Критерии оценивания и решения олимпиады «Высшая проба» по информатике, 2014/2015 учебный год

10-11 класс

Критерии оценивания

Задача 1: за каждое верно указанное число начисляется 1 балла
Задача 2: за каждое верно указанное число начисляется 2 балла
Задача 3: каждый тест оценивается в 3 балла
Задача 4: баллы по тестам: 5, 4, 4, 4, 3
Задача 5: каждый тест оценивается в 4 балла
Задача 6: каждый тест оценивается в 5 баллов

Все тесты приведены в отдельном архиве.

Задача 1

112 177 240 74 245 32778 49156 36608 57349 40654

Вариант 1
Посчитать вручную

Вход
Число Объем Биты

В двоичном представлении необходимо выделить первые биты
Начало=Число / (2^(8*Объем-Биты))
Затем последние биты
Конец=Число Mod (2^(Биты))
Выделяем неизменяемую часть
Середина= (Число Mod (2^(8*Объем-Биты))) / (2^(Биты))
Затем собираем новое число
(Выход)
Конец*(2^(8*Объем-Биты))+Середина*(2^(Биты))+Начало

Вариант 2
Те же расчеты, что и в варианте 1, провести в электронной таблице
Электронная таблица (Excel) в режиме формул

A	B	C	D	E	F	G	H	I
Число	объем (байт)	биты	необходимая степень двойки для выделения начала	необходимая степень двойки для выделения конца	начальные биты	середина	конечные биты	результат
7	1	4	=2^(8*B2-C2)	=2^C2	=ОТБР(A2/D2)	=ОТБР(ОСТАТ(A2;D2)/E2)	=ОСТАТ(A2;E2)	=G2*E2+F2+H2*D2
27	1	4	=2^(8*B3-C3)	=2^C3	=ОТБР(A3/D3)	=ОТБР(ОСТАТ(A3;D3)/E3)	=ОСТАТ(A3;E3)	=G3*E3+F3+H3*D3
51	1	2	=2^(8*B4-C4)	=2^C4	=ОТБР(A4/D4)	=ОТБР(ОСТАТ(A4;D4)/E4)	=ОСТАТ(A4;E4)	=G4*E4+F4+H4*D4
137	1	2	=2^(8*B5-C5)	=2^C5	=ОТБР(A5/D5)	=ОТБР(ОСТАТ(A5;D5)/E5)	=ОСТАТ(A5;E5)	=G5*E5+F5+H5*D5
183	1	3	=2^(8*B6-C6)	=2^C6	=ОТБР(A6/D6)	=ОТБР(ОСТАТ(A6;D6)/E6)	=ОСТАТ(A6;E6)	=G6*E6+F6+H6*D6
11	2	1	=2^(8*B7-C7)	=2^C7	=ОТБР(A7/D7)	=ОТБР(ОСТАТ(A7;D7)/E7)	=ОСТАТ(A7;E7)	=G7*E7+F7+H7*D7
7	2	2	=2^(8*B8-C8)	=2^C8	=ОТБР(A8/D8)	=ОТБР(ОСТАТ(A8;D8)/E8)	=ОСТАТ(A8;E8)	=G8*E8+F8+H8*D8

143	2	8	=2^(8*B9-C9)	=2^C9	=ОТБР(A9/D9)	=ОТБР(ОСТАТ(A9;D9)/E9)	=ОСТАТ(A9;E9)	=G9*E9+F9+H9*D9
40967	2	3	=2^(8*B10-C10)	=2^C10	=ОТБР(A10/D10)	=ОТБР(ОСТАТ(A10;D10)/E10)	=ОСТАТ(A10;E10)	=G10*E10+F10+H10*D10
61129	2	4	=2^(8*B11-C11)	=2^C11	=ОТБР(A11/D11)	=ОТБР(ОСТАТ(A11;D11)/E11)	=ОСТАТ(A11;E11)	=G11*E11+F11+H11*D11

Число	объем (байт)	биты	необходимая степень двойки для выделения начала	необходимая степень двойки для выделения конца	начальные биты	середина	конечные биты	результат
7	1	4	16	16	0	0	7	112
27	1	4	16	16	1	0	11	177
51	1	2	64	4	0	12	3	240
137	1	2	64	4	2	2	1	74
183	1	3	32	8	5	2	7	245
11	2	1	32768	2	0	5	1	32778
7	2	2	16384	4	0	1	3	49156
143	2	8	256	256	0	0	143	36608
40967	2	3	8192	8	5	0	7	57349
61129	2	4	4096	16	14	236	9	40654

Вариант 3

Написать программу (Паскаль)

```
program p1;
var x,y,a,b,c:longint;
   v,k,i:byte;

{функция вычисления степени }
function pow(x,n:longint):longint;
var y:longint;i:integer;
begin
 y:=1;
 for i:=1 to n do
  y:=y*x;
 pow:=y;
end;

begin

for i := 1 to  10  do
begin
 readln(x,v,k);
 a := x div pow(2,8*v-k);   {начало числа}
 b := x Mod pow(2,k);       {конец числа }
 c := (x Mod pow(2,8*v-k)) div pow(2,k); {середина }
 y := b* pow(2,8*v-k) + c*pow(2,k)+a; {новое число }
 writeln(y)
end;
end.
```

Задача 2

0 7969 180 56045 162

Задача 3

Для получения ответа надо посчитать количество 1 - это общая площадь кораблей.
Чуть сложнее посчитать количество самих кораблей. Для этого нужно посчитать левые верхние углы, т.е. такие клеточки с 1, что выше 0 или граница матрицы, а также 0 левее или граница матрицы.

```
program p3;
var x:array[1..100,1..100]of integer;
i,j,n,k,s :integer;  left,top:boolean;

begin
{ввод поля}
 readln(n);
 for i:=1 to n do
  begin
   for j:=1 to n do
    read(x[i,j]);
  end;

 s:=0; {количество 1 – площадь кораблей}
 k:=0;  {количество кораблей}
 for i:=1 to n do
   for j:=1 to n do
    if x[i,j]=1 then {если в клеточке часть корабля}
     begin
      s:=s+1;   {увеличиваем площадь}

      if (i=1) then  {сверху – граница матрицы}
       left:=true
      else
       if x[i-1,j]=0 then {сверху нет коробля}
         left :=true
       else
         left:=false;
      if (j=1) then  {слева– граница матрицы}
        top:=true
      else
       if x[i,j-1]=0 then {слева нет коробля}
         top:=true
       else
         top:=false;
      if  left and top then {слева и сверху нет короблей – это верхний левый угол корабля}
        k:=k+1;
     end;
  if k<>0 then s:=s div k;
```

```
  writeln(s); {подсчет среднего}

end.
```

Задача 4

Пример программы (Паскаль)

```
program p4;
var x,y,z:array[1..103] of byte;
n,k,s,r,i,count:integer;

{функция проверки бьет ли король данную координату.
Условие того, что король бьет поле:
Все координаты поля отличаются от соответствующих координат короля не более, чем на 1.
|X-Xk|<=1 and |Y-Yk|<=1 and |Z-Zk|<=1
Координаты короля хранятся в первых элементах массивов координат.
}

function ko (a,b,c:byte):boolean;
begin
 ko:= (abs(x[1]-a)<=1)and(abs(y[1]-b)<=1)and(abs(z[1]-c)<=1)
end;

{функция проверки бьет ли ладья данную координату
Условие того, что ладья бьет поле – совпадение двух из трех координат
X=Xl and Y=Yl  or X=Xl and Z=Zl   or Y=Yl and Z=Zl
+
Пербрать все фигуры и проверить – не стоят ли они между ладьей и данной фигурой
}

function l (j,a,b,c:byte):boolean;
var i,s:integer;
begin
 s:=101;
 if (x[j]=a)and (y[j]=b) then
  begin
  s:=0;
  for i:= 1 to k+3 do
    if (x[i]=a)and (y[i]=b) and
       ((z[j]<z[i])and(z[i]<c)or(z[j]>z[i])and(z[i]>c)) then
      s:=s+1;
  end;
 if (z[j]=c)and (y[j]=b) then
  begin
  s:=0;
  for i:= 1 to k+3 do
    if (y[i]=b)and (z[i]=c) and
       ((x[j]<x[i])and(x[i]<a)or(x[j]>x[i])and(x[i]>a)) then
      s:=s+1;
```

```
  end;
 if (z[j]=c)and (x[j]=a) then
  begin
  s:=0;
  for i:= 1 to k+3 do
      if (x[i]=a)and (z[i]=c) and
        ((y[j]<y[i])and(y[i]<b)or(y[j]>y[i])and(y[i]>b)) then
       s:=s+1
  end;
 l:= s=0
end;

{сама программа}
begin
{вводится размерность игрового куба и координаты белых фигур}
readln(n);
for i:=1 to 3 do
 readln(x[i],y[i],z[i]);
{вводится количество и координаты черных фигур}
readln(k);
for i:=4 to k+3 do
 readln(x[i],y[i],z[i]);
{просматриваем каждую черную фигуру, в s подсчитываем сколько белых фигур бьют ее}
for i:=4 to k+3 do
 begin
 s:=0;
 if ko(x[i],y[i],z[i])then
  begin
     s:=s+1;
  end;
 if l(2,x[i],y[i],z[i])then
  begin

   s:=s+1;
  end;
 if l(3,x[i],y[i],z[i])then
  begin
   s:=s+1;
  end;
{в count подсчитываем количество фигур под ударом}
   if s>0 then count:=count+1;
 end;
   writeln( k-count);
end.
```

Задача 5

Идея решения заключается в следующем: необходимо построить выпуклую фигуру наибольшей площади «натянутую» на заданные точки.

Для этого сначала выбираем самую левую точку, если с такой координатой несколько точек, то самую нижнюю. Переставим ее на 1-ую позицию в массиве. Далее она будет началом всех лучей.

Все остальные точки в массиве сортируются по возрастанию углового коэффициента (тангенса угла наклона) прямых, проведенных из этих точек в первую точку массива.

Далее обходим массив, исключая лишние точки.

Новую фигуру разбиваем на треугольники, площадь выпуклой фигуры – сумма площадей этих треугольников. Одна из вершин каждого треугольника - первая точка в массиве две другие – соседние вершины многоугольника.

Пример программы (Паскаль)

```
program p;

const n=15;
type data=array[1..2] of real; {тип точка – две координаты}
    arint=array[1..n] of data;  {тип массива координат точек}
var x,new: arint; {исходный массив координат точек и массив результирующей фигуры}
   s:data;
   i,j,m: integer;
   Sf: real;    {площадь}

function r(a,b:data):real;
{функция вычисления расстояния между двумя точкам}
begin
r:= sqrt(sqr(a[1]-b[1])+sqr(a[2]-b[2]));
end;

function square(x,y,z:data):real;
{функция вычисления площади треугольника с вершинами в трех данных точках}
var p,a,b,c:real;
begin
a:=r(x,y);
b:=r(x,z);
c:=r(y,z);
p:=(a+b+c)/2;
square:= sqrt(p*(p-a)*(p-b)*(p-c));
end;

function f(a,b,c:data):real;
{функция вычисляет положение точки C относительно прямой, проходящей через A и B}
begin
    if (b[1]-a[1])=0 then
```

```
      f:=(c[1]-a[1])
    else
      f:=(((b[2]-a[2])*(c[1]-a[1]))/(b[1]-a[1]))-(c[2]-a[2])
end;

procedure read_array(var ar: arint;m:integer);
{процедура ввода точек в массив координат}
var i: integer;
begin
   for i:=1 to m do
   begin
       readln(ar[i][1],ar[i][2]);
   end
end;


function more(a,b:data):boolean;
{Через первую точку массива проводят прямые через заданную точку A и через точку B,
далее точки сравниваются по коэффициенту угла наклона этих прямых}
var dxi,dxj,dyi,dyj,d:real;
begin
    dxi:=a[1]-x[1][1];
    dxj:=b[1]-x[1][1];
    dyi:=a[2]-x[1][2];
    dyj:=b[2]-x[1][2];
    if dxj=0 then
      if dxi=0 then more:=dyj-dyi<=0
      else more:=true
    else if dxi<>0 then begin
        d:=((dyi/dxi)-(dyj/dxj));
        if d=0 then more:=dxj<dxi
        else more:=d<0
    end else more:=false
end;

function less(a,b:data):boolean;
{функция сравнения двух точек, меньшей считается та, которая левее, а при равных
координатах по оси Ox та, которая ниже}
begin
 if a[1]<b[1] then less:=true
 else if (a[1]=b[1])and(a[2]<b[2]) then less:=true
 else less:=false
end;

function min(x:arint;m:integer):integer;
{поиск минимальной (в смысле описанной ранее функции сравнения less) точки в массиве}
var s,i:integer;
begin
    s:=1;
    for i:=2 to m do
```

```
        if less(x[i],x[s]) then  s:=i;
    min:=s
end;

procedure sort(var ar: arint; m:integer);
{процедура сортирует методом пузырька точки в массиве по углу наклона прямых,
исходящих из первой точки массива в остальные его точки}
var k,i:integer; t:data;
begin
 for k:=m downto 2 do
  for i:= 2 to (k-1) do
   if more(ar[i],ar[i+1]) then
    begin
     t:=ar[i];
     ar[i]:=ar[i+1];
     ar[i+1]:=t;
    end;
end;


begin
   {вводим исходные данные}
    readln(m);
    read_array(x,m);

    {находим самую левую нижнюю точку и переставляем ее в начало массива}
    i:=min(x,m);
    s:=x[i];
    x[i]:=x[1];
    x[1]:=s;

    {сортируем точки по углу наклона прямых, проведенных из этих точек в первую}
    sort(x,m);

    {строим выпуклую фигуру new}
    i:=0;
    j:=0;
   {переписываем в массив точки с той же координатой по Оу, что и у первой точки}
    repeat
        i:=i+1;
        j:=j+1;
        new[i]:=x[j]
    until x[j][1]<>x[1][1];
    {если в массиве меньше трех точек добавим еще точку}
    if i<3 then begin
        i:=i+1;
        j:=j+1;
        new[i]:=x[j]
    end;
```

```
  {обходим точки, если очередная добавляемая  образует прямую с последней добавленной, которая разбивает фигуру таким образом, что первая и предпоследняя добавленная в фигуру лежат в разных полуплоскостях, тогда исключам последнюю добавленную. Повторяем, исключая лишние вершины. Затем переносим рассмотренную вершину из исходного массива в новый}
    while j<m do begin
        while f(x[j+1],new[i],new[i-1])*f(x[j+1],new[i],new[1])<=0 do
            i:=i-1;
        i:=i+1;
        j:=j+1;
        new[i]:=x[j];
    end;

    {разбиваем новую фигуру на треугольники, одна вершина – это первая в массива, остальные – попарно соседние вершины многоугольника. Набор таких треугольников полностью покрывает всю выпуклую фигуру}
    sf:=0;
    for j:=3 to i do
     sf:=sf+square(new[1],new[j-1],new[j]);

    writeln(sf);

end.
```

Задача 6

Пример программы (C++)

```
#include <iostream>
#include <vector>
#include <set>
#include <utility>

using namespace std;

enum {MAXN = 100005};

int i, j, a, b, c, n, m;
vector <pair <int, int> > g[MAXN];
int ans[MAXN][2];
pair <int, int> from[MAXN][2];
set <pair <int, pair <int, int> > > q;

int main() {
        cin >> n >> m;
        for (i = 0; i < m; i++) {
                cin >> a >> b >> c;
                a--; b--;
                g[a].push_back({ b, c });
```

```
		g[b].push_back({ a, c });
	}
	for (i = 0; i < n; i++) {
		ans[i][0] = INT_MAX;
		ans[i][1] = INT_MAX;
	}
	from[0][0] = { -1, -1 };
	ans[0][0] = 0;
	ans[0][1] = 0;
	q.insert({ 0, { 0, 0 } });
	while (!q.empty()) {
		auto now = *q.begin();
		q.erase(q.begin());
		while (!q.empty() && ans[now.second.first][now.second.second] < now.first) {
			now = *q.begin();
			q.erase(q.begin());
		}
		int nowv = now.second.first;
		int nowt = now.second.second;
		for (auto neibp : g[nowv]) {
			int neib = neibp.first;
			int dist = neibp.second;
			if (ans[nowv][nowt] + dist < ans[neib][nowt]) {
				ans[neib][nowt] = ans[nowv][nowt] + dist;
				from[neib][nowt] = { nowv, nowt };
				q.insert({ ans[neib][nowt], { neib, nowt } });
			}
			if (nowt == 0 && ans[nowv][nowt] < ans[neib][1]) {
				ans[neib][1] = ans[nowv][nowt];
				from[neib][1] = { nowv, nowt };
				q.insert({ ans[neib][1], { neib, 1 } });
			}
		}
	}
	vector <int> cities;
	cout << ans[n - 1][1] << ' ';
	int nowv = n - 1, nowt = 1;
	int f, s;
	while (nowv != -1) {
		cities.push_back(nowv);
		auto now = from[nowv][nowt];
		if (now.second == 0 && nowt == 1) {
			f = now.second;
			s = nowt;
		}
		nowv = now.first;
		nowt = now.second;
	}
	reverse(cities.begin(), cities.end());
	cout << cities.size() - 1 << '\n';
```

```
	for (auto c : cities)
		cout << c + 1 << ' ';
	cout << '\n';
	cout << f + 1 << ' ' << s + 1;
	return 0;
}
```