Выборка из миллиарда

Условие задачи. Из числового интервала от единицы до миллиарда выбираются случайным образом без повторений миллион чисел и записываются в файл. Необходимо за приемлемое время выяснить, какое наименьшее число отсутствует в файле. Использовать массивы или иные структуры данных, их заменяющие, запрещается.

Изюминка данной задачи заключается в том, что не сразу понятно, в чем собственно проблема. Почему эта задача сложна? Почему нельзя использовать какой-нибудь простой, лобовой метод, ведь условие выглядит достаточно просто. Так иногда бывает. Кажется, что все хорошо и просто, поэтому требуются специальные усилия для того, чтобы хотя бы понять с чем мы столкнулись. В такой ситуации, чтобы увидеть реальное положение дел, можно придумать любое решение, какое придет в голову, и посмотреть, почему оно не будет работать.

Разрешите предложить вам следующий вариант: упорядочим числа в файле в порядке возрастания. Алгоритм упорядочения по возрастанию никак нельзя назвать логически сложным, сортировка — хорошо изученный процесс. После упорядочения будет достаточно одного прохода файла, в ходе которого легко обнаружить первое число, чей сосед справа отличается от него более чем на единицу. Тогда найденное число из файла + единица и будет искомым. При таком подходе задача действительно выглядит элементарной. Теперь подумаем, почему это не сработает. Пусть, будем упорядочивать файл наибыстрейшим образом. Какой бы мы при этом не выбрали алгоритм придется выполнить несколько миллиардов файловых операций, а если вспомнить, что файловые операции выполняются медленно, то становится понятно в чем изюминка задачи — *нельзя читать файл слишком много раз, надо покончить с этим делом за несколько проходов*. Если вопроса, почему эта задача сложная, не осталось, приступим к ее решению.

В поиске решения задачи мы применим метод аналогии. Его суть заключается в двух действиях:

1. Вспомним задачу с похожим условием, но уже решенную, и решенную достаточно удачно.

2. Попытаемся переложить известное решение на данную задачу.

Наша задача очень похожа на задачу поиска корня заданного числа. В задаче о корне имеется числовой интервал, в котором очень много чисел, и среди них необходимо найти одно, обладающее определенными свойствами (оно является корнем). Есть правда очень существенное отличие. Числа интервала в задаче о корне расположены в порядке возрастания, в нашей же задаче они расположены совершенно хаотично. Но может быть это окажется не очень существенным. Если для вас задача о корне неизвестна, тогда приведем текст ее решения в листинге

Листинг

```
program example;
 uses crt;
 var
  a,x,l,p,e:real;
begin
 clrscr;
```

```
 read(a,e);
 l:=0;
 p:=a;
 repeat
  x:=(l+p)/2;
  if x*x>a then p:=x else l:=x;
 until (p-l<e);
 write((p+l)/2);
end.
```

Мы не будем подробно разбирать алгоритм этой программы, она достаточно прозрачна, и вы можете разобраться в ней сами. Для нас важен принцип. А принципиально здесь происходит следующее: на каждом шагу главного цикла интервал поиска разбивается на два, затем определяется новый интервал, в котором далее надо вести поиск корня. Попробуем использовать этот принцип для нашей задачи.

Поделим миллиардный интервал на два интервала по 500 миллионов. Где может находиться искомое число? Очевидно, в первом, так как 500-миллионный интервал миллионом чисел не заполнить. Теперь поделим первый 500-миллионный интервал на два и т. д. Рано или поздно заключение о том, что интервал, содержащий искомое число, будет первым, окажется несправедливым. Поэтому давайте для любой пары интервалов разработаем более универсальный метод.

Заметим, что в нашем простом способе определения мы делали вывод на основании сравнения того, сколько чисел должен содержать интервал, и сколько он их содержит на самом деле. Разовьем эту идею, она кажется перспективной. Предположим, на текущий момент есть интервал [3000, 3999]. Очевидно для того, чтобы быть полностью заполненным, он должен содержать 1000 чисел (напомним, что числа в файле не повторяются, иначе чисел в интервале может оказаться существенно больше 1000 и при этом интервал будет не заполненным). Мы знаем, что все они не меньше 3000 и не больше 3999. Это легко проверяемый факт. Достаточно один раз прочитать файл и подсчитать, сколько в нем чисел, укладывающихся в данный интервал. Если окажется, что чисел меньше чем длина интервала, то, следовательно, в нем есть число, не записанное в файл.

Процесс разбиения интервала на два следующих необходимо завершать, когда в текущем интервале останется два числа, тогда следующее разбиение даст два интервала, в одном из которых будет одно число, а в другом ни одного. Первое число этого другого интервала и есть искомое. Запишем алгоритм. Его главные объекты —два счетчика. Первый считает числа, входящие в первый интервал, второй — числа, входящие во второй интервал.

```
Пока длина интервала больше единицы, делать
  Разбить интервал на два
    Читать файл и для каждого числа, прочитанного из файла, делать:
      Если число принадлежит первому интервалу, то первый счетчик
      увеличивается на единицу
      Иначе второй счетчик увеличивается на единицу
    Текущий интервал — это интервал, значение счетчика которого
    меньше длины интервала.
```

Записанный алгоритм неплохо выражает общую идею, но в нем много неопределенных понятий. Самый главный вопрос — так ли важно делить именно на два интервала, может быть лучше делить на 3 или 5. Выигрыш, который мы можем получить, изменив количество интервалов, заключается в количестве необходимых прочтений файла,

а как вы помните, именно многократное чтение файла и создает проблему для решения задачи. Посмотрим, как быстро мы получим решение с разным количеством интервалов.

Для миллиарда количество шагов будет равно показателю степени двойки, при которой степень превысит миллиард. При делении интервала на 10 мы получим количество шагов, равное показателю степени 10, при котором степень превысит миллиард. Таким образом, действительно увеличение количества интервалов дает выигрыш во времени. Но, рассуждая подобным образом, мы приходим к противоречию, так как наша цепочка рассуждений быстро приведет к необходимости разбить миллиард на миллиард интервалов. Это, конечно, бессмыслица и надо подумать, откуда она появилась.

Во-первых, мы забыли про память. Чем больше интервалов, тем большее количество счетчиков мы должны помнить, а это затраты на память. Кстати сейчас мы можем сформулировать одно общее правило.

Правило компенсации

Если вам удалось улучшить какую-то характеристику вашей программы, то вы должны ожидать, что какая-то другая характеристика станет хуже.

Это правило не имеет характера закона. В принципе можно так улучшить программу, что все ее характеристики станут лучше. Она станет быстрее работать, в то же время уменьшатся затраты на память, ее текст станет короче, понятнее и т. д. Это вполне возможно, но чаще всего такое всеобщее улучшение будет означать, что предыдущий вариант программы был совсем плох. А чем лучше ваша программа, чем изощреннее алгоритм и чем эффективнее использованы средства языка, тем полученное правило действует точнее.

В нашей задаче, увеличивая количество интервалов, мы теряем в памяти, таким образом, количество интервалов ограничено объемом имеющейся оперативной памяти. Но на самом деле мы теряем и в скорости! Для того чтобы перейти на следующий шаг, необходимо найти счетчик, значение которого меньше длины текущего интервала, а если массив будет велик, то его обход также займет время и совершенно не очевидно, что эти затраты времени можно игнорировать.

Конечно, для идеального решения необходимо взвесить все выигрыши и потери, а также определиться, какое количество интервалов выгодно. Но точное решение этого вопроса вряд ли возможно. Во-первых, для этого необходимо очень точно знать скорость выполнения операций процессором, и, во-вторых, предстоит сравнивать величины, имеющие различную природу (это скорость счета и память), а корректно сравнивать можно только величины одной природы. Поэтому не будем искать идеальное решение (так как непонятно, что это такое) и ограничимся достаточно хорошим. При количестве интервалов, равном 10, нам потребуется всего 12 шагов, то есть 12 прочтений файла, что немного, для миллионного файла такая работа займет приемлемое время.

Следующая неясность алгоритма — это разбиение интервала. Как именно это сделать? Вопрос не слишком прост, так как текущий интервал не только уменьшается в 10 раз, но и достаточно произвольно скачет по исходному миллиарду (искомое число может оказаться где угодно).

Ранее было заявлено два важных утверждения относительно интервала: его длина переменна, его положение переменно. Следовательно, для задания интервала нужны две величины: длина и число, с которого он начинается, левая точка.

Проблем с расчетом длины нет. На первом шаге длина равна миллиарду, на каждом последующем длина уменьшается в десять раз. Левая точка первого шага — это ноль. Обозначим текущую левую точку как НАЧАЛЬНАЯ. Попробуем посчитать, куда она переместится после очередного шага. Пусть новым интервалом будет интервал с номером $L = 1, 2, 3, ..., 10$. Обозначим длину предыдущего интервала через ДЛИНА. Тогда Новую НАЧАЛЬНУЮ точку можно вычислить по формуле:

Новая НАЧАЛЬНАЯ = (L – 1)*(ДЛИНА / 10)

На самом деле эта формула несколько неверна. Она вычисляет новое положение НАЧАЛЬНОЙ точки относительно НАЧАЛЬНОЙ точки текущего интервала. А еще надо учесть, что все числа левее предыдущей НАЧАЛЬНОЙ точки уже исключены из рассмотрения. Это можно учесть следующим образом:

Новая НАЧАЛЬНАЯ = Предыдущая НАЧАЛЬНАЯ + (L – 1)*(ДЛИНА / 10)

Следующий неясный момент заключается в процедуре определения, к какому из интервалов принадлежит очередное прочитанное число. Рассмотрим первый шаг с исходным интервалом. В каждый из внутренних интервалов может входить 100 миллионов чисел. В первый входят числа от нуля до 99 999 999. И вот тут небольшой момент срабатывания интуиции. Заметим, что если любое число из этого интервала мы разделим нацело на 100 000 000, то результатом будет ноль. Точно также любое число из интервала 100 000 000 до 199 999 999, деленное на 100 миллионов, даст 1 и т. д. То есть если мы число разделим на 100 миллионов и прибавим 1, то как раз и получим номер интервала. Запишем это в общем виде, для чего введем обозначения: ДЛИНА — длина интервала, ЧИСЛО — число, прочитанное из файла и ИНДЕКС — индекс вложенного интервала.

ИНДЕКС = Целая часть от (ЧИСЛО / ДЛИНУ) + 1

Эта формула полезна для понимания того, что нам требуется, но она неточна. Попробуйте проверить ее действие хотя бы для второго шага. Ошибка будет в следующем. Предположим, в файле есть число пятьсот миллионов. На первом шаге (деление на 100 миллионов) это число по нашей формуле даст индекс 5, а на втором это же число даст индекс 50, то есть несуществующий, а на третьем это же число даст индекс 500 и т. д. Отсюда следует, что перед тем, как вычислять индекс, надо выяснить, входит ли прочитанное число в интересующий нас интервал, если сказать более точно, то необходимо выяснить, находится ли это число между НАЧАЛЬНОЙ точкой и точкой НАЧАЛЬНАЯ + ДЛИНА интервала. Теперь наша формула превращается в небольшой алгоритмический фрагмент.

Если ЧИСЛО >= НАЧАЛЬНОГО и ЧИСЛО < НАЧАЛЬНОЕ + ДЛИНА

То ИНДЕКС = Целая часть от (ЧИСЛО / ДЛИНУ) + 1

Это еще не все уточнения, но прежде чем идти дальше сделаем небольшое замечание о методе наших рассуждений. В этой задаче мы выводим уже вторую формулу. Обратите внимание, как это делается. Во-первых, мы записываем формулу расчета из самых простых и очевидных соображений, чаще всего исходя из какого-то простого примера,

например исходя из начального состояния данных. Затем мы берем более сложный пример, и если формула отказывается работать, то корректируем ее с учетом новой, полученной информации, потом ищем следующую ошибку и т. д. Конечно, может возникнуть вопрос, откуда мы знаем, что ошибки закончатся? Точного, строгого ответа на данный вопрос дать невозможно, но есть одно очень сильное соображение. Если один пример считается правильно, а другой нет, то между этими примерами есть какое-то принципиальное отличие, не сводящееся к разнице в количественном значении. В наших двух примерах, рассмотренных ранее, различие следующее: миллиард — это начальный интервал, второй интервал в 100 миллионов — это интервал, впервые полученный расчетным способом. Такова принципиальная разница. Примеров, между которыми есть принципиальная разница, не может быть сколь угодно много, а следовательно, ошибки должны закончиться. Конечно термин "принципиальная разница" не очень ясен, но будем надеяться, что интуитивного понимания окажется достаточно, все равно дать его точное определение не получится.

А теперь вернемся к формуле. Мы рассуждаем об интервале, но интервалов у нас на каждом шаге два. Это большой интервал, который мы разбиваем на меньшие, и сами меньшие. Впрочем раньше уже мелькало понятие вложенного интервала, но как-то четко об этих различиях ничего не было сказано. Это нормально. Практически невозможно сразу дать точные определения всем используемым понятиям, но сейчас это сделать необходимо, *мы выходим на конкретные формулы, и неточности в понятиях приведут к неточности в формулах*. И такая неточность в нашей формуле уже появилась. В фрагменте приведенном ранее дважды используется ДЛИНА, а что при этом имеется ввиду? В команде ЕСЛИ имеется ввиду длина текущего интервала в котором осуществляется поиск, а в формуле имеется ввиду длина интервала разбиения (давайте далее называть его так), который в 10 раз меньше текущего. Поэтому уточним формулу следующим образом:

Если ЧИСЛО >= НАЧАЛЬНОМУ и ЧИСЛО < НАЧАЛЬНОЕ + ДЛИНА

То ИНДЕКС = Целая часть от (ЧИСЛО / ДЛИНУ РАЗБИЕНИЯ) + 1

А в программе ДЛИНУ РАЗБИЕНИЯ будем вычислять, как десятую часть от ДЛИНЫ. И это еще не все. Предположим, что текущий интервал включает в себя число 500 миллионов, а длина текущего интервала например 1000. Ничего невозможного в этом нет. То есть число 500 миллионов окажется вполне допустимым, а индекс, который оно породит, будет 500 миллионов деленное на 1000, что, конечно же, ошибка. И эта ошибка заключается в том, что индекс надо считать также с учетом числового значения НАЧАЛЬНОЙ точки. В конечном итоге формула приобретает такой вид:

Если ЧИСЛО >= НАЧАЛЬНОМУ и ЧИСЛО < НАЧАЛЬНОЕ + ДЛИНА

То ИНДЕКС = Целая часть от ((ЧИСЛО - НАЧАЛЬНАЯ)/

ДЛИНУ РАЗБИЕНИЯ) + 1

Конечно, трудно предположить, что все эти рассуждения могли быть проведены до того, как написана программа. На самом деле разумно написать программу с первоначальным вариантом формулы, затем в процессе отладки находить ошибки и, отталкиваясь от них, проводить дальнейшие рассуждения. Так всегда легче. Редко кто может провести полный анализ всех проблем до написания программы.

Теперь все логические проблемы решены и можно сесть за написание программы, но есть еще одно серьезное *но*. В условии задачи речь идет о миллиардном интервале. Как

мы будем работать с такими длинными целыми? Конечно, есть в языках программирования тип "длинное целое", но все ли арифметические операции с ним работают корректно? Еще одна проблема связана с тем, что нам нужен миллион целых, случайных и неповторяющихся. Их генерация — это отдельная математическая и алгоритмическая проблема, к нашей задаче не имеющая прямого отношения. Однако сказать, что получение файла данных не наша проблема, мы тоже не можем, так как необходимо проверить программу.

Если посмотреть на полученный алгоритм, то видно, что длина интервала для него не имеет принципиального значения. Поэтому, будет правильно, если для отладки мы используем достаточно большой, но не миллиардный интервал. Можно даже ограничить себя типом "целое". В общем, ограничим себя проверкой логики. Кто хочет, может заняться решением всех технических проблем, связанных с большими числами. Итак, мы будем работать с 9000 числами выбранными из 10 000–го интервала.

Листинг

```
program example;
 uses crt;
 var
  f:file of integer;
  long, a, x, n, i: integer;
  s:array[1..10] of integer;
begin
 clrscr;
 assign(f,'dat');
 long:=10000;
 x:=0;
 for n:=1 to 4 do
  begin
   reset(f);
   {Инициализация массива счетчиков}
   for i:=1 to 10 do s[i]:=0;
   while not eof(f) do
    begin
     read(f,a);
     {Проверка на вхождение в текущий интервал}
     if (a>=x) and (a<x+long) then
       begin
        {Расчет индекса интервала разбиения}
        i:=((a-x) div (long div 10))+1;
        s[i]:=s[i]+1;
       end;
    end;
   for i:=1 to 10 do write(s[i],' ');
   writeln;
   readkey;
   i:=1;
   {Поиск первого незаполненного интервала разбиения}
   while s[i]>=(long div 10) do i:=i+1;
   writeln('i=',i);
   {Переход к новому интервалу}
   x:=x+(i-1)*(long div 10);
   long:=long div 10;
  end;
  write(x);
end.
```

И последнее. Как заполнить нужный файл? Мы решили, что исходная структура данных не очень велика, поэтому можно воспользоваться простым методом. Будем заполнять файл так:

Пока не записано нужное количество чисел делать

```
  Генерируем случайное число
  Проверяем есть ли оно в файле
  Если нет то
    Записываем его в файл
    Счетчик записанных чисел увеличиваем на единицу.
```

В листинге приведена программа заполнения файла.

Листинг

```
program example;
 uses crt;
 var
  f:file of integer;
  a,b,n:integer;
  flag:boolean;
begin
 randomize;
 clrscr;
 assign(f,'dat');rewrite(f);
 for n:=0 to 1999 do
  write(f,n);
 while n<6000 do
  begin
    a:=random(10000);
    reset(f);
    flag:=true;
    while not eof(f) do
     begin
      read(f,b);
      if b=a then flag:=false;
     end;
    if flag then
     begin
      n:=n+1;
      clrscr;
      gotoxy(40,12);write(n);
      seek(f,filesize(f));
      write(f,a);
     end;
  end;
end.
```

В заключение. Рассмотренная задача, пожалуй, единственная в книге, где так много времени и усилий тратится на вывод формул, поэтому, пользуясь моментом, обратим ваше внимание на одну важную проблему. В выводимых формулах мы считаем индексы. Обратите внимание на поправку +1. Интересно, что такие поправки при расчете индексов появляются достаточно часто. Либо +1, либо –1. Их появление всегда объясняется логикой решения, но почему-то они часто упускаются неопытными программистами. Возможно дело в том, что эти поправки порождаются какими-то мелкими и более тонкими эффектами, чем главная идея решения, может быть здесь срабатывает какой-то психологический эффект. Но в любом случае полезно запомнить, что при *расчете*

индексов, независимо от того, для чего он считается, в конечной формуле может появиться поправка в единицу.