Занятие 17. Быстрая сортировка

Пусть требуется отсортировать часть массива $a[l..r]$ по возрастанию, т.е. сортировке подлежат все элементы массивы a с индексами от l до r включительно.

Алгоритм быстрой сортировки так же, как и алгоритм сортировки слиянием, основан на алгоритмической стратегии «разделяй и властвуй». Но теперь сортируемый массив делится не пополам, а по несколько иному признаку. Этапы решения задачи сортировки массива по возрастанию будут следующие.

1. *Разделение*. Выбирается некоторый опорный элемент массива, обозначим его (элемент) x. Элементы массива переупорядочиваются таким образом, чтобы в левой части массива расположились элементы, меньшие или равные x, а в правой — большие или равные x.
2. *Покорение*. К левой части массива применяется алгоритм быстрой сортировки, если длина этой части больше одного элемента. Аналогично для правой части.
3. *Комбинирование*. Поскольку все операции с массивом выполняются в месте его начального расположения (без использования вспомогательных массивов), то на этом этапе вообще ничего делать не надо.

Из сказанного выше ясно, что самую большую сложность представляет этап разделения.

Разделение

Выбор опорного элемента

Как выбирать опорный элемент? Оказывается, от ответа на этот вопрос зависит скорость работы алгоритма, но не его правильность. Алгоритм будет приводить к требуемой цели — упорядочиванию элементов массива по возрастанию — при любом выборе опорного элемента. А вот для достижения максимальной скорости работы опорный элемент надо выбирать так, чтобы размеры левой и правой частей массива[1] были примерно равными. Тогда количество действий, требуемых для сортировки массива из n элементов, при больших n будет расти пропорционально $n \cdot \log(n)$. Если же на каждом этапе быстрой сортировки опорный элемент выбирать так, что одна из частей массива: левая или правая — будет состоять всего из одного элемента, то производительность метода будет наихудшей. Количество действий в этом случае будет расти пропорционально n^2, как и для прямых методов сортировки. В среднем же для случайных массивов производительность метода мало зависит от выбора опорного элемента и остаётся пропорциональной $n \cdot \log(n)$. Мы будем использовать в качестве опорного элемент, стоящий в середине сортируемой части массива.

Переупорядочивание

Этот этап можно осуществить следующим образом.

1. Выбираем опорный элемент x:=$a[(l+r)$ div $2]$
2. $i \coloneqq l$
3. $j \coloneqq r$
4. Двигаясь вправо, просматриваем поочерёдно все элементы массива, пока не встретим некоторый $a[i] \geq x$.
5. Двигаясь влево, просматриваем поочерёдно все элементы массива, пока не встретим некоторый $a[j] \leq x$.
6. Если $i \leq j$,
    a) меняем местами $a[i]$ и $a[j]$,
    b) i увеличиваем на единицу, продвигаясь на 1 шаг вправо,
    c) j уменьшаем на единицу, продвигаясь на 1 шаг влево.
7. Если $i > j$, то цель достигнута, иначе перейти к пункту 4

[1] См. этап «Разделение»

Выполнение алгоритма закончится, когда процессы просмотров в пунктах 4 и 5 встретятся где-то в середине массива. В результате окажется, что все элементы подмассива $a[l..i-1]$ меньше или равны x, а все элементы подмассива $a[j+1..r]$ больше или равны x.

Примерно так описал этап разделения автор алгоритма быстрой сортировки Ч. Хоар.

Пример 1.

Покажем процесс разделения на примере следующего массива $a[1..8]$.

44	55	12	***42***	94	6	18	67

Опорным элементом будет $x = 42$.
Выполняя пункт 4 алгоритма, получим $i = 1$.
Выполняя пункт 5, получим $j = 7$.
Обмениваем $a[1]$ и $a[7]$ и продвигаемся к следующим элементам. Новые значения $i = 2$ и $j = 6$

18	55	12	***42***	94	6	44	67

Уже пройденная часть массива показана заливкой.
$i \le j$, поэтому продолжаем выполнение операторов тела цикла — переходим к пункту 4.
В результате выполнения п. 4 и п. 5 значения переменных i и j не изменятся.
Выполняя п. 6, обмениваем $a[2]$ с $a[6]$ и переходим к $i = 3$ и $j = 5$.

18	6	12	***42***	94	55	44	67

Опять $i \le j$, поэтому переходим к п. 4.
В результате п. 4 получаем $i = 4$.
В результате п. 5 получаем $j = 4$.
Операторы п. 6 приведут к тому, что $a[4] = 42$ будет обменян сам с собой, а также (и это главное) изменятся значения переменных i и j. Теперь $i = 5$, а $j = 3$.

18	6	12	***42***	94	55	44	67

Теперь $i > j$, а это значит, что работа процедуры разделения завершена. Теперь все элементы в части массива левее индекса $i = 5$ меньше или равны x, а правее $j = 3$ — больше или равны x. Цель достигнута.

Обратим внимание на некоторые детали. Весьма важно, что в п. 4 мы пропускали элементы строго меньшие x, а в п. 5 — строго большие x. Это гарантирует, что мы не выйдем за границы массива: x будет играть роль барьерного элемента. Однако при этом придётся выполнять перестановку элементов, равных x, что совсем не является обязательным.

В пункте 6 нельзя отказаться от проверки условия. Убедимся в этом на ещё одном примере.

Пример 2.

Пусть мы сортируем следующий массив.

1	2	***3***	10	0

Опорный элемент $x = 3$. После первого выполнения циклов из пунктов 4, 5 окажется $i = 3$, $j = 5$. В п. 6 обмениваем $a[3]$ и $a[5]$. Опорный элемент уходит со своего места. Теперь $i = j = 4$. Получаем такую картину.

1	2	0	10	***3***
			i, j	

Далее пункт 7 возвращает нас к пункту 4. Выполнив ещё раз п. 4, получим всё то же $i = 4$.

Выполнив пункт 5, получим $j = 3$.

1	2	0	10	***3***
		j	*i*	

Весь массив уже просмотрен, и, как и требовалось, все его элементы левее индекса i меньше или равны x, а все элементы с индексами, большими j, — больше или равны x. Обмен $a[i]$ и $a[j]$ теперь только испортит результат. Чтобы этого не произошло, и нужна проверка условия в пункте 6.

Заметим, что $i \le j$ в пункте 6 нельзя просто заменить строгим неравенством $i < j$. При $i = j$ можно не выполнять перестановку пункта 6(a), но сдвинуться к следующим элементам, как того требуют пункты 6(b) и 6(c), нужно обязательно. Иначе мы можем просто «застрять» на элементе, для которого $i = j$, если он окажется равным опорному, потому что выполнение цикла пунктов 4 – 7 в этом случае никогда не завершится. Так случилось бы в примере 1.

Покорение

Чтобы закончить построение алгоритма, осталось всего лишь добавить рекурсивные вызовы процедуры для левой и правой части массива, если в них больше одного элемента. Как формализовать выражение «больше одного элемента»?

Левая часть массива с элементами не больше опорного располагается от индекса l до индекса $i - 1$ включительно. Следовательно, если $i < r$, то существует и правая часть длиной больше одного элемента. Её индексы от i до r включительно.

Правая часть массива с элементами не меньше опорного имеет индексы, большие j. Следовательно, если $j > l$, то существует и левая часть длиной больше одного элемента. Её индексы от l до j включительно.

Блок-схема процедуры сортировки QSort(a,L,R) представлена на рис. 1. Здесь a — имя сортируемого массива, L — левая, а R — правая граница сортируемой части массива.

Чтобы отсортировать весь массив $a[1..n]$ надо вызвать эту процедуру следующим образом

QSort(a,1,n)

Программа быстрой сортировки

```
{$mode delphi}
Program Quick_Sort;
Const MAXN = 10000000;   //Максимальное количество элементов массива

Type TElem = Integer;
     massiv  = array[1..MAXN] of TElem;

//Быстрая сортировка массива a из n элементов
Procedure QuickSort(Var a : massiv; n : Integer);

  Procedure QSort(L,R : Integer);
  Var x, tmp : TElem;
      i, j   : Integer;
  Begin
    x:=a[(L+R) div 2];
    i:=L;
    j:=R;
    Repeat
      While a[i] < x Do
        i:=i+1;
      While a[j] > x Do
        j:=j-1;
```

```
      If i<=j Then
      Begin
        tmp:=a[i]; //Перестановка a[i] и a[j]
        a[i]:=a[j];
        a[j]:=tmp;
        i:=i+1;
        j:=j-1
      End
    Until i>j;
    If i<R Then
      QSort(i,R);
    If j>L Then
      QSort(L,j)
  End;

Begin //для QuickSort
  QSort(1,n)
End;


//Основной алгоритм
Var a   : massiv;
    f   : Text;
    i,N : Integer;
Begin
  Assign(f,'qsort.in');
  Reset(f);
  Read(f,N); //Количество элементов массива)
  For i:=1 To N Do
  Read(f,a[i]);
  Close(f);

  QuickSort(a,N);

  Assign(f,'qsort.out');
  Rewrite(f);
  For i:=1 To N Do
  Begin
    Write(f,a[i]:10, ' ');
    If i mod 7 = 0 Then
    WriteLn(f)
  End;
  Close(f)
End.
```

В этой программе мы применили процедуру QSort, вложенную в процедуру QuickSort. Параметры процедуры QuickSort (массив *a* и количество элементов *n*) являются глобальными для вложенной процедуры QSort. Вообще, пусть мы описали такую последовательность подпрограмм.

```
Procedure Proc1(a,b : Integer);
Var c,d : Real; //a,b,c,d доступны для Proc1, Proc2, Func1
     //Вызов Proc2 и Func1 доступен только внутри блока,
     //начинаемого здесь …
     Procedure Proc2;
     Var m,n : Integer; //Локальные для Proc2
     Begin
     …
     End;

     Function Func1(x : Integer) : Integer;
     Var z : Integer; //Локальная для Func1
     Begin
     …
     End;
```

```
Var p : Integer;
Begin
…
End; // … и заканчиваемого здесь. Конец Proc1.
```

Тогда параметры a и b, а также переменные c и d являются глобальными для процедур Proc1, Proc2 и функции Func1. Переменные же m и n являются локальными для Proc2, параметр x и переменная z — локальными для Func1. Переменная p не будет доступна из Proc2 и Func1. Кроме того, Proc2 и Func1 не могут быть вызваны за пределами того блока, в котором они описаны.

Конечно, для того, чтобы запрограммировать алгоритм быстрой сортировки, не обязательно пользоваться вложенными подпрограммами. Можно было просто описать процедуру

```
Procedure QSort(Var a : massiv; L,R : Integer);
```

вообще не используя QuickSort, а потом вызывать её QSort(a,1,n). Если в программе предполагается использовать быструю сортировку части массива, не обязательно начинающуюся с первого элемента, то последний вариант является оптимальным.

Ещё о свойствах быстрой сортировки

Несмотря на то, что в худшем случае скорость работы быстрой сортировки невелика, в среднем это самый лучший из известных методов сортировки массива на том же месте (т.е. без использования дополнительной памяти). Если сравнивать его с сортировкой слиянием, то стоит отметить, что в среднем быстрая сортировка справляется со своей работой быстрее. Однако сортировка слиянием даже в худшем случае сохраняет скорость работы $n \cdot \log(n)$, в то время как быстрая сортировка становится квадратичной.

Ещё одним недостатком быстрой сортировки является недостаточно высокая производительность при сортировке небольших массивов. Впрочем, как пишет Н. Вирт, этим грешат все усовершенствованные методы. Для сортировки небольших массивов лучше использовать один из прямых методов, например, сортировку вставкой.

В отличие от сортировки слиянием, быстрая сортировка не является стабильной. Как уже отмечалось в занятии 16, это значит, что относительный порядок равных элементов после сортировки может быть изменён.

Задача

На пятой индивидуальной интернет-олимпиаде 2009-2010 учебного года, проведённой 27 февраля 2010 года СПбГУ ИТМО, была предложена следующая задача.

Задача C. Календарь

Имя входного файла:	calendar.in
Имя выходного файла:	calendar.out
Ограничение по времени:	2 секунды
Ограничение по памяти:	256 мегабайт

Однажды крупный марсианский магнат Аман Робрамович решил заказать в марсианской типографии «Флатланд-Печать» серию календарей с собственным профилем на каждой странице. Но так как Великий Финансовый Кризис не миновал и Марса, в типографии осталось недостаточное для качественной печати количество чернил, поэтому даты было решено печатать по старинке, на печатной машинке.

Достав из запасов печатную машинку «Лысач», работники типографии обнаружили, что некоторые головки сломаны. Напомним, что на печатных машинках, чтобы напечатать одну цифру требуется исправность соответствующей головки (например, чтобы напечатать цифру «4» необходимо, чтобы головка с цифрой «4» была исправна).

Работники типографии хотят узнать, в каком количестве дат им придется дописывать некоторые цифры карандашом.

Формат входного файла

В первой строке входного файла находится два числа n — количество месяцев на марсе и k — число сломанных головок в печатной машинке ($1 \le n \le 100000$, $0 \le k \le 10$). Во второй строке находится n чисел a_1, a_2, …, a_n — количество дней в месяцах ($1 \le a_i \le 100000$). В следующей строке находится k различных чисел — цифры, которые нельзя напечатать.

Формат выходного файла

В выходной файл выведите количество дат, цифры в которых придется дописывать работникам типографии.

Примеры

calendar.in	calendar.out
12 2 31 28 31 30 31 30 31 31 30 31 30 31 0 3	78
2 2 5 15 0 1	8

Далее в тексте задачи приводилось пояснение к первому примеру в виде распечатки полученного в типографии календаря. Вы можете видеть его, скачав полные тексты задач http://neerc.ifmo.ru/school/io/archive/20100227/problems-individual-20100227.pdf

Решение

Искать количество неправильно напечатанных дат отдельно для каждого месяца, а потом их складывать непродуктивно по времени. Придётся многократно проделывать одну и ту же работу. Ведь если мы знаем, что в предыдущем месяце из a_i дней неправильно распечаталось c_i дат, то в следующем месяце из $a_{i+1} > a_i$ дней будут неправильно распечатаны всё те же c_i дат и ещё какое-то количество дат из промежутка $[a_i + 1, a_{i+1}]$, в записи которых есть «сломанные цифры». Этим и воспользуемся.

Расположим месяцы марсианского календаря в порядке неубывания количества дней в них.

Пусть s — множество «плохих» цифр, а функция

```
Function CountBadNumbers(min,max : Integer; s : BadDigit) : Integer;
```

вычисляет количество чисел из промежутка от min до max включительно, в записи которых есть хотя бы одна цифра из множества s.

Тогда общее количество непечатаемых дат можно найти по следующему алгоритму.

```
c := CountBadNumbers(1,a[1],s);
count := c;
Для i от 2 до N повторять
{
    c := c + CountBadNumbers(a[i-1]+1,a[i],s);
    count := count + c
}
```

Дальнейшая детализация алгоритма не представляет большой сложности. Чтобы узнать, есть ли в записи числа *x* непечатаемые цифры, будем последовательно делить *x* на 10 и проверять, не принадлежит ли остаток множеству s, а потом отбрасывать последнюю цифру, выполняя целочисленное деление на 10. И так, пока *x* не обратится в нуль.

Учитывая, что упорядочивать придётся массив, в котором до 100000 элементов, приходим к выводу, что ни один из прямых методов сортировки не подойдёт. Воспользуемся быстрой сортировкой. Обратим внимание и на то, что полное количество неправильно напечатанных дат может быть очень большим. Наихудший случай будет соответствовать всем 10 сломанным головкам, 10^5 месяцам в году по 10^5 дней. Итого получится $10^5 \cdot 10^5 = 10^{10}$. Для хранения такого числа будем использовать переменную типа int64.

В приведённой ниже программе вместо переменной *c* используется переменная curr.

```
Program Calendar;

Type massiv = array[1..100000] of Integer;
     Digit = 0..9;
     BadDigit = set of Digit;

(* Быстрая сортировка *)
Procedure qsort(var a : massiv; l,r : integer);
Var x,temp   : Integer;
    i,j      : Integer;
Begin
  x :=a[(l+r) div 2];
  i:=l;
  j:=r;
  While i<j Do
  Begin
    While a[i]<x Do
     i:=i+1;
    While a[j]>x Do
     j:=j-1;
    If i<=j Then
    Begin
      temp:=a[i];
      a[i]:=a[j];
      a[j]:=temp;
      i:=i+1;
      j:=j-1
    End
  End;
  If j>l Then
    qsort(a,l,j);
  If i<r Then
    qsort(a,i,r)
End;

(* Есть ли в составе числа x непечатаемые цифры? *)
Function BadNumber(x : Integer; s : BadDigit) : Boolean;
Var d : Digit;
Begin
  BadNumber := false;
  While x>0 Do
  Begin
    d := x mod 10;
    If d in s Then
    Begin
      BadNumber := true;
      Exit //Выход из подпрограммы
    End;
    x := x div 10
  End
End;

(* Количество "плохих" чисел на отрезке [min,max] *)
Function CountBadNumbers(min,max : Integer; s : BadDigit) : Integer;
```

```
Var count : Integer;
Begin
  count := 0;
  While min<=max Do
  Begin
    If BadNumber(min,s) Then
     count := count+1;
    min:=min+1
  End;
  CountBadNumbers := count
End;

Var f        : Text;
    N,k      : Integer;
    i, x     : Integer;
    a        : massiv;
    bad      : BadDigit;
    curr     : Integer;
    count    : Int64;
Begin
  Assign(f,'calendar.in');
  Reset(f);
  Read(f,N,k);

  For i:=1 To N Do
    Read(f,a[i]);

  (* Формируем множество непечатаемых цифр *)
  bad:=[];
  For i:=1 To k Do
  Begin
    Read(f,x);
    bad:=bad+[x]
  End;
  Close(f);

  qsort(a,1,N);

  curr := CountBadNumbers(1,a[1],bad);
  count := curr;
  For i:=2 To N Do
  Begin
    curr := curr + CountBadNumbers(a[i-1]+1,a[i],bad);
    count := count + curr
  End;

  Assign(f,'calendar.out');
  Rewrite(f);
  Write(f,count);
  Close(f);
End.
```

Для тестирования полученной программы можно скачать тесты жюри олимпиады с сайта ИТМО http://neerc.ifmo.ru/school/io/archive/20100227/archive-individual-20100227.rar Как ими пользоваться, смотрите в занятии 13.

Вопросы и задания

1. Как изменить алгоритм быстрой сортировки для сортировки массива по убыванию?
2. Как уже отмечалось, в качестве опорного можно использовать любой элемент массива. Что можно сказать о скорости сортировки уже упорядоченного массива, если в качестве опорного выбрать первый элемент массива? последний? Рассмотрите случаи массива,

упорядоченного по возрастанию, по убыванию. Чем в этом случае хорош наш выбор опорного элемента?

3. Автор алгоритма быстрой сортировки Ч. Хоар предлагал в качестве опорного выбирать случайный элемент массива. При этом у алгоритма будет хорошее среднее время работы, и никакие конкретные входные данные не могут ухудшить его производительность до уровня наихудшего случая. Напишите программу быстрой сортировки при таком выборе опорного элемента. Вспомните, что функция языка Паскаль Random(m) возвращает случайное целое число x в диапазоне $0 \le x < m$.
4. Время работы быстрой сортировки можно улучшить, воспользовавшись тем, что алгоритм сортировки по методу вставок быстро работает для почти отсортированных последовательностей. Можно поступить следующим образом. Когда процедура быстрой сортировки начнёт рекурсивно вызываться для обработки подмассивов, содержащих менее M элементов, в ней не будет выполняться никаких действий, кроме выхода из процедуры. После возврата из процедуры быстрой сортировки, вызванной на самом высоком уровне (QSort(a,1,n)), вызывается процедура сортировки вставками для уже почти отсортированного массива. Р. Седжвик пишет, что выбор точного значения M не критичен. Выбор значения M от 5 до 20 даёт приблизительно одинаково хорошие результаты. Скорость сортировки можно увеличить на 10%. В работах Р. Седжвика и Д. Кнута говорится о том, что наилучшее значение M=9. Напишите программу сортировки описанным методом. Будьте аккуратными при программировании. Имейте в виду, что ошибка в реализации быстрой сортировки будет исправлена при сортировке вставками, но вместо ускорения сортировки вы можете получить её замедление.
5. Вернитесь к задаче «Большой квадратный кусок» занятия 13. Замените в её решении алгоритм сортировки на один из вариантов быстрой сортировки. Сравните время работы программы на тестах большой размерности при использовании сортировки вставками, сортировки Шелла и быстрой сортировки. Сделайте вывод о применимости различных методов сортировки для решения этой задачи. Какие ещё методы сортировки можно было применить в задаче «Календарь»?

Литература

1. Н. Вирт. Алгоритмы и структуры данных. — СПб.: Невский диалект, 2008.
2. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ. Второе издание. — Москва, Санкт-Петербург, Киев. Издательство «Вильямс», 2010.
3. Д. Кнут. Искусство программирования для ЭВМ. Том 3. Сортировка и поиск. Издание 3. — Издательский дом «Вильямс», 2005.
4. Р. Седжвик. Фундаментальные алгоритмы на C++. Части 1 – 4. — Москва, Санкт-Петербург, Киев. Издательство «DiaSoft», 2001.

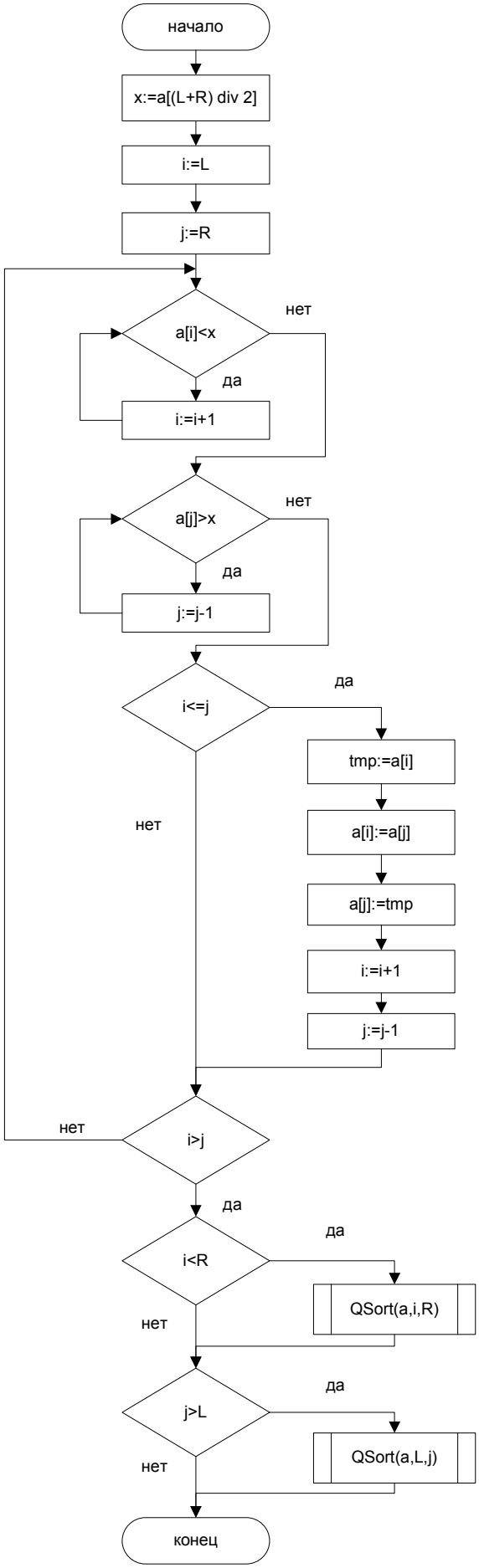

Рисунок 1. Процедура QSort(a, L, R)