Гончаренко Валерий Евстафиевич
Доцент кафедры «Информационные технологии в экономике и организация производства» (ИТЭ и ОП) ГОУ ВПО «Ивановский государственный университет», кандидат технических наук. Ответственный организатор городской олимпиады школьников по информатике, член экспертной комиссии ЕГЭ по информатике и ИКТ.

Алгоритм сортировки подсчётом и варианты его программной реализации

Сортировка линейного массива является классической задачей обработки данных. Алгоритмов сортировки достаточно много. Наиболее полный их список приведён в Википедии в разделе «Алгоритм сортировки» [1]. В учебных курсах по программированию традиционно рассматриваются два наиболее простых для понимания алгоритма: сортировка выбором и сортировка пузырьком. В этой статье вы познакомитесь ещё с одним алгоритмом быстрой сортировки.

Вычислительная сложность алгоритмов сортировки выбором и сортировки пузырьком – $O(n^2)$, где n – количество элементов массива. Эти алгоритмы выполняют очень большое количество операций сравнения и перестановок элементов массива в процессе их упорядочивания по признаку невозрастания или неубывания значений. Для массивов размером от 10 до 1000 элементов при быстродействии современных ПК это, возможно, и не создаёт проблем с задержкой времени выполнения программы, а вот для больших массивов размером 100–200 тыс. элементов или очень больших – размером 1–50 млн элементов и более – возникают реальные неудобства из-за большой длительности выполнения программ. А ведь в конечных программах сортировка массива может быть лишь предварительным этапом в обработке данных.

В противовес двум простым, но очень медленным алгоритмам, рассматривается рекурсивный алгоритм быстрой сортировки, вычислительная сложность которого оценивается как $O(n\log_2 n)$, хотя в худшем случае может оказаться равной и $O(n^2)$. В этом алгоритме также используются операции сравнения и перестановок элементов, но уже в гораздо меньшем количестве. Для понимания этот алгоритм сложнее и относится к алгоритмам неустойчивой сортировки.

А можно ли выполнить сортировку линейного массива без операций сравнения и промежуточных перестановок элементов, а сразу определить их место в упорядоченной последовательности? Да, можно, и это реализуется в алгоритме, получившем название «Сортировка подсчётом» (Counting sort), сложность алгоритма $O(n+k)$.

Суть вопроса на простом примере

Рассмотрим более подробно идею этого алгоритма на примере линейного массива целых чисел из пяти элементов: $x[0]=-38$, $x[1]=64$, $x[2]=7$, $x[3]=64$, $x[4]=7$.

Любую последовательность целых чисел можно преобразовать к значениям на отрезке положительной полуоси целых чисел, начиная с нуля. Для этого достаточно из всех значений алгебраически вычесть наименьшее значение (min). В приведённом примере $\min=-38$, вычитая его из всех элементов, получим: $x[0]=0$, $x[1]=102$, $x[2]=45$, $x[3]=102$, $x[4]=45$.

Числа на числовой оси априори упорядочены, значение числа одновременно является его порядковым номером в линейной последовательности чисел. Каждому элементу преобразованного массива можно поставить в соответствие значение на упорядоченной последовательности чисел на отрезке положительной полуоси чисел. Остаётся решить только вопрос с повторяющимися и отсутствующими значениями в сортируемом массиве.

Рассматриваемому массиву можно поставить в соответствие не отрезок положительной полуоси чисел, а вспомогательный линейный массив элементов, индексы которых начинаются с нуля и заканчиваются значением $\max-\min=64-(-38)=102$, где max и min – соответственно наибольшее и наименьшее значения в сортируемом массиве. Для нашего примера индексы вспомогательного массива будут принимать значения от 0 до 102. Обозначив вспомогательный массив через m и изначально присвоив всем его элементам нулевые значения, получим:

$$m[0]=0,\ m[1]=0,\ ...,\ m[102]=0.$$

Если теперь последовательно обращаться к преобразованным значениям исходного массива $x[i]$ и увеличивать на 1 значение элемента массива m с индексом, равным $x[i]$, то в итоге получим: $m[0]=1$, $m[1]=0$, ..., $m[45]=2$, $m[46]=0$, ..., $m[101]=0$, $m[102]=2$. Все не указанные явно элементы этого массива имеют нулевые значения, а сумма равна количеству элементов массива x.

Используя сформированный массив m, можно получить упорядоченный массив x. Для этого необходимо обращаться последовательно к каждому элементу массива m и создавать $m[j]$ раз очередное значение индекса i и присваивать элементу $x[i]$ значение $(j+\min)$. При значении $m[j]=0$ соответственно ни разу не создаётся очередное значение индекса i и не создаётся элемент массива $x[i]$. В итоге получим упорядоченный массив x: $x[0]=-38$, $x[1]=7$, $x[2]=7$, $x[3]=64$, $x[4]=64$.

Размер массива m зависит от вариационного размаха значений элементов массива x, и для рассмотренного примера оказывается равным

$$(\max - \min + 1) = 64 - (-38) + 1 = 103.$$

Дисковая память снимает проблему огромных массивов

Известно, что Pascal отводит для статических данных выполняемой программы один сектор оперативной памяти ёмкостью 64 Кб. Если объявить линейный массив из 30 000 элементов типа integer, то он займёт почти всю область выделенной памяти и на вспомогательный массив места не останется.

Когда мы говорим о каком-то количестве данных, необходимых для решения определённой задачи, совсем не обязательно представлять их в форме линейного массива в оперативной памяти компьютера. Эта форма хранения данных скорее является временной, предоставляющей определённые удобства обработки данных. Долговременным и надёжным хранением данных является структура их хранения в типизированном файле на жёстком диске или ином внешнем носителе данных. Если предположить, что исходные данные будут записаны в типизированный файл, а также будет сформирован вспомогательный файл записей по аналогии с массивом m, то снимается проблема предельного количества исходных данных, для которых необходимо выполнить сортировку подсчётом. Отметим, что структура типизированного файла

очень близка к структуре линейного массива, так как данные в типизированном файле образуют вектор данных, что обеспечивает прямой доступ к записям в файле по их номеру от начала вектора. Конфигурация современных ПК предоставляет практически неограниченный ресурс дисковой памяти. С учётом вышесказанного ниже приведён исходный текст программы на языке Pascal, выполняющей сортировку записей в типизированном файле целых чисел в соответствии с алгоритмом сортировки подсчётом.

```
1   Program sort_file_to_file;
2   Uses Crt;
3   Const NameDat='c:\ffdat.sss';
4      NameSort='c:\ffsort.sss';
5   Var fdat,fsort:file of integer;
6     i,j,N:longint;
7     min,max,z:integer;
8     k:integer;
9   begin
10    Clrscr;
11    randomize;
```

```
12    write('Сколько чисел записать в файл?-  ');
13    readln(N);
14    assign(fdat,NameDat);
15    rewrite(fdat);
16    max:=Low(integer);
17    min:=High(integer);
18    for i:=0 to (N-1) do
19       begin
20         z:=random(1000);
21          write(fdat,z);
22          if z<min then min:=z;
23          if z>max then max:=z;
24          write(z,' ');
25       end;
26     writeln;
27     writeln('min=',min,'max=',max);
28     assign(fsort,NameSort);
29     rewrite(fsort);
30     z:=0;
31     for i:=0 to (max-min) do
32         write(fsort,z);
33     seek(fdat,0);
34     repeat
35        read(fdat,z);
36        z:=z-min;
37         seek(fsort,z);
38         read(fsort,k);
39         inc(k);
40         seek(fsort,z);
41         write(fsort,k);
42     until Eof(fdat);
43     seek(fsort,0);
44     seek(fdat,0);
45     i:=0;
46     repeat
47        read(fsort,z);
48        k:=FilePos(fsort)+min-1;
49        for j:=1 to z do
50           begin
51             inc(i);
52             write(fdat,k);
53           end;
54     until Eof(fsort);
55     seek(fdat,0);
56     writeln('Упорядоченные числа из файла данных:');
57     writeln('Для продолжения нажмите ENTER');
58     readln;
59     repeat
60        read(fdat,z);
61        write(z,'  ');
```

```
62     until Eof(fdat);
63  writeln;
64  writeln('Всего в упорядоченном файле чисел ',i);
65  close(fdat);
66  erase(fdat);
67  close(fsort);
68  erase(fsort);
69  writeln('Для завершения нажмите ENTER');
70  readln;
71  End.
```

Для удобства комментария программы её строки пронумерованы.

В строках 3–4 объявлены две константы строкового типа **NameDat** – полное имя файла исходных данных и **NameSort** – полное имя вспомогательного файла. При необходимости эти имена можно изменить в соответствии с файловой структурой ПК.

В строке 12 на экран выводится запрос на количество чисел N, которые с помощью генератора случайных чисел **random( )** будут записаны в типизированный файл **fdat**. Предоставляется возможность ввести значение N из очень большого диапазона. Необходимо помнить, что N предопределяет время выполнения программы, и для $N = 100$ млн необходимо будет набраться терпения в ожидании завершения работы программы.

В строках 18–25 в цикле **for** в файл **fdat** записывается N целых случайных чисел и определяются наименьшее (min) и наибольшее (max) значения.

После открытия вспомогательного файла **fsort** в строках 31–32 в него записываются нулевые значения $(\max - \min + 1)$ раз.

В строках 34–42 с помощью цикла **repeat** из файла исходных данных последовательно считываются целые числа в переменную z, из которой вычитается значение min. Полученное значение z используется в качестве номера записи во вспомогательном файле **fsort**. Значение записи считывается в переменную k, увеличивается на 1 и опять записывается на прежнее место. По сути своей значение k отражает количество чисел в исходном файле, значения которых равны номерам записи во вспомогательном файле с учётом смещения на min.

После формирования вспомогательного файла остаётся только заново присвоить значения всем записям в исходном файле, которые будут упорядочены по признаку неубывания, что и выполняется в строках 46–54 в цикле **repeat**. Каждая запись учитывается в счётчике i, который по завершении цикла **repeat** должен совпасть со значением N. Счётчик служит для контроля правильности работы программы, и его использование может быть исключено из программы.

В строках 59–62 упорядоченные значения из файла исходных данных выводятся на экран. Выводы на экран наглядно демонстрируют состояние исходного файла данных до и после сортировки, но для больших значений N существенно увеличивают время выполнения программы и могут быть исключены.

В строках 65–68 используемые в программе файлы **fdat** и **fsort** закрываются и удаляются из файло-

вой структуры. В реальной задаче, скорее всего, файл **fdat** не надо удалять, а в сохранении файла **fsort** нет необходимости.

Чтобы убедиться в том, что программа действительно работоспособна и действительно выполняет сортировку чисел в исходном файле, её обязательно надо набрать в редакторе одной из версий языка Pascal и запустить на выполнение, предварительно устранив все непроизвольные ошибки, которые заметит компилятор.

Дисковая память – хорошо, а оперативная – лучше

Рассмотрим использование дополнительного ресурса оперативной памяти в программной реализации алгоритма сортировки подсчётом. Если дисковую память объёмом 80–120 Гб можно условно считать не ограниченной для размещения в ней основного и вспомогательных файлов, то оперативную память современных ПК порядка 2 Гб можно считать просто большой. Это обстоятельство заставляет выполнить предварительную оценку её достаточности для использования в каждом конкретном случае. Выбирая оперативную память в качестве дополнительного ресурса, получаем возможность существенного сокращения времени обработки данных.

Доступ к данным в оперативной памяти требует намного меньше времени по сравнению с временем доступа к данным на жёстком диске.

Использование вспомогательного файла на дисковой памяти гарантирует запись файла практически любой длины. Использование для аналогичной цели динамической памяти даёт существенный выигрыш во времени выполнения программы, но необходимо быть уверенным в достаточном её количестве для выполняемой программы. Ниже приведён пример программы, в которой для сортировки исходного массива из 30 000 элементов используется динамический массив **pm**.

```
1    Program sort_din;
2    Uses Crt;
3    Const n=30000;
4    Type Tmas=array[0..(N-1)] of integer;
5      Tpoint=^Tmas;
6    Var mas:Tmas;
7      pm:Tpoint;
8      i,j,k:integer;
9      min,max,z:integer;
10   begin
11     Clrscr;
12     writeln('Исходный массив:');
13     min:=High(integer);
```

```
14      max:=Low(integer);
15      for i:=0 to (n-1) do
16         begin
17           mas[i]:=random(420)-100;
18           write(mas[i]:5);
19           if mas[i]<min then min:=mas[i];
20           if mas[i]>max then max:=mas[i];
21         end;
22       writeln;
23       writeln('min=',min,'max=',max);
24       readln;
25       pm:=nil;
26       new(pm);
27       for i:=0 to (max-min) do
28           pm^[i]:=0;
29       for i:=0 to (n-1) do
30          begin
31            z:=mas[i]-min;
32            inc(pm^[z]);
33          end;
34       i:=-1;
35       for j:=0 to (max-min) do
36           for k:=1 to pm^[j] do
37              begin
38                inc(i);
39                mas[i]:=j+min;
40              end;
41       writeln('отсортированный массив:');
42       for i:=0 to (n-1) do
43         write(mas[i]:5);
44       dispose(pm);
45       pm:=nil;
46       writeln;
47       writeln('Для завершения нажмите ENTER');
48    readln;
49    End.
```

Комментарии к программе:

В строках 4–5 описываются тип линейного массива **Tmas** и тип указателя на линейный массив **Tpoint**.

В строках 12–22 выполняется инициализация исходного массива размером N и определение в нём наименьшего (min) и наибольшего (max) значений среди элементов массива.

В строках 26–29 с помощью процедуры **new** в области динамически распределяемой памяти (куча) резервируется место для динамического линейного массива **pm**. Динамический массив используется по аналогии, как и типизированный файл в описанных ранее программах. Использование динамического массива упрощает программу, т. к. отпадает необходимость в выполнении операций позиционирования внутреннего указателя в файле. Всем элементам динамического мас-

сива с индексами от 0 до (max – – min) присваиваются нулевые значения.

В строках 29–33 в цикле **for** перебираются последовательно все индексы исходного массива и вычисляется смещённое значение $z = \text{mas}[i] - \text{min}$, и элемент динамического массива с номером z увеличивается на 1. Значение этого элемента отражает количество элементов с одинаковым значением в исходном массиве, равных номеру элемента в динамическом массиве.

В строках 35–41, после того как завершилось формирование динамического массива, в цикле **for** перебираются последовательно все индексы динамического массива от 0 до (max – min), и во внутреннем цикле **for** при изменении счётчика цикла k от 1 до значения элемента динамического массива выполняется генерация очередного номера исходного массива, которому присваивается значение $(j + \text{min})$, где j – номер элемента динамического массива. В итоге формируется упорядоченный по признаку неубывания значений исходный массив.

В строках 44–45 с помощью процедуры **dispose** освобождается динамически распределённая под линейный массив память и указателю **pm** присваивается значение **nil** (в никуда).

Выполнение сортировки линейного массива из 30 000 элементов рассмотренной программой выполняется практически мгновенно. Задержка времени при выводе результатов на экран монитора не фиксируется. Визуально такие же результаты показывает и программа сортировки при реализации рекурсивного алгоритма быстрой сортировки.

Рассмотренные программы расширяют представление о возможных применениях алгоритма сортировки подсчётом, используя при этом дополнительные ресурсы дисковой или оперативной памяти ПК. Этот алгоритм характеризуется высокой скоростью выполнения. Можно заметить ещё одно преимущество этого алгоритма. Если необходимо из исходных данных получить упорядоченную последовательность без повторяющихся значений, то достаточно внести незначительные изменения в программу, не увеличивая на единицу вспомогательный элемент, а присваивая ему единицу.

В заключение можно отметить, что программная реализация задач на языке Pascal преследует цель сделать их понятными для более широкого круга читателей. Нет никаких препятствий для реализации рассмотренных алгоритмов, например, на языках C++ или Java, получивших популярность в последнее время.

Литература

1. http://ru.wikipedia.org/wiki/Алгоритм_сортировки.