Классификация олимпиадных задач

Классификация олимпиадных задач весьма условна, причем часто задачу можно отнести к нескольким разделам, а реально она только в одном.

1. Рекуррентные соотношения и динамическое программирование
2. Сортировки и последовательности
3. Переборные задачи
4. Структуры данных
5. Задачи на графах
6. Арифметика
7. Геометрия
8. Поиск
9. Разное

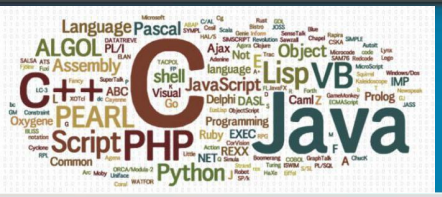
Олимпиады на поиск эффективных алгоритмов

Региональная дистанционно-очная олимпиада по программированию **(март-апрель, СФ БашГУ)**

Республиканская олимпиада по программированию **(октябрь, СФ БашГУ)**

CODEFORCES β
Sponsored by Telegram

Соревнования и олимпиады по спортивному программированию **(постоянно)**



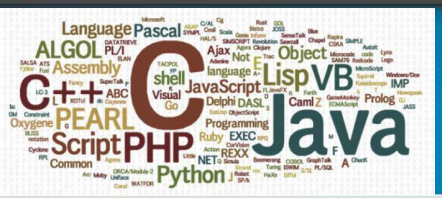
Примеры задач на поиск эффективных решений

Задача 1. Тройки чисел будем называть друзьями, если сумма всех делителей (исключая сами числа) любых двух чисел из тройки равна третьему числу. Например, числа 238, 255 и 371 – друзья: сумма делителей числа 238 равна 194, сумма делителей числа 255 равна 177, а сумма делителей числа 371 равна 61; получаем: 194+177=371, 194+61=255, 177+61=238.

Необходимо найти тройки чисел-друзей в диапазоне от 1 до *N*. Преимущественным условием работы программы должна выступать скорость ее работы.

Входные данные: В первой строке входного файла INPUT.TXT записано число *N*, причем $1 \le N \le 10^6$.

Выходные данные: В строки выходного файла OUTPUT.TXT вывести тройки чисел-друзей, каждая тройка чисел – с новой строки. Числа в тройке разделены двумя пробелами. При выводе необходимо исключить повторы (комбинации из одних и тех же чисел, входящих в тройку).



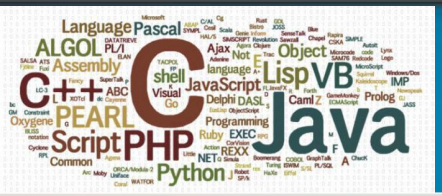
Примеры задач на поиск эффективных решений

Неэффективное решение

```
program N1;
{$APPTYPE CONSOLE}
uses SysUtils, DateUtils;
var i,j,n,s1,s2,s3,ch: longint;
      t1,t2: TDateTime;
      f: text;

function SD(x: longint): longint;
var i,j,s: longint;
begin
  j:=x div 2;
  s:=1;
  for i:=2 to j do if x mod i = 0 then s:=s+i;
  SD:=s;
end;

begin
  t1:=Now;
  assign(f,'input.txt');
  reset(f); read(f,n); close(f);
 assign(f,'output.txt'); rewrite(f);
```

```
for i:=1 to n-1 do
    begin
      s1:=SD(i); j:=i+1;
      while j<=n do
        begin
          s2:=SD(j);
          ch:=s1+s2;
          if (ch>i) and (ch>j) then
            begin
              s3:=SD(ch);
              if (s3+s2=i) and (s3+s1=j) and (i<>ch) and (j<>ch) then
                 begin writeln(f, i,'  ',j,'  ',ch); j:=n; end;
            end;
          j:=j+1;
        end;
    end;
  close(f);
  t2:=Now;
  writeln('Time ',MilliSecondsBetween(t2,t1)/1000:5:3,' s.');
  write('Program is complete');
  readln;
end.
```

Такая простая проверка всех чисел в диапазоне для $N \geq 10^4$ совершенно неприемлема.

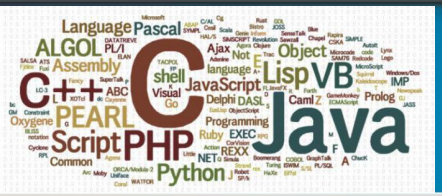


Примеры задач на поиск эффективных решений

Направления улучшения решения

1. Можно существенно улучшить функцию *SD*. В ней используется избыточный диапазон значений для поиска делителей числа x – проверяется диапазон до половины значения числа, т.е. до $x/2$. Однако достаточно использовать диапазон лишь до корня из x. При таком способе подсчета делителей числа необходимо исключить двойной учет делителей, если число x является полным квадратом.

2. При переборе чисел из диапазона происходит повторный подсчет сумм делителей чисел и во внешнем, и во внутреннем цикле. Поэтому, так как в условии задачи указано, что предпочтение должно отдаваться скорости счета, можно использовать дополнительный объем памяти для хранения предварительно подсчитанных сумм делителей чисел всего диапазона.

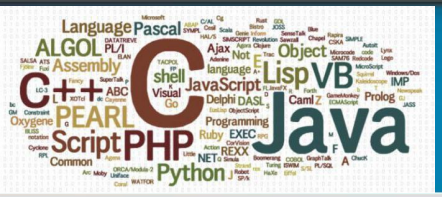


Примеры задач на поиск эффективных решений

Эффективное решение

```
program N1_optimal;
{$APPTYPE CONSOLE}
uses SysUtils, DateUtils;
var i,j,n,s1,s2,s3,ch: longint;
      a: array[1..1000000] of longint;
      t1,t2: TDateTime;
      f: text;

function SD(x: longint): longint;
var i,j,s: longint;  f: boolean;
begin
  j:=trunc(sqrt(x));
  if j*j=x then f:=true else f:=false;
  s:=1;
  for i:=2 to j do
    if x mod i = 0 then
          begin s:=s+i; s:=s+trunc(x/i); end;
  if f=true then s:=s-j;
  SD:=s;
end;

begin
  t1:=Now;
  assign(f,'input.txt');
  reset(f); read(f,n); close(f);
  assign(f,'output.txt'); rewrite(f);
```

```
for i:=1 to n do a[i]:=SD(i);
  for i:=1 to n-1 do
    begin
      s1:=a[i];
      j:=i+1;
      while j<=n do
        begin
          s2:=a[j];
          ch:=s1+s2;
          if (ch>i) and (ch>j) and (ch<=n) then
            begin
              s3:=a[ch];
              if (s3+s2=i) and (s3+s1=j) and (i<>ch) and (j<>ch) then
                 begin writeln(f,i,'  ',j,'  ',ch); j:=n; end;
            end;
          j:=j+1;
        end;
    end;
  close(f);
  t2:=Now;
  writeln('Time ',MilliSecondsBetween(t2,t1)/1000:5:3,' s.');
  write('Program is complete');
  readln;
end.
```



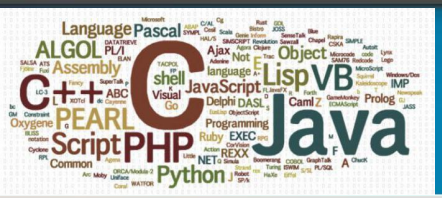
Примеры задач на поиск эффективных решений

Задача 2. Дан одномерный массив, содержащий n целых положительных элементов, причем значение каждого из элементов этого массива не менее 1 и не более n. Найти элементы массива, которые встречаются в нем ровно по одному разу.

Входные данные: В первой строке входного файла INPUT.TXT записано число n – количество элементов в массиве $1 \le n \le 10^6$. В остальных строках файла через пробел идет перечисление элементов.

Выходные данные: В выходной файл OUTPUT.TXT нужно вывести элементы, встречающиеся в массиве ровно по одному разу (последовательность вывода элементов не имеет значения).

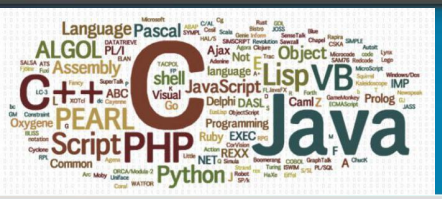
Дополнительные требования: Прежде, чем решать задачу, данные должны быть обязательно считаны из файла в массив.



Примеры задач на поиск эффективных решений

Описание способа решения

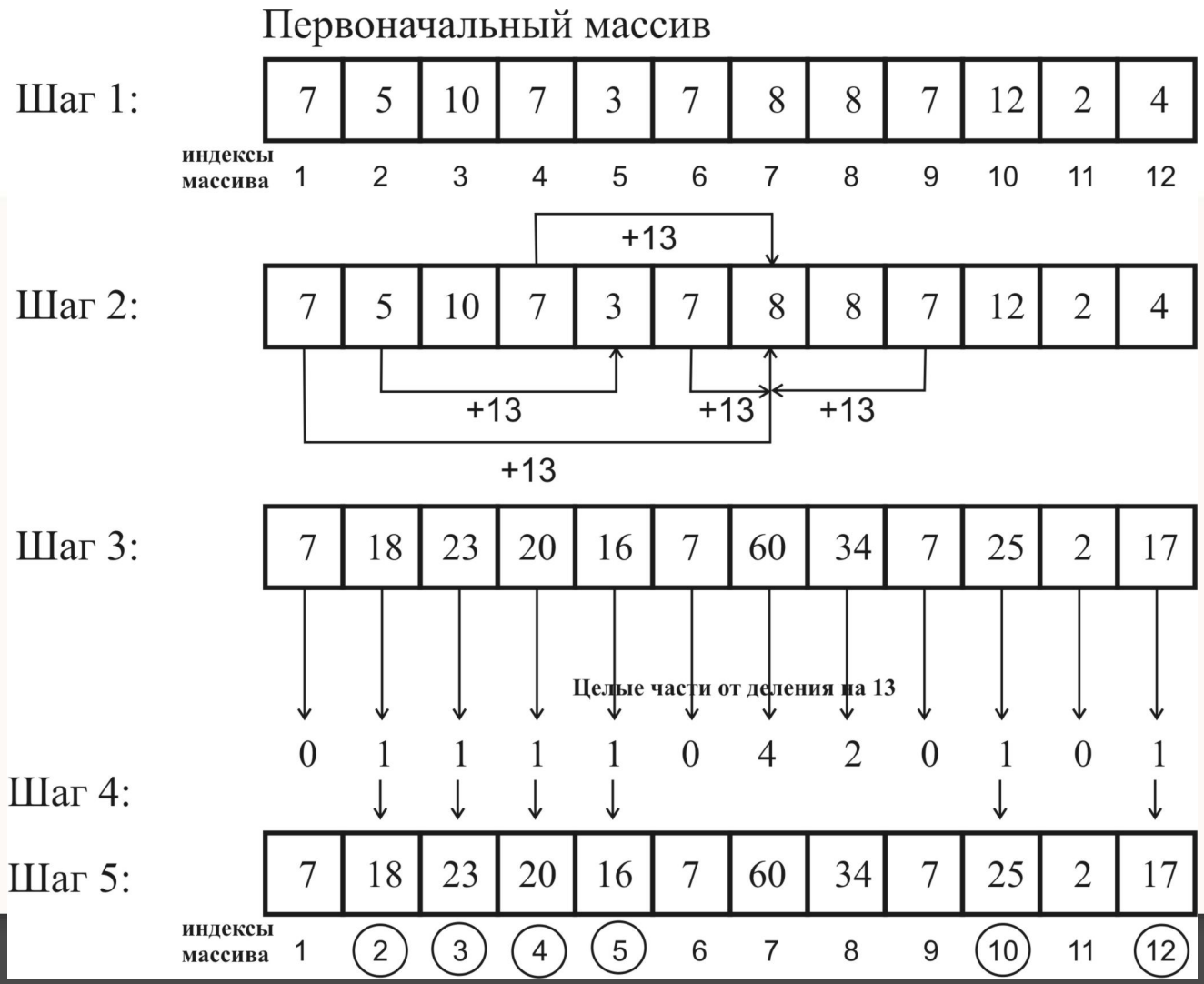
Значения элементов массива не превышают n. Поэтому задача может быть решена в два прохода по массиву. Во время первого прохода к элементу, номер которого соответствует значению элемента первоначального массива, добавляем значение, равное $n+1$. Во время второго прохода остается проверить полученный массив и выбрать номера элементов, значения которых изменились только на $n+1$. Такие номера и будут значениями элементов первоначального массива, которые встречались только по одному разу (если какие-то элементы не изменялись вообще, то это означает, что).
Для того, чтобы во время первого прохода по массиву знать первоначальное значение исходного элемента массива (в результате операций суммирования на предыдущих шагах любой элемент исходного массива мог измениться), необходимо рассматривать не сам этот элемент, а остаток от деления этого элемента на значение, равное $n+1$.

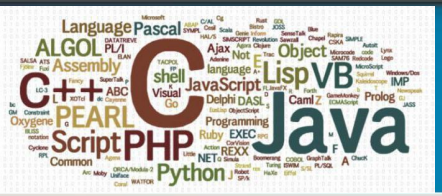


Примеры задач на поиск эффективных решений

Графическая интерпретация при n =12

На рисунке обведены номера индексов, показывающих значения неповторяющихся элементов в массиве: во введенном массиве по одному разу встречаются элементы 2, 3, 4, 5, 10 и 12.

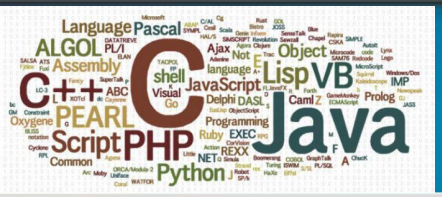




Примеры задач на поиск эффективных решений

Эффективное решение

```
#include <fstream>
using namespace std;
int main()
{ ifstream F;
  long i=0, n;
  long *a= new long [n];
  F.open("INPUT.txt",ios::in);
  if (F) { F>>n;
           while (!F.eof()) F>>a[i++]; } else cout<<"Файл не найден!"<<endl;
  F.close();
  for(i=0; i<n; i++) a[a[i]%(n+1)-1]+=(n+1);  // собственно, само решение
  ofstream f;
  f.open("OUTPUT.txt");
  for(i=0; i<n; i++)  if ((int)a[i]/(n+1) == 1) f<<(i+1)<<" ";
  f.close();
  delete[]a; return 0; }
```



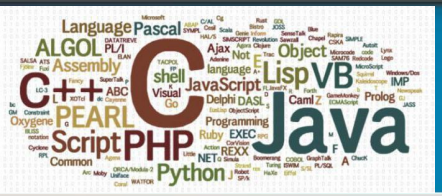
Примеры задач на поиск эффективных решений

Задача 3. Написать программу, позволяющую находить в 24-битовом изображении такие цвета, которых в нем больше всего. Если этому условию отвечают несколько цветов (несколько цветов точек встречаются одинаковое количество раз), то необходимо определить все такие цвета.

Входные данные: В первой и второй строках входного файла INPUT.TXT записаны числа N и M – ширина и высота изображения в точках (пикселях). В следующих строках файла перечисляются через пробел тройки чисел – оттенки цветов точек (так как всего точек в изображении $N \times M$, то таких троек также $N \times M$). Тройки отделяются друг от друга также пробелом. При этом $2 \le N,M \le 10^4$.

Выходные данные: В первую строку выходного файла OUTPUT.TXT вывести единственное число – максимальное количество точек одного цвета в изображении. В следующих строках файла вывести цвета точек (тройки оттенков), которых в изображении больше всего – каждую тройку оттенков – с новой строки.

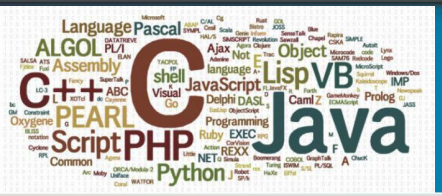
Дополнительные требования: время счета – не более 3 минут.



Примеры задач на поиск эффективных решений

Примеры содержимого входного и выходного файлов

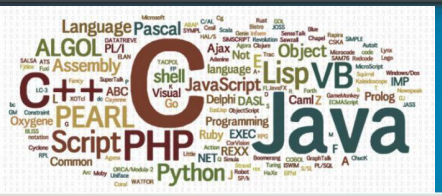
№	INPUT.TXT	OUTPUT.TXT
1	3 3 167 237 252 16 94 229 167 237 252 167 237 252 56 47 9 167 237 252 4 90 100 42 37 230 96 73 73	4 167 237 252
2	4 5 **201 99 30** 66 68 140 173 166 74 246 39 244 173 166 74 53 122 128 173 166 74 187 117 101 156 125 133 191 249 174 151 58 148 168 22 215 221 148 51 92 91 42 230 48 179 **201 99 30** 124 23 56 115 103 249 **201 99 30** 62 63 115	3 173 166 74 201 99 30
3	10 10 3 1 3 3 1 2 2 0 2 3 3 0 2 0 1 1 3 2 1 0 1 3 2 0 3 3 3 1 1 2 0 0 2 3 1 2 1 1 2 0 3 2 2 2 0 1 3 1 1 0 2 3 3 0 0 2 0 3 3 2 3 0 1 3 3 2 3 2 3 3 2 2 0 0 3 1 3 1 1 1 3 2 3 1 0 3 0 2 0 2 0 3 1 3 1 1 3 1 1 1 1 0 1 1 3 1 1 2 2 2 0 3 1 2 0 3 2 2 1 2 3 0 1 1 3 1 3 2 3 0 0 0 0 1 3 0 0 3 2 3 3 2 1 0 1 3 1 0 2 1 2 2 1 0 0 3 0 3 2 3 0 0 3 0 2 0 0 3 2 2 2 2 1 2 2 1 0 1 3 0 3 0 2 3 3 0 1 2 0 0 0 2 1 1 2 3 1 2 2 3 3 0 2 2 2 3 3 0 1 0 0 1 2 2 3 0 0 0 3 3 1 1 1 0 2 3 1 1 2 0 1 1 1 1 1 1 0 1 3 1 1 2 0 1 0 1 0 2 1 0 1 1 3 2 3 0 2 0 3 3 1 3 0 2 0 0 2 0 2 0 1 2 2 0 1 3 0 2 0 1 3 1 3 2 3 0 2 3 3 2 1 3 0 3 3 1 2 0 2 3	6 1 3 1



Примеры задач на поиск эффективных решений

Описание способа решения

Каждый оттенок цвета точки лежит в диапазоне от 0 до 255. Поэтому для решения задачи необходимо использовать трехмерный массив размерностью 256×256×256 элементов типа *longint* и при считывании значений трех очередных оттенков (цвета текущей точки) просто увеличивать значение соответствующего элемента массива на единицу. После просмотра всех оттенков необходимо найти в трехмерном массиве максимальный элемент (или элементы, если их несколько) и вывести соответствующие оттенки.

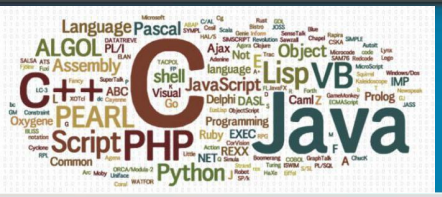


Примеры задач на поиск эффективных решений

Эффективное решение

```
program N3;
{$APPTYPE CONSOLE}
uses SysUtils,DateUtils;
var i,j,l,nx,ny,nz,R,G,B: byte;
      a: array[0..255,0..255,0..255] of longint;
      n1,n2: integer;
      k,m,max: longint;
      t1,t2: TDateTime;
      f: text;
begin
  t1:=Now;
  for i:=0 to 255 do       // обнуляем массив a
     for j:=0 to 255 do
        for l:=0 to 255 do a[i,j,l]:=0;
  assign(f,'input.txt'); reset(f);
  read(f,n1); read(f,n2);
  m:=n1*n2;  k:=1;
// считываем значения "цветов" точек и
сразу подсчитываем их количество
  while k<=m do
    begin
      read(f,R); read(f,G); read(f,B);
      a[R,G,B]:=a[R,G,B]+1;    k:=k+1;
    end;  close(f);
```

```
max:=0;  nx:=0; ny:=0; nz:=0;
{ определяем "цвет" точек, которых в "изображении" больше
всего - для этого ищем максимальное значение в массиве a }
 for i:=0 to 255 do
    for j:=0 to 255 do
       for l:=0 to 255 do
        if a[i,j,l]>max then          // нашли новый максимум
          begin
            max:=a[i,j,l];
            nx:=i; ny:=j; nz:=l;     // запоминаем "цвет" точки
          end;
  assign(f,'output.txt');
  rewrite(f);
  writeln(f,max); // выводим max количество точек одного цвета
{ ищем точки, цвет которых больше всего встречается, т.к.
может быть несколько цветов, встречающихся max раз }
 for i:=0 to 255 do
    for j:=0 to 255 do
       for l:=0 to 255 do  if a[i,j,l]=max then writeln(f,i,' ',j,' ',l);
  close(f);
  t2:=Now;
  writeln('Time ',MilliSecondsBetween(t2,t1)/1000:5:3,' s.');
  write('Program is complete');
  readln; end.
```



Примеры задач на поиск эффективных решений

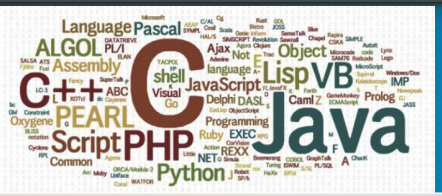
Задача 4. В матрице A размером $N \times M$, не содержащей повторяющихся элементов, необходимо найти элементы, наиболее близкие друг к другу по значению. Например, для матрицы

$$\begin{pmatrix} 1 & 23 & 17 & 3 \\ 15 & 16 & 12 & 6 \\ 56 & 5 & 4 & 43 \end{pmatrix}$$

наиболее близкими друг к другу по значению являются пары элементов 3 и 4, 4 и 5, 5 и 6, 16 и 17 (т.к. элементы во всех этих парах отличаются по значению друг от друга на 1).

Входные данные: В первых двух строках входного файла INPUT.TXT записаны два числа (по одному в каждой строке): N и M, причем $2 \le N \le 10000$, $2 \le M \le 10000$. В остальных строках файла через пробел идет перечисление элементов матрицы A (каждая строка матрицы – с новой строки в файле). Для каждого элемента a_{ij} матрицы A выполнено условие: $1 \le a_{ij} \le 10^9$.

Выходные данные: В строки выходного файла OUTPUT.TXT вывести пары чисел, удовлетворяющие условию задачи. Числа в паре разделены двумя пробелами, каждая пара – с новой строки. При выводе необходимо исключить повторы.

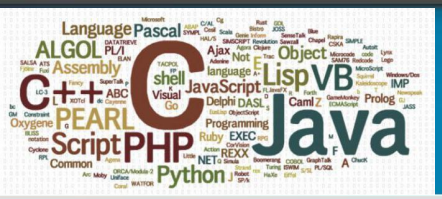


Примеры задач на поиск эффективных решений

Возможное неэффективное решение

Наиболее привлекательным выглядит применение алгоритма быстрой сортировки, однако в зависимости от того, как будет реализована вся задача в целом, использование такого метода приведет к затратам памяти от 400 Мб до 1 Гб для матриц размерностью $10^4 \times 10^4$ элементов. При этом в зависимости от параметров вычислительной системы, на которой будет запущена данная программа, мы получим очень сильно различающийся результат по времени вычислений.

Так, например, на компьютере с процессором AMD Sempron с тактовой частотой 1.6 ГГц и объемом ОЗУ 384 Мб для матрицы максимальной размерности время счета составит около 7 минут. Компьютер с процессором Core i3 с тактовой частотой 3.4 ГГц и объемом ОЗУ 4 Гб справится с этой же задачей примерно за 35 секунд. Таким образом, в данном случае более мощный компьютер решит задачу в 12 раз быстрее.



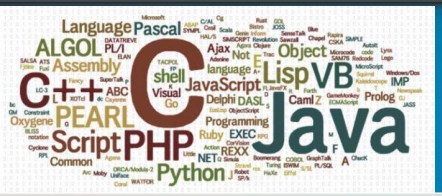
Примеры задач на поиск эффективных решений

Описание эффективного способа решения

1. Для хранения чисел исходной матрицы будем использовать не двумерный, а одномерный массив.
2. Откажемся от использования сортировки элементов.

В одномерный массив будем записывать значения, равные 0 или 1 – если, например, в матрице присутствует элемент 1276, то в соответствующий элемент одномерного массива (с индексом 1276) поместим единицу, в противном случае – ноль. Т.к. значения элементов матрицы не превосходят 10^9, то потребуется одномерный массив на 10^9 элементов. Значения элементов массива – единицы и нули, поэтому можно объявить его как тип *byte*, и на каждый элемент массива потребуется один байт памяти (всего тогда потребуется около 954 Мб памяти).

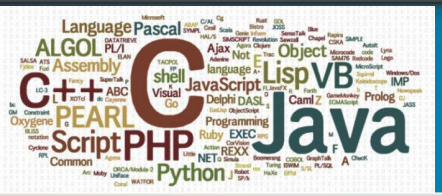
На самом деле для размещения информации о 10^9 элементов можно использовать в 8 раз меньше памяти, т.е. всего 120 Мб. Для этого представим последовательности из 8 нулей и единиц в виде символов – тогда для хранения элементов исходной матрицы будет достаточно массива типа *byte* на 125 млн. элементов. Таким образом, потребуется осуществлять перевод из двоичной системы в десятичную, и обратно. С целью ускорения таких операций перевода потребуется еще два дополнительных небольших массива (в коде ниже они обозначены через *bin* и *db*).



Примеры задач на поиск эффективных решений

Последовательность реализации

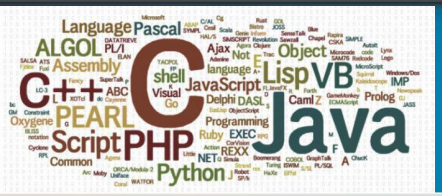
1. Подготовка (заполнение) массивов для ускорения двоичного представления чисел (массивы *bin* и *db*).
2. Заполнение основного массива *a* значениями элементов исходной матрицы на основе описанного выше правила.
3. Определение минимальной разницы между элементами основного массива *a* (по сути, находим разницу между соседними элементами).
4. Поиск пар элементов, разница между которыми равна минимальной.



Примеры задач на поиск эффективных решений

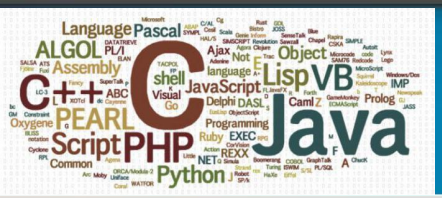
Эффективное решение

```
program N4;
{$APPTYPE CONSOLE}
uses SysUtils,DateUtils,Math;
var i,j,n,k,min,pos,elem,a1,a2: longint;
     s: string[8];  f: text;
    a: array[1..125000000] of byte;    // числа, состоящие из позиций элементов
    bin: array[0..255] of string[8];     // двоичное представление чисел 0 – 255
    db: array[1..8] of byte;  { значения чисел для различных разрядов единиц, на которые будет
                                изменяться элемент массива a, например, если 1 добавляем в
                                позицию pos=2,  то это означает, что нужно добавить 64}
   t1,t2: TDateTime;

function ByteToBinary(A: Byte): string;  // перевод числа в двоичный формат
var  vrem: string;  B: Byte;
begin
  vrem:='';
  while A>1 do begin    B:=A mod 2;  A:=A div 2;
                        if B=1 then Vrem:='1'+Vrem else Vrem:='0'+Vrem;
                end;
  if A=1 then Vrem:='1'+Vrem;
  while Length(Vrem)<8 do Vrem:='0'+Vrem;    // Дополняем строку Vrem до 8 символов (1 байт)
  ByteToBinary:=Vrem;
end;
```



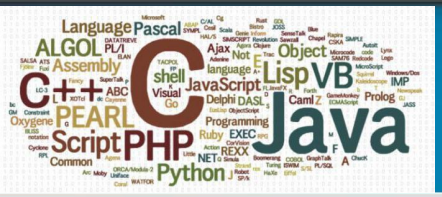
Примеры задач на поиск эффективных решений

```
function BinaryToByte(s: string):byte; // перевод числа в десятичный формат
 var i,j,k: byte;
begin
  i:=1; k:=0;
  For j:=7 DownTo 0 do
    begin
      k:=k+StrToInt(s[i]) shl j;   i:=i+1;
    end;
  BinaryToByte:=k;
end;

begin  // Основная программа
 t1:=Now;
 //готовим двоичное представление чисел
 for i:=0 to 255 do bin[i]:=ByteToBinary(i);
for i:=7 downto 0 do db[8-i]:=trunc(IntPower(2,i));
 assign(f,'input.txt'); reset(f);
  n:=0;
  read(f,i); read(f,j);                    // пропускаем размерность массива
  for k:=1 to 125000000 do a[k]:=0; // начальные значения элементов
```



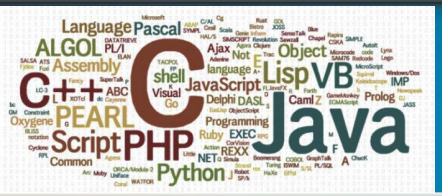
Примеры задач на поиск эффективных решений

```
while not eof(f) do       // формируем массив a
  BeGin
    read(f,i);
    if i>n then n:=i;    //запоминаем максимальный элемент входного массива
    elem:= i div 8;      // номер элемента массива a, который нужно изменять
    if i>elem*8 then   // в этом случае нужен следующий элемент
      begin
        inc(elem);
        pos:= i mod 8;  // позиция цифры находится как остаток
      end else pos:=8; // если кратно 8, то позиция изменяемой цифры - 8-я.

{ Пример. Пусть i = 34 (нужно записать единицу в позицию 34). Тогда elem:= i div 2 даст 4. Но
34 > 4*8, поэтому берем elem = 5. В этом элементе единицу надо поставить в позицию i mod 8,
т.е. в позицию 2. Т.о., если пятый элемент был 00000000, то он станет 0100000 }

    a[elem]:=a[elem]+db[pos];
  EnD;
 close(f);
 k:=n div 8;  // вычисляем, сколько элементов в массиве a использовано
 if n>k*8 then inc(k);
{  ищем 1-й элемент, отличный от нуля – в нем информация о 1-м элементе входного массива }
 n:=1;
 while a[n]=0 do inc(n);
```



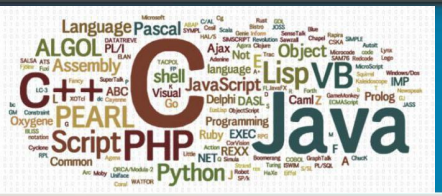
Примеры задач на поиск эффективных решений

```
{ ищем в его двоичной записи первую единицу - она определяет значение первого элемента/
входного массива }
 s:=bin[a[n]];
 i:=1;
 while s[i]<>'1' do inc(i);
 a1:=(n-1)*8+i; // первый элемент входного массива
 min:=2000000000;
 for i:=n to k do     // ищем минимальную разность
   if a[i]<>0 then  // рассматриваем только ненулевые элементы
   begin
     s:=bin[a[i]];
    for j:=1 to 8 do
      if s[j]='1' then
        begin
          a2:=(i-1)*8+j;
          if (a2>a1) and (a2-a1<min) then min:=a2-a1;
          a1:=a2;
        end;
   end;
```



Примеры задач на поиск эффективных решений

```
// ищем ближайшие элементы
s:=bin[a[n]];
i:=1;
while s[i]<>'1' do inc(i);
a1:=(n-1)*8+i;        // первый элемент входного массива
assign(f,'output.txt');  rewrite(f);
for i:=n to k do        // ищем элементы с минимальной  разностью
  if a[i]<>0 then      // рассматриваем только ненулевые элементы
  begin
     s:=bin[a[i]];
     for j:=1 to 8 do
        if s[j]='1' then  begin
                             a2:=(i-1)*8+j;
                             if (a2-a1=min) then writeln(f,a1,'  ',a2);
                             a1:=a2;
                          end;
  end;
close(f);
t2:=Now;
writeln('Time ',MilliSecondsBetween(t2,t1)/1000:5:3,' s.');
write('Program is complete');
readln; end.
```



Примеры задач на поиск эффективных решений

Результативность

Единственная трудность при программной реализации описанного способа – это получение реальных значений элементов, хранящихся в массиве *a*.

Тем не менее, способ исключает необходимость применения сортировки, что делает его более быстрым (а значит, и эффективным) в условиях ограниченных ресурсов.

Так, на компьютере с процессором AMD Sempron с тактовой частотой 1.6 ГГц и объемом ОЗУ 384 Мб для матрицы максимальной размерности время счета составит всего около 58 секунд, по сравнению с 7-ю минутами при использовании алгоритма быстрой сортировки. Примечательно, что для компьютера с процессором Core i3 с тактовой частотой 3.4 ГГц и объемом ОЗУ 4 Гб прироста практически не будет – время счета составит около 32 секунд (при этом будет выигрыш по объему используемой памяти).