Пример разработки программы с использованием подпрограмм

На предыдущих занятиях мы уже говорили о том, что реальные задачи столь сложны, что для их решения приходится разбивать задачу на подзадачи, последовательно выкристаллизовывая алгоритм. Для решения подзадач составляют подпрограммы. Это позволяет не запутаться в своей собственной программе, облегчает её отладку (поиск ошибок в ней) и модификацию. Иногда для улучшения программы достаточно заменить код всего одной из подпрограмм, не изменяя основного алгоритма.

Покажем это на примере.

На третьей командной олимпиаде цикла интернет-олимпиад[1] для школьников сезона 2010-2011 года, проводимой жюри всероссийской командной олимпиады школьников по программированию, предлагалась задача «Большой квадратный кусок».

Большой квадратный кусок

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

На день рождения Пете подарили прямоугольный торт со сторонами n и m. Петя любит прямоугольники, но еще больше ему нравятся квадраты. Естественно, он хотел разрезать свой торт на много одинаковых квадратных кусочков. К сожалению, его маленький брат Лёша добрался до торта раньше и уже сделал несколько разрезов, параллельных сторонам торта.

Петя сильно расстроился. Лёша хочет его утешить. Для этого он решил из одного из получившихся кусков вырезать квадратный кусочек максимально возможного размера. Для начала ему нужно узнать, какого максимального размера квадрат можно вырезать из получившихся кусков.
Помогите Лёше.

Формат входного файла

Первая строка входного файла содержит три целых числа n, m и k ($1 \le m, n \le 10^9$, $0 \le k \le 10^5$). Следующие k строк содержат описания разрезов. Каждый разрез задается числами t и v. Введем декартову систему координат так, чтобы один из углов торта имел координаты $(0, 0)$, а другой — (n, m). Тогда $t = 0$ означает разрез вдоль прямой $x = v$ $(0 \le x \le n)$, а $t = 1$ — разрез вдоль прямой $y = v$ $(0 \le y \le m)$.

Формат выходного файла

Выведите единственное число — сторону квадратного куска максимального размера, который можно вырезать из полученных после разрезания торта кусков.
Попытаемся решить эту задачу.

Чтобы найти наибольший квадратный кусок, надо найти наиболее отстоящие друг от друга *соседние* разрезы по осям x и y. Это даст наибольшую длину dx и наибольшую ширину dy полученных кусков, т.е. размеры наибольшего прямоугольного куска. Ответом на вопрос задачи будет наименьший из этих двух размеров: dx или dy.

Алгоритм решения задачи получается следующим.

1. Ввести данные о разрезах в массивы x и y с подсчётом количества элементов в этих массивах.

[1] См. http://neerc.ifmo.ru/school/io/

2. Отсортировать массив x по возрастанию[2].
3. Отсортировать массив y по возрастанию.
4. Найти максимальную разность dx между соседними элементами в массиве x.
5. Найти максимальную разность dy между соседними элементами в массиве y.
6. Найти минимальное из чисел dx и dy.

Приступим к реализации намеченного плана. Определим тип данных для массивов

```
Const MAXK=100000;
Type massiv = array[0..MAXK] of
Integer;
```

Для ввода данных разработаем процедуру

```
Procedure Input(out x,y :
massiv; out Nx, Ny : Integer);
```

Здесь Nx и Ny — количества элементов в массивах x и y соответственно. Все параметры этой процедуры будут выходными.

Первыми элементами массивов будут нули — координаты угла торта, вторыми элементами — координаты угла по диагонали, т.е. $x[2] = n, y[2] = m$. Остальные элементы берём из координат разрезов. Блок-схему алгоритма смотрите на рис. 1.

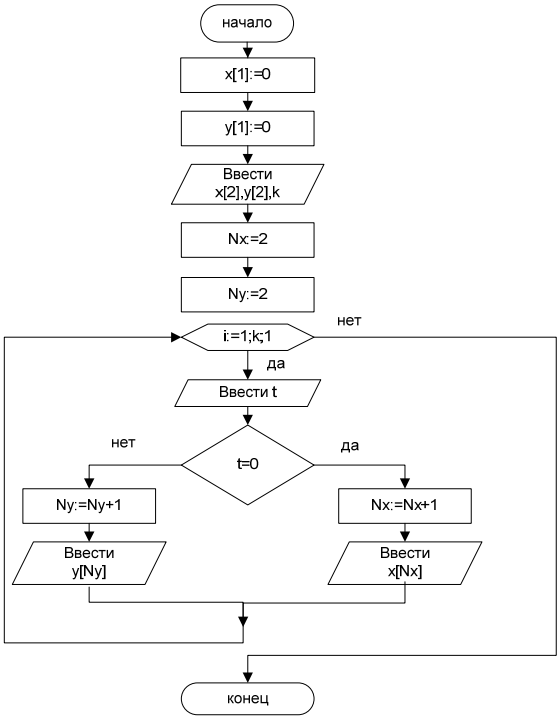

Рисунок 1. Процедура Input

Для поиска максимальной разности между соседними элементами отсортированного массива разработаем функцию

```
Function MaxDiff(const a : massiv; N : Integer) : Integer;
```

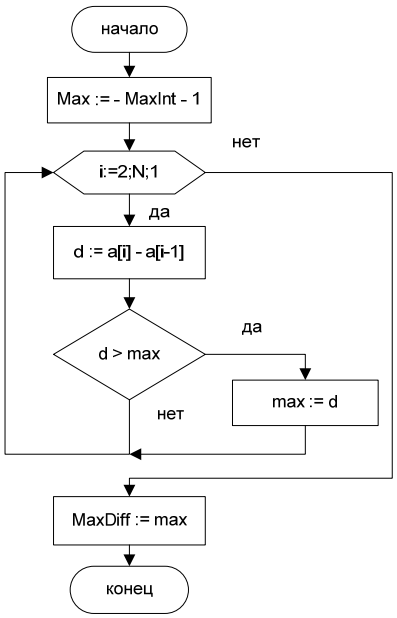

Рисунок 2. Функция MaxDiff

Массив в этой подпрограмме изменяться не будет. Его копирования при передаче также следует избегать, ведь в нём может быть до 100000 элементов. Поэтому передавать его в подпрограмму будем как параметр-константу. N — это количество элементов массива. Алгоритм функции представлен на рис. 2. Обратите внимание на команду

```
Max := - MaxInt - 1
```

Константа MaxInt равна максимальному числу типа Integer. Модуль минимального элемента этого же типа на 1 больше. Таким способом мы получили минимально возможное число типа Integer. Любое значение указанного типа будет не меньше только что вычисленного значения Max.

Теперь о сортировке. На занятии 7 было рассмотрено несколько алгоритмов сортировки. Попробуем для решения задачи «Большой квадратный кусок» использовать сортировку прямым включением с использованием барьерного элемента. Именно для него мы описали массив с индексами, начинающимися с нуля, хотя пока нигде не использовали нулевого элемента массива. Блок-схему сортировки рассматривать здесь не будем. Смотрите рис. 4 в занятии 7.

---

[2] Заметим, что сортировать можно было и по убыванию. Важно лишь то, что координаты соседних разрезов после сортировки должны оказаться рядом друг с другом. Но алгоритм функции MaxDiff при сортировке по убыванию пришлось бы изменить. Подумайте, как.

Итак, получаем следующую программу на FreePascal.

```
{$mode delphi}
Program maxpiece;
Const MAXK=100000;
Type massiv = array[0..MAXK] of Integer;

(****************************************
 *  Сортировка массива прямым включением  *
 *  N - количество элементов массива      *
 ****************************************)

Procedure Sort(Var a : massiv; N : Integer);
Var i,j,x : Integer;
Begin
(* Сортировка прямым включением *)
  For i:=2 To N Do
  Begin
    x:=a[i];
    a[0]:=x; //Установка барьера
    j:=i;
    While x < a[j-1] Do
    Begin
     a[j]:=a[j-1];
     j:=j-1
    End;
    a[j]:=x
  End
End;


(*********************************************************************
Нахождение максимальной разности между соседними элементами массива
N - количество элементов массива
*********************************************************************)

Function MaxDiff(const a : massiv; N : Integer) : Integer;
Var max : Integer;
    i,d : Integer;
Begin
  max := - MaxInt - 1;
  For i:=2 To N Do
  Begin
    d:=a[i]-a[i-1];
    If d>max Then
      max:=d
  End;
  MaxDiff := max
End;

(****************************
 *        Ввод данных         *
 ****************************)
Procedure Input(out x,y : massiv; out Nx, Ny : Integer);
Var f   : Text;
    i,k : Integer;
    t   : Byte;
Begin
  Assign(f,'maxpiece.in');
  Reset(f);
  x[1]:=0;
  y[1]:=0;
  Read(f,x[2],y[2],k);
  Nx:=2;
```

```
  Ny:=2;
  For i:=1 To k Do
  Begin
    Read(f,t);
    If t=0 Then
    Begin
      Nx:=Nx+1;
      Read(f,x[Nx]);
    End
    Else Begin
           Ny:=Ny+1;
           Read(f,y[Ny])
         End
  End;
  Close(f);
End;

(*****************************************************************)

Var x, y    : massiv;
    f       : Text;
    i       : Integer;
    Nx,Ny   : Integer; (* Количества элементов в массивах x и y *)
    dx, dy  : Integer;
Begin
  Input(x,y,Nx,Ny);

  Assign(f,'maxpiece.out');
  Rewrite(f);

  Sort(x,Nx);
  Sort(y,Ny);

  //Максимальное расстояние между соседними разрезами
  dx:=MaxDiff(x,Nx); //по оси x
  dy:=MaxDiff(y,Ny); //по оси y

  //Наибольший кусок имеет размер dx на dy. Выбираем меньшее
  If dx<dy Then
    Write(f,dx)
  Else
    Write(f,dy);

  Close(f)
End.
```

Протестируем полученную программу. Для этого скачаем файл http://neerc.ifmo.ru/school/io/archive/20101023/archive-basic-20101023.rar В папке maxpiece\tests расположены тесты для проверки решения этой задачи. Чтобы ими воспользоваться

1. скопируем в папку с нашей программой файл 01
2. переименуем его в maxpiece.in
3. Запустим на выполнение нашу программу.
4. Сравним содержимое maxpiece.out с содержимым файла 01.a из папки с тестами (файлы с расширением *a* содержат правильные ответы), а также оценим, хотя бы «на глаз», время работы программы.

Мы убедимся, что наша программа выдаёт правильные ответы, но время её работы не всегда удовлетворяет ограничению в 2 секунды. Например, на тесте 18 или 27 оно гораздо больше. Почему так происходит, догадаться несложно. Мы использовали медленный алгоритм сортировки. Значит, его надо заменить другим. К счастью, для этого не надо переписывать заново всю программу. Достаточно просто заменить реализацию процедуры Sort. Из рассмотренных нами в

занятии 7 алгоритмов сортировки самой быстрой является сортировка Шелла с использованием последовательности Седжвика. Ею и воспользуемся. Блок-схема сортировки Шелла представлена на рис. 8 в занятии 7. Однако там остался нерешённым вопрос о вычислении последовательности Седжвика. Вернее, этот вопрос предлагался для самостоятельного решения. Разберёмся в нём подробнее.

Напомню, что последовательность Седжвика задаётся формулой

$$h_s = \begin{cases} 9 \cdot 2^s - 9 \cdot 2^{\frac{s}{2}} + 1, \text{если } s \text{ чётно} \\ 8 \cdot 2^s - 6 \cdot 2^{\frac{s+1}{2}} + 1, \text{если } s \text{ нечётно} \end{cases}$$

Здесь нумерация элементов последовательности смещений h_s начинается с нуля, причём 0 условно считается чётным числом.

Эта последовательность быстро нарастает. Уже при $s = 28$ значение h_s не умещается в 32-битной переменной типа Integer. Можете проверить это, например, воспользовавшись стандартным калькулятором Windows и приведённой выше формулой даже без всякого программирования. Получите $h_{28} = 2415771649$. Это больше, чем MaxInt. Смещений от h_0 до h_{27} достаточно для эффективной сортировки массивов с количеством элементов до $3h_{28}$. Это больше 7 миллиардов. Несложно было бы вручную вычислить указанные 28 значений и создать из них массив h. Можно также вычислять этот массив в ходе процедуры сортировки, что, правда, несколько снизит скорость работы подпрограммы сортировки. Пойдём вторым путём.

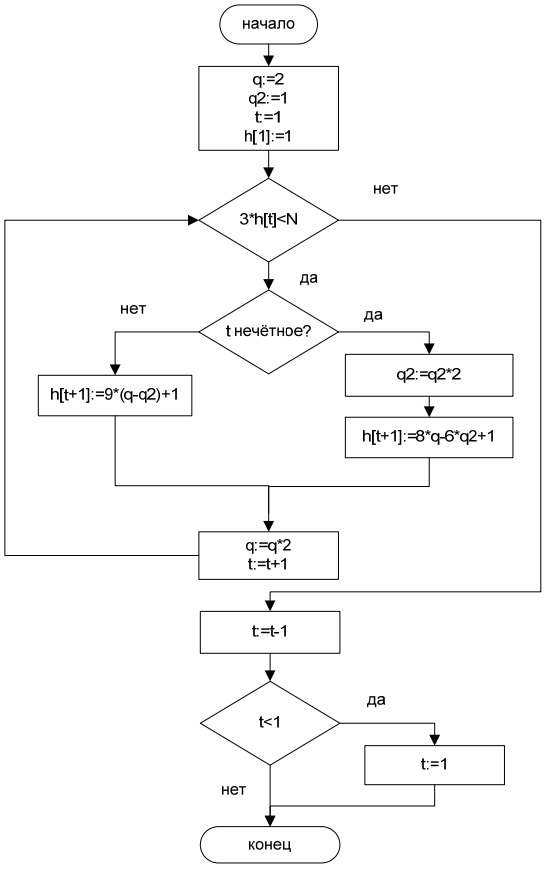

Рисунок 3. Получение последовательности Седжвика

На рис. 3 представлен алгоритм вычисления смещений h с использованием последовательности Седжвика. В отличие от изложенного выше, здесь нумерация элементов массива h начинается с единицы. В конце работы алгоритма переменная t равна индексу того элемента, с которого надо начинать использование массива смещений в полном соответствии с алгоритмом рис. 8 занятия 7. Переменная q используется для вычисления 2^t, переменная q2 — для вычисления $2^{\frac{t-1}{2}}$.

Теперь осталось только объединить алгоритмы рис. 3 этого занятия и рис. 8 занятия 7, создав новую версию процедуры Sort, и заменить ею одноимённую процедуру уже написанной программы. Сделайте это сами и убедитесь, что скорость решения задачи значительно возрастёт. При этом вы можете создать отдельную процедуру для вычисления массива h, а можно эти вычисления провести в самой подпрограмме Sort. Возможно, вам понадобится стандартная логическая функция odd(x), возвращающая значение true, если целое число x является нечётным. Хотя можно обойтись и без неё.

Вопросы и задания

1. Зачем в конце алгоритма рис. 3 проводить сравнение значения переменной t с единицей?
2. Как, не проводя точного вычисления h_{28}, убедиться только при помощи формулы, задающей последовательность Седжвика, и устных вычислений, что $h_{28} > MaxInt$?
3. Можно ли при описании формальных параметров процедуры Input использовать модификатор **var** вместо **out**? При описании процедуры Sort использовать **out** вместо **var**?

4. Закончите разработку программы, о которой шла речь в тексте занятия, с использованием сортировки Шелла. Протестируйте полученную программу. Сравните время её работы с временем работы первого варианта решения этой задачи, использующего сортировку прямым включением. Нужна ли в новой версии программы нумерация элементов массивов x и y с нуля?
5. Реализуйте решение этой задачи, используя сортировку по убыванию вместо сортировки по возрастанию. Какие изменения пришлось вам внести?

Литература

1. Условия задач третьей командной олимпиады цикла интернет-олимпиад для школьников сезона 2010-2011 года (23 октября 2010 года, базовая номинация) http://neerc.ifmo.ru/school/io/archive/20101023/problems-basic-20101023.pdf
2. Тесты и решения жюри для задач третьей командной олимпиады школьников (23 октября, базовая номинация) http://neerc.ifmo.ru/school/io/archive/20101023/archive-basic-20101023.rar
3. Д. Кнут. Искусство программирования для ЭВМ. Том 3. Сортировка и поиск. Издание 3. — Издательский дом «Вильямс», 2005.