[i];
    for j:=i+1 to N do
    begin
      if a[j]<x then
      begin
        k:=j;
        x:=a[k];
```

```
      end;
    end;
    a[k]:=a[i];
    a[i]:=x;
  end;
end;
```

Сложность алгоритма сортировки выбором оценивается как $O(N^2)$. Несмотря на одинаковый класс сложности, метод сортировки вставками на практике оказывается быстрее сортировки выбором.

Пузырьковая сортировка

Пузырьковую сортировку часто называют обменной, так как обмен двух соседних элементов массива в этом методе играет ключевую роль: сравнение и обмен соседних элементов будут продолжаться до тех пор, пока не будет отсортирован весь массив.

При выполнении пузырьковой сортировки выполняются проходы по массиву справа налево, при этом происходит сравнение соседних элементов и их обмен. Таким образом наименьшие элементы просеиваются в направлении левого конца массива. Если представить массив расположенным вертикально, то сортировка будет выглядеть, как погружение тяжёлых элементов и всплытие лёгких. Именно поэтому данный метод называется пузырьковой сортировкой.

Листинг 4

```
procedure BubbleSort;
var
x: integer;
i,j: integer;
begin
  for i:=1 to N-1 do
    for j:=N downto i+1 do
      if a[j-1]>a[j] then
      begin
        x:=a[j-1];
        a[j-1]:=a[j];
        a[j]:=x;
      end;
end;
```

Очевидно, что этот алгоритм можно улучшить. Часто бывает так, что последние проходы не влияют на порядок элементов, так как они уже полностью отсортированы. Самый простой способ улучшить алгоритм – запоминать, имел ли место хотя бы один обмен во время очередного прохода. Если за проход не было ни одного обмена, это значит, что массив уже полностью отсортирован, и алгоритм может быть остановлен. Такой вариант алгоритма иногда называют «пузырёк с флажком». Однако можно сделать ещё одно улучшение, запоминая не только факт обмена, но и позицию последнего обмена. Ясно, например, что все пары соседних элементов левее этого значения индекса уже упорядочены, поэтому следующие проходы могут останавливаться на этом значении индекса.

Кроме того, можно заметить довольно любопытную асимметрию. Если в начале только один «лёгкий» элемент стоит не на своём месте в «тяжёлом» конце, то он доберётся до своего места за один проход, а если «тяжёлый» элемент стоит в «лёгком» конце, то он будет опускаться в свою позицию только на один шаг за каждый проход.

Такая неестественная асимметрия наводит на мысль о третьем усовершенствовании: менять направление последовательных проходов. Получающийся алгоритм называется шейкерной сортировкой, по аналогии с шейкером, используемом барменом.

Листинг 5

```
procedure ShakerSort;
var
x: integer;
j,L,R,k: integer;
begin
  L:=2;
  R:=N;
  k:=R;
  REPEAT
    for j:=R downto L do
      if a[j-1]>a[j] then
      begin
        x:=a[j-1];
        a[j-1]:=a[j];
        a[j]:=x;
        k:=j;
      end;
    L:=k+1;
    for j:=L to R do
      if a[j-1]>a[j] then
      begin
        x:=a[j-1];
        a[j-1]:=a[j];
        a[j]:=x;
        k:=j;
      end;
    R:=k-1;
  UNTIL L>R;
end;
```

Пузырьковая сортировка, как и рассмотренные ранее алгоритмы, имеет сложность $O(N^2)$. Шейкерная сортировка никак не меняет количество перестановок, а только помогает избавиться от лишних сравнений. Дональд Кнут показал, что среднее число сравнений в шейкерной сортировке равно $(N^2 - N * (k + \ln(N)))/2$, но, так как количество перестановок не изменилось, сложность шейкерной сортировки тоже составляет $O(N^2)$.

Сравнение базовых алгоритмов внутренней сортировки

В Табл. 1 показана зависимость времени сортировки массива случайных чисел от размера массива и используемого алгоритма. Все измерения производились на обычном персональном компьютере.

Табл. 1. Зависимость времени сортировки от размера случайного массива и используемого алгоритма

Размер массива	Сортировка				
	Вставками	Двоичными вставками	Выбором	Пузырьком	Шейкерная
512	0,001	<мс	0,001	0,001	0,001
1024	0,001	0,001	0,002	0,004	0,003

2048	0,003	0,003	0,01	0,017	0,011
4096	0,014	0,011	0,039	0,067	0,062
8192	0,054	0,041	0,245	0,377	0,234
16384	0,33	0,222	0,753	1,157	1,006
32768	0,936	0,763	2,766	5,134	4,319
65536	4,04	3,266	12,288	20,398	16,871
131072	17,203	13,304	48,231	83,509	71,537
262144	67,68	49,944	190,034	350,494	287,609
524288	267,185	204,412	767,644	1368,294	1040,356
1048576	1063,415	801,508	3074,911	5585,435	4512,285

Полученные результаты позволяют убедиться в том, что временная сложность этих алгоритмов действительно пропорциональна $O(N^2)$: при переходе от строки к строке размер массива увеличивается в два раза, а время сортировки возрастает почти в четыре раза.

Кроме того, Табл. 1 позволяет сделать некоторые выводы об относительной скорости работы изученных методов. Быстрейшим оказывается метод сортировки двоичными вставками, за ним с небольшим отрывом следует метод сортировки вставками, третье место с существенным отрывом занимает сортировка выбором, а шейкерная и пузырьковая сортировки замыкают список производительности базовых сортировок.

Помимо сравнения производительности на случайных данных довольно интересно сравнить эффективность сортировок на уже упорядоченных массивах и массивах, упорядоченных в обратном порядке.

Табл. 2. Зависимость времени сортировки от размера упорядоченного массива и используемого алгоритма

Размер массива	**Сортировка**				
	Вставками	**Двоичными вставками**	**Выбором**	**Пузырьком**	**Шейкерная**
512	<мс	<мс	0,001	0,001	<мс
1024	<мс	<мс	0,001	0,002	<мс
2048	<мс	<мс	0,004	0,005	<мс
4096	<мс	0,001	0,044	0,021	<мс
8192	<мс	0,001	0,069	0,083	<мс
16384	<мс	0,001	0,352	0,341	<мс
32768	<мс	0,004	1,367	1,438	<мс
65536	0,001	0,004	5,043	6,234	<мс
131072	0,001	0,021	22,411	27,035	<мс
262144	0,001	0,047	91,165	110,007	<мс
524288	0,006	0,041	364,172	432,075	0,001
1048576	0,005	0,132	1425,93	1717,414	0,003

Табл. 3. Зависимость времени сортировки от размера упорядоченного в обратном порядке массива и используемого алгоритма

Размер массива	**Сортировка**				
	Вставками	**Двоичными вставками**	**Выбором**	**Пузырьком**	**Шейкерная**
512	0,001	0,001	<мс	0,001	0,001

1024	0,002	0,001	0,001	0,003	0,003
2048	0,007	0,005	0,007	0,012	0,012
4096	0,026	0,02	0,02	0,047	0,075
8192	0,175	0,081	0,08	0,204	0,229
16384	0,435	0,468	0,339	0,818	0,823
32768	1,851	1,593	1,383	3,778	3,634
65536	7,825	6,496	6,055	15,511	15,748
131072	32,315	22,87	23,97	58,697	60,501
262144	132,404	105,238	99,832	235,387	239,167
524288	540,227	417,817	420,853	952,926	995,296
1048576	2228,403	1671,841	1766,5	3833,158	3980,125

Учитывая устойчивость сортировки вставками и её высокое (по сравнению с другими базовыми алгоритмами) быстродействие как на случайных, так и на упорядоченных массивах, можно рекомендовать эту сортировку для практического использования на небольших объёмах исходных данных.