Департамент образования администрации города Нижнего Новгорода
Нижегородский государственный университет им. Н. И. Лобачевского
Городской ресурсный центр физико-математического образования

Восьмая нижегородская городская олимпиада школьников по информатике

9 февраля 2012 г.

Нижний Новгород
2012

Результаты, архивы и другие материалы олимпиады
можно найти на сайте `http://olympiads.nnov.ru`

Оригинал-макет подготовлен в системе LaTeX 2ε
с использованием набора шрифтов LH.

Восьмая городская олимпиада по информатике

9 февраля 2012 г. Городской ресурсный центр физико-математического образования в лице учредителей: Департамента образования администрации города Нижнего Новгорода, Нижегородского государственного университета им. Н. И. Лобачевского, Института прикладной физики РАН и МБОУ Лицей № 40 проводит восьмую городскую олимпиаду по информатике среди учащихся 6–11 классов образовательных учреждений города Нижнего Новгорода. Целью проведения олимпиады является поиск талантливой молодёжи, привлечение её в науку, повышения уровня преподавания предметов физико-математического цикла.

Генеральным спонсором городской олимпиады школьников по информатике является компания «Мера НН».

Восьмая городская олимпиада проводится на трех компьютерных площадках:

- Нижегородский государственный университет им. Н. И. Лобачевского (компьютерные классы механико-математического факультета) — 25 мест;
- Институт прикладной физики Российской академии наук (компьютерные классы Научно-образовательного центра) — 23 места;
- Муниципальное бюджетное образовательное учреждение лицей № 40 (компьютерный центр) — 22 места;

Олимпиада проводится в соответствии с Положением о городской олимпиаде по информатике (Приказ № 1453 от 07.11.2008 Департамента образования и СПЗД администрации г. Нижнего Новгорода).

Разрешённые среды программирования — Borland Pascal, Free Pascal, Borland C, Borland C++, GNU C (MinGW), GNU C++ (MinGW), Basic. Ввод и вывод данных в программах осуществляется через файлы.

Регламент проведения олимпиады

9:30—10:00	Регистрация участников (на всех компьютерных площадках одновременно)
10:00—15:00	Решение задач олимпиады
15:00—15:30	Перерыв для сбора всех участников олимпиады в конференц-зале НОЦ ИПФ РАН, переезд в ИПФ РАН, кофе-брейк.
15:30—16:30	Открытое тестирование
16:30—18:00	Приветствие участников олимпиады, выступления учредителей, спонсоров. Подведение итогов олимпиады. Поздравление победителей и призёров.

Состав оргкомитета олимпиады

Швецов В. И., проректор по информатизации ННГУ им. Н. И. Лобачевского, профессор;
Сидоркина С. Л., первый заместитель директора департамента образования администрации города Нижнего Новгорода;
Лелюх В. Д., старший преподаватель ННГУ им. Н. И. Лобачевского;
Цветков М. И., начальник отдела общего среднего образования департамента образования администрации города Нижнего Новгорода;
Бовкун И. Л., главный специалист отдела общего среднего образования департамента образования администрации города Нижнего Новгорода;
Смирнов А. И., директор НОЦ ИПФ РАН, профессор;
Фейгина Т. А., заместитель директора НОЦ ИПФ РАН;
Умнова Н. С., директор муниципального бюджетного образовательного учреждения лицей № 40;
Евстратова Л. П., заместитель директора по учебно-воспитательной работе МБОУ лицей № 40;
Гашпар И. Л., куратор НОЦ ИПФ РАН;
Братчикова Т. А., учитель МБОУ лицей № 40;
Денисов В. В., зав. лабораторией ННГУ им. Н. И. Лобачевского;

Состав предметной комиссии (жюри)

Председатель — Лелюх В. Д., старший преподаватель ННГУ;

Члены комиссии:
Евстратова Л. П., заместитель директора МОУ лицей № 40, ответственный секретарь комиссии;
Бударагин Д. И., студент 6 курса ВШОПФ ННГУ;
Вадимов В. Л., студент 3 курса ВШОПФ ННГУ;
Демидов А. Н., инженер, НПП «ПРИМА»;
Епифанов В. Ю., студент 3 курса мехмата ННГУ;
Калинин П. А., младший научный сотрудник ИПФ РАН;
Круглов А. А., младший научный сотрудник ИПФ РАН;
Матросов М. В., студент 6 курса ВМК ННГУ;
Тимушев Р. И., консультант, Grid Dynamics;
Хаймович И. М., аспирант ИФМ РАН;
Шмелёв А. С., магистрант 2 года ВМК ННГУ.

Ниже приведены примеры текстов программ, написанных на разрешённых языках программирования. Программы считывают данные (два числа в одной строке) из файла с именем `example.in` и выводят в файл с именем `example.out` их сумму:

Pascal

```
var buf,bufo:text;
    a,b:integer;
begin
  assign(buf,'example.in');
  reset(buf);
  read(buf,a,b);
  assign(bufo,'example.out');
  rewrite(bufo);
  writeln(bufo,a+b);
  close(buf);
  close(bufo);
end.
```

C/C++

```
#include <stdio.h>
int main() {
  FILE *buf,*bufo;
  int a,b;
  buf=fopen("example.in","r");
  fscanf(buf,"%d %d",&a,&b);
  fclose(buf);
  bufo=fopen("example.out","w");
  fprintf(bufo,"%d\n",a+b);
  fclose(bufo);
  return 0;
}
```

Basic

```
OPEN "example.in" FOR INPUT AS #1
OPEN "example.out" FOR OUTPUT AS #2
INPUT #1, A, B
PRINT #2, A + B
CLOSE #2, #1
```

VIII Городская олимпиада школьников по информатике 9 февраля 2012 г.

Задача 1. Переливание духов

Входной файл	`flow.in`
Выходной файл	`flow.out`
Ограничение по времени	1 с
Максимальный балл за задачу	100

> ... но нужен ли на самом деле кому-нибудь $6\frac{1}{7}$-пунктовый шрифт, который на три четверти — шрифт Baskerville, и на четверть — Helvetica?
>
> *Дональд Э. Кнут. Идея Мета-фонта*[1]

Маленькая девочка Оля добралась до полки, на которой стоят флаконы с маминими любимыми духами, и начала заниматься своим любимым занятием — переливанием жидкостей из одного флакона в другой. Когда наконец мама застала её за этим занятием, прекрасная коллекция духов была уже безнадёжно утрачена.

К счастью, Оля — аккуратная девочка, поэтому все свои действия она записывала на бумажку. Помогите ей успокоить маму: определите, каков состав духов в первом (мамином любимом) флаконе, чтобы мама смогла придумать этой смеси новое название и рассказывать всем, какие прекрасные духи она смогла сделать вместе с дочерью.

Считайте, что Оля не пролила ни одной капли, а также что она тщательно встряхивала флаконы после каждого переливания. Учтите, что в маминых флаконах порой не видно, есть ли там жидкость, и потому Оля иногда могла пытаться переливать духи из пустого флакона (в результате, естественно, ничего не переливалось).

Формат входных данных

На первой строке входного файла находятся два числа N и M — количество флаконов и число типов маминых любимых духов соответственно ($2 \leqslant N \leqslant 100$; $1 \leqslant M \leqslant 100$). Далее следуют N строк, на i-ой из которых находятся два числа — тип L_i и объем V_i духов, находившихся изначально в i-ом флаконе ($1 \leqslant L_i \leqslant M$; $0 \leqslant V_i \leqslant 1000$). Возможно, что в нескольких флаконах находились духи одного и того же типа; возможно, что какого-то типа вообще не было на полке.

Далее во входном файле следует строка с числом K — количеством совершённых переливаний ($1 \leqslant K \leqslant 1000$). За ней следуют K строк, на k-ой из которых находятся три числа S_k, T_k и A_k — номера флаконов, откуда и куда переливала Оля при k-ом переливании, и количество перелитой жидкости (в процентах от количества жидкости в S_k-ом флаконе перед переливанием). Гарантируется, что $1 \leqslant S_k, T_k \leqslant N$, что $S_k \neq T_k$, и что $0 \leqslant A_k \leqslant 100$. Все числа во входном файле целые.

[1] ... but does anybody really need a $6\frac{1}{7}$-point font that is one fourth of the way between Baskerville and Helvetica? — Donald E. Knuth, The Concept of a Meta-Font

Формат выходных данных

В выходной файл выведите M чисел — процентное содержание всех видов духов (от первого до M-ого) в первом флаконе после последнего переливания. Выводите результат с точностью не меньше двух знаков после запятой. Гарантируется, что после последнего переливания первый флакон оказался непустым.

Пример

Входной файл	*Выходной файл*
3 2 1 100 2 200 1 500 2 3 2 20 2 1 50	60.00 40.00

Решение

Эта незамысловатая задача была самой простой на олимпиаде, для её решения достаточно было промоделировать действия маленькой девочки Оли.

В двумерном массиве V для каждого флакона будем хранить состав его содержимого: в ячейке V_{ij} будем хранить объём j-го типа духов в i-м флаконе. Изначально в каждом флаконе присутствует только один тип духов, поэтому в каждой строке массива будет лишь одно ненулевое значение. При переливании из флакона s во флакон t нам достаточно уменьшить значения V_{sj} и соответственно увеличить V_{tj} для всех j от 1 до M. После выполнения всех переливаний нам остаётся вывести в выходной файл процентные доли каждого типа духов, что также делается легко.

Пример правильной программы

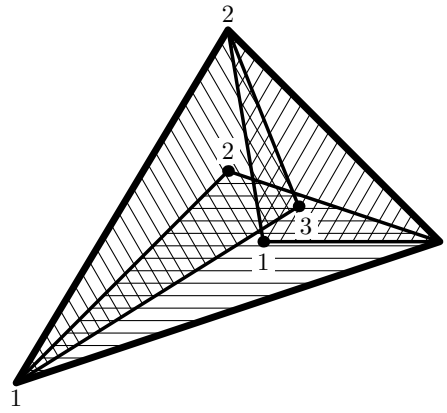
```
const MAX_N = 100;
const MAX_M = 100;
var liq: array[1..MAX_N, 1..MAX_M] of real;
    a, b, s: real;
    n, m, k, i, j, p, q: integer;
begin
  assign(input,'flow.in'); reset(input);
  assign(output,'flow.out'); rewrite(output);
  read(n,m);
  for i:=1 to n do begin
    read(j);
    read(liq[i,j]);
  end;
  read(k);
  for i:=1 to k do begin
    read(p,q,b);
    for j:=1 to m do begin
      a:=liq[p,j]*b/100;
      liq[q,j]:=liq[q,j]+a;
      liq[p,j]:=liq[p,j]-a;
    end;
  end;
  s:=0;
  for i:=1 to m do s:=s+liq[1, i];
  for i:=1 to m do
    write(liq[1,i]/s*100:0:2,' ');
  writeln;
  close(output);
  close(input);
end.
```

Задача 2. Освещение двора

Входной файл	`light.in`
Выходной файл	`light.out`
Ограничение по времени	1 с
Максимальный балл за задачу	100

Фирма, в которой всё ещё работает ваш друг, собирается расширяться на новые маршруты, и потому приобрела новую площадку для ночного отстоя автобусов. Площадка раньше относилась к военной части, поэтому она имеет необычную форму — форму треугольника, огороженного забором.

Для того, чтобы освещать площадку ночью, на ней надо установить несколько прожекторов. К счастью, у фирмы как раз в наличие есть три регулируемых прожектора, а на территории площадки обнаружились три высоких столба. Было решено на каждый столб повесить по прожектору и отрегулировать их так, чтобы каждый прожектор освещал ровно одну из трёх сторон забора: один прожектор должен освещать одну сторону забора, другой — другую, третий — третью. Никакой прожектор не должен освещать ни миллиметра «чужой» стены.



Конечно, прожекторы должны освещать не только забор, но вообще всю территорию площадки, поэтому возникла проблема: надо определить, какой прожектор на какую сторону забора направить. Зная ваши высокие навыки в решении подобных задач, фирма обратилась к вам за помощью. Напишите программу, которая будет решать эту задачу.

Естественно, каждый прожектор освещает не только стену, но и всю соответствующую часть двора — треугольник с вершиной в месте, где находится столб, на котором висит прожектор, и основанием, совпадающим с соответствующей стороной забора. Будем считать, что стороны этого треугольника тоже освещены прожектором. Тенью от столбов пренебрегайте.

Формат входных данных

Первая строка входного файла содержит одно число T — количество тестовых примеров, которые идут дальше ($1 \leqslant T \leqslant 1000$).

Далее следуют описания T примеров. В каждом сначала идут шесть чисел x_1, y_1, x_2, y_2, x_3, y_3 — координаты вершин площадки, после чего идут шесть чисел X_1, Y_1, X_2, Y_2, X_3, Y_3 — координаты столбов. Гарантируется, что все столбы находятся строго внутри площадки. Гарантируется, что площадь площадки строго больше нуля. Гарантируется, что никакие два столба не совпадают. Все координаты во входном файле целые и не превосходят по модулю 800.

Формат выходных данных

Выведите в выходной файл T строк по три числа в каждой: для каждого примера выведите номер стороны забора, на которую должен светить прожектор, на-

ходящийся на первом столбе, потом номер стороны, на которую должен светить прожектор со второго столба, и наконец номер стороны, на которую должен светить прожектор с третьего столба.

Первая сторона — та, которая соединяет вершины (x_1, y_1) и (x_2, y_2), вторая — (x_2, y_2) и (x_3, y_3), третья — (x_3, y_3) и (x_1, y_1).

Если решения не существует, выведите в соответствующую строку выходного файла три минус единицы: '`-1 -1 -1`'.

Пример

Входной файл	*Выходной файл*
2 0 0 6 10 12 4 7 4 6 6 8 5 -10 -10 0 10 10 -10 0 1 0 -1 1 1	2 3 1 3 2 1

Примечание: Первый пример соответствует рисунку.
Примечание: Среди тестов будут такие, в которых $T \leqslant 10$; суммарная стоимость таких тестов будет 70 баллов.

Решение

Для каждого теста переберём все варианты ответа и найдём, какой из них нам подходит (таких вариантов всего $6 = (3!)$).

Пусть мы выбрали, какие прожектора куда направлены. Теперь нам нужно проверить, освещён ли треугольник полностью, или в нем есть хотя бы одна неосвещённая точка. Для этого сделаем следующее: создадим некоторый список точек L, в который включим: вершины исходного треугольника, точки расположения столбов, а также точки попарных пересечений границ областей, освещённых каждым прожектором (отрезков, соединяющих точку, где находится прожектор, с концами стороны, которую этот прожектор освещает). Затем для каждой тройки вершин из этого списка найдём их центр масс и проверим, освещён ли он при данной конфигурации прожекторов; для этого нужно проверить, попадает ли эта точка в зону действия какого-нибудь из прожекторов. Утверждается, что проверки только точек такого вида достаточно: если все они освещены, то вся площадка освещена полностью и текущий вариант ответа нас полностью устраивает.

Действительно, точек такого вида достаточно из следующих соображений: если провести все отрезки — границы освещённых областей, то весь треугольник распадётся на многоугольные связные кусочки, каждый из которых либо целиком освещён, либо нет. Вершинами этих кусочков могут быть только точки из нашего списка точек L. Если какая-то область не освещена, то очевидно, что среди вершин этой области найдутся три, центр масс которых не освещён; так как мы рассматриваем все тройки вершин из списка L, то и эту тройку мы рассмотрим.

Заметим также, что задачу можно было решать и несколько быстрее. А именно, из рассуждений предыдущего абзаца следует, что если на площадке есть неосвещённые точки, то есть точка из списка L, в бесконечно малой окрестности которой есть неосвещённые точки. Поэтому просто для каждой точки из списка L проверим это. А именно, рассмотрим очередную точку и переберём все прожекторы. Для

каждого прожектора возможны три случая: либо точка находится строго внутри области, освещённой этим прожектором — тогда окрестность точки тоже освещена этим же прожектором и можно переходить к следующей точке; либо точка находится строго вне области — тогда прожектор не освещает никакую часть окрестности и этот прожектор можно не учитывать при проверке этой точки; либо точка находится на границе этой области. В последнем случае в окрестности точки граница области, освещённой этим прожектором, представляет собой два луча — все такие лучи добавим в один список для данной точки, отсортируем по полярному углу и достаточно легко проверим, что полный угол вокруг этой точки освещён хотя бы каким-нибудь прожектором.

Именно последний вариант решения и приведён в примере программы ниже; отметим также, что оно реализовано полностью в целых числах, что позволяет избавиться от проблем с ограниченной точностью вещественных чисел. Примеры решений, реализующих первый подход, можно скачать с сайта `http://olympiads.nnov.ru` в архиве решений жюри (решения `light_pk_real_2.pas`, `light_ve.pas`).

Пример правильной программы

```
{$mode objfpc}
var f,g:text;
type tFrac=record x,y:int64; end;
const perm:array[1..6,1..3] of longint=((1,2,
3),(1,3,2),(2,1,3),(2,3,1),(3,2,1),(3,1,2));
var x,y:array[1..4] of longint;
    xx,yy:array[1..3] of longint;
    t,tests:longint;
    i,j,k:longint;
    sector:array[1..3] of record
        x0,y0,x1,y1,x2,y2:longint; end;
    ok:boolean;
    was:boolean;

function gcd(a,b:int64):int64;
begin
if b=0 then gcd:=a
else gcd:=gcd(b, a mod b);
end;
operator :=(a:tFrac)c:extended;
begin c:=a.x/a.y; end;
operator :=(a:int64)c:tFrac;
begin c.x:=a; c.y:=1; end;
procedure norm(var a:tFrac);
var d:int64;
begin  d:=gcd(a.x,a.y);
a.x:=a.x div d; a.y:=a.y div d;
if a.y<0 then
begin a.x:=-a.x; a.y:=-a.y;end;
end;
operator +(a,b:tFrac)c:tFrac;
var m1,m2:int64;
    d:int64;
begin  d:=gcd(a.y,b.y);
m1:=b.y div d;  m2:=a.y div d;
c.x:=a.x*m1+b.x*m2;  c.y:=a.y*m1;
norm(c);
end;
operator -(a,b:tFrac)c:tFrac;
var m1,m2:int64;
    d:int64;
begin  d:=gcd(a.y,b.y);
m1:=b.y div d;  m2:=a.y div d;
c.x:=a.x*m1-b.x*m2;  c.y:=a.y*m1;
norm(c);
end;
operator *(a,b:tFrac)c:tFrac;
begin
c.x:=a.x*b.x;  c.y:=a.y*b.y;
norm(c);
end;
operator /(a,b:tFrac)c:tFrac;
begin
c.x:=a.x*b.y;  c.y:=a.y*b.x;
norm(c);
end;
operator =(a,b:tFrac)c:boolean;
begin norm(a);  norm(b);
c:=(a.x=b.x)and(a.y=b.y);
end;
function less0(a:tFrac):boolean; inline;
begin  result:=a.x<0;  end;
function gt0(a:tFrac):boolean; inline;
begin  result:=a.x>0;  end;
function eq0(a:tFrac):boolean; inline;
begin  result:=a.x=0;  end;
operator <(a,b:tFrac)c:boolean;
begin  c:=less0(a-b);  end;
operator >(a,b:tFrac)c:boolean;
begin  c:=b<a;  end;
operator <=(a,b:tFrac)c:boolean;
var t:tFrac;
begin  t:=a-b;  c:=t.x<=0;  end;
operator >=(a,b:tFrac)c:boolean;
var t:tFrac;
begin  t:=a-b;  c:=t.x>=0;  end;

function vects(x1,y1,x2,y2:int64):longint;
```

```
                                    overload;
var v:int64;
begin
v:=x1*y2-x2*y1;
if v<0 then result:=-1
else if v=0 then result:=0
else result:=1;
end;

function vects(x1,y1,x2,y2:tFrac):longint;
                                    overload;
var v:tFrac;
begin
v:=x1*y2-x2*y1;
if less0(v) then result:=-1
else if eq0(v) then result:=0
else result:=1;
end;

function max(a,b:int64):int64;
begin
if a>b then result:=a
else result:=b;
end;
function min(a,b:int64):int64;
begin
if a<b then result:=a
else result:=b;
end;

function intersect(x1,y1,x2,y2,x3,y3,
    x4,y4:int64;var xx,yy:tFrac):boolean;
var d:int64;
    t,s:tFrac;
begin
xx:=0;
yy:=0;
d:=(y3-y4)*(x2-x1)-(x3-x4)*(y2-y1);
if d=0 then begin
   result:=false;
   exit;
end;
t:=tFrac(((x3-x1)*(y3-y4)-(x3-x4)*(y3-y1)))/d;
s:=tFrac(((x2-x1)*(y3-y1)-(x3-x1)*(y2-y1)))/d;
if (0<t)and(t<1)and(0<s)and(s<1) then begin
   result:=true;
   xx:=x1+(x2-x1)*t;
   yy:=y1+(y2-y1)*t;
end else result:=false;
end;

function check(x,y:tFrac):boolean;
var dir:array[0..6] of record
         dx,dy:tFrac;t:longint; end;
    ndir:longint;
    i,j:longint;
    s1,s2:longint;
    bal:longint;

procedure adddir(xx,yy:tFrac;t:longint);
begin
inc(ndir);
dir[ndir].dx:=xx-x;
dir[ndir].dy:=yy-y;
dir[ndir].t:=t;
end;

function quat(i:longint):longint;
begin
if (dir[i].dx>=0)and(dir[i].dy>0)then quat:=1;
if (dir[i].dx<0)and(dir[i].dy>=0)then quat:=2;
if (dir[i].dx<=0)and(dir[i].dy<0)then quat:=3;
if (dir[i].dx>0)and(dir[i].dy<=0)then quat:=4;
end;

procedure addpair(x1,y1,x2,y2:longint);
begin
adddir(x1,y1,1);
adddir(x2,y2,-1);
if quat(ndir)<quat(ndir-1) then
    inc(bal);
end;

function less(i,j:longint):boolean;
var q1,q2:longint;
    s:longint;
begin
less:=false;
q1:=quat(i);
q2:=quat(j);
if q1<q2 then less:=true;
s:=vects(dir[i].dx,dir[i].dy,
            dir[j].dx,dir[j].dy);
if s>0 then less:=true;
if s=0 then
    if dir[i].t>dir[j].t then less:=true;
end;

begin
ndir:=0;
bal:=0;
for i:=1 to 3 do begin
    with sector[i] do begin
        s1:=vects(x1-x0,y1-y0,x-x0,y-y0);
        s2:=vects(x-x0,y-y0,x2-x0,y2-y0);
        if (s1>0)and(s2>0) then begin
             // completely inside
            check:=true;
            exit;
        end;
        if (s1<0)or(s2<0) then
           continue; // completely outside
        if (s1=0)and(s2<>0) then
             // on first side
           addpair(x1,y1,x0,y0);
        if (s1<>0)and(s2=0) then
             // on second side
            addpair(x0,y0,x2,y2);
        if (s1=0)and(s2=0) then
              // obviously vertex
            addpair(x1,y1,x2,y2);
    end;
end;
for i:=1 to ndir do
```

```
    for j:=1 to ndir-1 do
        if less(j+1,j) then begin
            dir[0]:=dir[j+1];
            dir[j+1]:=dir[j];
            dir[j]:=dir[0];
        end;
for i:=1 to ndir do begin
    bal:=bal+dir[i].t;
    if bal=0 then begin
        check:=false;
        exit;
    end;
end;
check:=true;
end;

procedure swap(var a,b:longint);
var t:int64;
begin
t:=a;
a:=b;
b:=t;
end;

procedure checkintersect(x1,y1,x2,y2,x3,y3,
                                  x4,y4:int64);
var xx,yy:tFrac;
begin
if intersect(x1,y1,x2,y2,x3,y3,x4,y4,xx,yy)
then ok:=ok and check(xx,yy);
end;

begin
assign(f,'light.in');reset(f);
assign(g,'light.out');rewrite(g);
read(f,tests);
for t:=1 to tests do begin
    for i:=1 to 3 do read(f,x[i],y[i]);
    x[4]:=x[1];
    y[4]:=y[1];
    for i:=1 to 3 do read(f,xx[i],yy[i]);
    was:=false;
    for i:=1 to 6 do begin
        for j:=1 to 3 do begin
            with sector[j] do begin
                x0:=xx[j];
                y0:=yy[j];
                x1:=x[perm[i,j]];
                y1:=y[perm[i,j]];
                x2:=x[perm[i,j]+1];
                y2:=y[perm[i,j]+1];
                if vects(x1-x0,y1-y0,
                  x2-x0,y2-y0)<0 then begin
                    swap(x1,x2);
                    swap(y1,y2);
                end;
            end;
        end;
        ok:=true;
        for j:=1 to 3 do begin
            ok:=ok and check(xx[j],yy[j]);
            for k:=j+1 to 3 do begin
checkintersect(sector[j].x0,sector[j].y0,
  sector[j].x1,sector[j].y1, sector[k].x0,
  sector[k].y0,sector[k].x1,sector[k].y1);
checkintersect(sector[j].x0,sector[j].y0,
  sector[j].x1,sector[j].y1, sector[k].x0,
  sector[k].y0,sector[k].x2,sector[k].y2);
checkintersect(sector[j].x0,sector[j].y0,
  sector[j].x2,sector[j].y2, sector[k].x0,
  sector[k].y0,sector[k].x1,sector[k].y1);
checkintersect(sector[j].x0,sector[j].y0,
  sector[j].x2,sector[j].y2, sector[k].x0,
  sector[k].y0,sector[k].x2,sector[k].y2);
            end;
        end;
        if ok then begin
            for j:=1 to 3 do
                write(g,perm[i,j],' ');
            writeln(g);
            was:=true;
            break;
        end;
    end;
    if not was then writeln(g,'-1 -1 -1');
end;
close(f);
close(g);
end.
```

Задача 3. Совершенно несчастливые билеты

Входной файл	tickets.in
Выходной файл	tickets.out
Ограничение по времени	1 с
Максимальный балл за задачу	100

Миша часто ездит в маршрутках. Миша законопослушный, поэтому он всегда покупает билет. Каждый билет в маршрутке имеет номер — $2N$-значное десятичное число, возможно, с ведущими нулями. После покупки билета Миша всегда проверяет, счастливый ли достался ему билет. Счастливым Миша считает такой билет, у которого сумма первых N цифр номера равна сумме последних N цифр. Если билет оказывается несчастливым, Миша ищет расстояние до ближайшего счастливого, т.е. минимальное число x такое, что если к номеру билета, полученного

Мишей, прибавить или отнять x (при этом, разумеется, полученный номер должен быть корректным номером билета, т.е. должен быть не меньше нуля и не больше $10^{2N} - 1$), то получившийся номер должен быть номером счастливого билета.

Билеты, у которых это расстояние максимально среди всех возможных, Миша называет совершенно несчастливыми. Мише очень интересно, сколько всего существует совершенно несчастливых билетов и какие номера у этих билетов. Но так как Миша плохо учился в школе, он не знает, как решать такую задачу, поэтому он обратился за помощью к вашему другу, специалисту по маршруткам. Он же перенаправил Мишу к вам.

Формат входных данных

Во входном файле содержится единственное число N ($1 \leqslant N \leqslant 100\,000$) — половина длины номера билетов.

Формат выходных данных

В первой строке выходного выведите одно число k — количество совершенно несчастливых билетов. В последующих k строках выведите номера билетов в порядке возрастания. Номера нужно выводить с ведущими нулями, если таковые есть.

Пример

Входной файл	*Выходной файл*
2	4 0050 0051 9948 9949

Примечание: В примере расстояние от билета с номером 0050 до ближайшего счастливого равно 50: если из этого номера вычесть 50, получится 0000 — счастливый номер. Аналогично, от билета 0051 расстояние до ближайшего счастливого тоже равно 50: если к номеру прибавить 50, получится 0101. У билетов 9948 и 9949 расстояние до ближайшего счастливого также равно 50; других билетов с таким расстоянием нет. Билетов с бóльшим расстоянием тоже нет, поэтому эти четыре билета и только они являются совершенно несчастливыми.

Решение

Сначала найдём x — расстояние от совершенно несчастливого билета до ближайшего счастливого. Легко убедиться, что $x = 5 \cdot 10^{N-1}$. Действительно, рассмотрим билеты с номерами $\overbrace{0\ldots0}^{2N}$ и $\overbrace{0\ldots0}^{N-1}1\overbrace{0\ldots0}^{N-1}1$. Очевидно, что оба они счастливые и между ними нет счастливых. Следовательно, $x \geq 5 \cdot 10^{N-1}$. Но, рассматривая билеты с номерами $\overline{AA}$ и $\overline{(A+1)(A+1)}$, где A — N-значное число (возможно, с ведущими нулями), можно понять, что бóльшее x мы не сможем получить, т.к. для любого билета между этими расстояние до ближайшего счастливого не больше $5 \cdot 10^{N-1}$.

Дальше возможны два случая: $N = 1$ и $N > 1$. В первом из них все номера всех счастливых билетов представимы в виде $\overline{AA}$, между каждыми двумя такими билетами есть пара совершенно несчастливых билетов, расстояние от которых до

ближайших счастливых равно 5. Поэтому всего есть 18 совершенно несчастливых билетов.

В другом случае легко понять, что есть как минимум четыре совершенно несчастливых билета: $\overbrace{0\ldots0}^{N}5\overbrace{0\ldots0}^{N-1}$ и $\overbrace{0\ldots0}^{N}5\overbrace{0\ldots0}^{N-2}1$, и им симметричные. Докажем, что других нет, для этого рассмотрим все билеты с номерами вида $\overline{AA}$ и $\overline{(A+1)(A+1)}$, где A — N-значное число, причём $0 < A < 10^N - 1$. Докажем, что между ними есть хотя бы один счастливый билет. Если число A не представимо в виде $\overbrace{9\ldots9}^{N}X0\ldots0$, то существует бóльшее число с такой же суммой цифр, как и A. В противном случае рассмотрим число $A+1$: из условий на A следует, что младшая цифра $A+1$ отлична от 9, а старшая отлична от 0. Тогда можем уменьшить старшую цифру $A+1$ на 1 и прибавить 1 к последней цифре, тем самым получив меньшее $A+1$ число с такой же суммой цифр. Таким образом всегда есть счастливый билет между билетами с номерами $\overline{AA}$ и $\overline{(A+1)(A+1)}$, причём легко убедиться в том, что расстояние от него до $\overline{AA}$ и $\overline{(A+1)(A+1)}$ строго больше 1, поэтому для любого билета между $\overline{AA}$ и $\overline{(A+1)(A+1)}$ расстояние до ближайшего счастливого строго меньше x.

Пример правильной программы

```
var f:text;
    n:integer;
    i:integer;
begin
  assign(f,'tickets.in');reset(f);
  read(f,n);
  close(f);
  assign(f,'tickets.out');rewrite(f);
  if n=1 then begin
    writeln(f,18);
    writeln(f,'05');
    writeln(f,'06');
    for i:=1 to 8 do begin
      writeln(f,i*11+5);
      writeln(f,i*11+6);
    end;
  end
  else begin
    writeln(f,4);
    for i:=1 to n do
      write(f,0);
    write(f,5);
    for i:=1 to n-1 do
      write(f,0);
    writeln(f);
    for i:=1 to n do
      write(f,0);
    write(f,5);
    for i:=1 to n-2 do
      write(f,0);
    writeln(f,1);

    for i:=1 to n do
      write(f,9);
    write(f,4);
    for i:=1 to n-2 do
      write(f,9);
    write(f,8);
    writeln(f);

    for i:=1 to n do
      write(f,9);
    write(f,4);
    for i:=1 to n-2 do
      write(f,9);
    write(f,9);
    writeln(f);
  end;
  close(f);
end.
```

Задача 4. BubbleGum

Входной файл	bubble.in
Выходной файл	bubble.out
Ограничение по времени	1 с
Максимальный балл за задачу	100

Мальчик Влад недавно побывал в Японии и привёз оттуда новую жевательную резинку. Вернувшись в университет после поездки, на первой же паре Влад раздал

жвачку всем своим $(N-1)$ однокурсникам и взял одну себе. Дождавшись момента, когда лектор отвернулся к доске, на счёт «три-четыре» все N студентов дружно начали надувать пузыри. Известно, что i-й студент надувает пузырь до максимально возможного размера за время t_i, после чего пузырь мгновенно лопается, и студент начинает надувать пузырь заново с той же скоростью.

Всё это время преподаватель настолько увлечён тонкостями квантового математического анализа, что не слышит ничего происходящего в аудитории. И только когда все N пузырей лопнут *одновременно*, преподаватель услышит шум и обернётся. И уж тогда студентам достанется, а больше всех тому, кто принёс на пару N жевательных резинок.

Определите, сколько времени студенты смогут наслаждаться надуванием пузырей, не замечаемые преподавателем.

Например, если $N = 2$, $t_1 = 2$, $t_2 = 3$, то будет происходить следующее:

Первый студент надувает пузырь с момента времени $t = 0$ до момента времени $t = 2$, потом пузырь лопается, и он надувает пузырь заново — с момента времени $t = 2$ до момента времени $t = 4$, а потом ещё раз — с момента времени $t = 4$ до $t = 6$.

Второй студент надувает пузырь с $t = 0$ до $t = 3$ и ещё раз с $t = 3$ до $t = 6$.

В момент $t = 6$ пузыри лопаются одновременно у обоих студентов, преподаватель оборачивается и говорит: «Всё, Влад! Ты меня достал!».

Формат входных данных

На первой строке входного файла находится одно целое число N — количество студентов ($1 \leqslant N \leqslant 10\,000$). На второй строке входного файла находятся N целых чисел $t_1, t_2, \ldots, t_N$. Гарантируется, что $1 \leqslant t_i \leqslant 1000$.

Формат выходных данных

Выведите в выходной файл одно число — время, в течение которого студенты во главе с Владом могут наслаждаться безнаказанным надуванием пузырей.

Пример

Входной файл	*Выходной файл*
2 2 3	6
1 1	1
2 16 1	16
3 627 182 85	9699690

Решение

Задача решалась достаточно просто. Очевидно, что ответом на поставленный в условии вопрос должно было стать наименьшее общее кратное (НОК) N чисел t_1, t_2, …, t_N. Единственной технической проблемой решения могли стать ограничения, в пределах которых НОК этих чисел могло не убраться ни в один из стандартных типов, представленных в языках программирования. Требовалось реализовать

длинную арифметику, то есть создать свой тип «длинных» чисел. Из арифметических операций достаточно было реализовать только умножение длинного числа на обычное «короткое». Такое умножение можно написать «столбиком», как учат в школе. Покажем, как можно реализовать реализовать НОК N чисел через умножение. Разложим предварительно каждое из N чисел на простые множители (ограничения позволяют проводить проверку на простоту самыми элементарными способами). Затем выбором максимальной степени, в которой каждый из этих простых множителей входит в данные N чисел, получим разложение НОК этих чисел на простые множители. Для получения результата необходимо перемножить эти степени, то есть реализовать умножение «длинного» числа на «короткое».

Пример правильной программы

```
const maxl=100;
      base=100000;
      blen=5;
type tlong=array[1..maxl] of longint;
var f:text;
    n:integer;
    t:array[1..10000] of longint;
    i,j:longint;
    ans:tlong;
    pow:longint;
    was:array[1..1000] of integer;
    d:integer;
label 1;

procedure mul(var a:tlong; b:integer);
var i:integer;
begin
  for i:=1 to maxl do
    a[i]:=a[i]*b;
  for i:=1 to maxl-1 do begin
    a[i+1]:=a[i+1]+a[i] div base;
    a[i]:=a[i] mod base;
  end;
end;

function fmt(a:longint):string;
var s:string;
begin
  str(a,s);
  while length(s)<blen do
    s:='0'+s;
  fmt:=s;
end;

begin
  assign(f,'bubble.in');reset(f);
  read(f,n);
  for i:=1 to n do
    read(f,t[i]);
  close(f);
  d:=0;
  fillchar(was,sizeof(was),0);
  {удалим повторяющиеся числа}
  for i:=1 to n do
    if was[t[i]]=0 then begin
      t[i-d]:=t[i];
      was[t[i]]:=1;
    end
    else inc(d);
  n:=n-d;
  fillchar(ans,sizeof(ans),0);
  ans[1]:=1;
  for i:=2 to 1000 do begin
    for j:=2 to i-1 do
      if i mod j=0 then goto 1;
    pow:=1;
    for j:=1 to n do begin
      while t[j] mod (pow*i)=0 do
        pow:=pow*i;
    end;
    mul(ans,pow);
    1:
  end;
  assign(f,'bubble.out');rewrite(f);
  for j:=maxl downto 1 do
    if ans[j]<>0 then
      break;
  write(f,ans[j]);
  for j:=j-1 downto 1 do
    write(f,fmt(ans[j]));
  close(f);
end.
```

Задача 5. Шоколадки

Входной файл	choco.in
Выходной файл	choco.out
Ограничение по времени	1 с
Максимальный балл за задачу	100

У мальчика Васи есть N шоколадок (возможно, разного веса). Вася пригласил к себе в гости K своих друзей и хочет подарить им шоколадки. Чтобы никому из

друзей не было обидно, Вася решил раздать шоколадки так, чтобы каждому другу досталось одно и то же количество шоколада (т.е. суммарный вес шоколадок, доставшихся каждому другу, должен быть одинаковым). Вася может раздать все свои шоколадки, может раздать лишь часть, но, поскольку он — очень гостеприимный мальчик, он не хочет оставлять друзей совсем без шоколада (т.е. сумма весов шоколадок, доставшихся каждому другу, должна быть строго положительной). Все шоколадки красиво упакованы, т.е. делить их на части нельзя.

Определите, сколько у Васи есть способов раздать шоколад своим друзьям. Два способа считайте различными тогда и только тогда, когда существует шоколадка, которая в одном способе досталась некоторому другу, а в другом — другому другу или вовсе не была отдана друзьям.

Формат входных данных

В первой строке входного файла находятся два натуральных числа N и K ($1 \leqslant N \leqslant 15$, $1 \leqslant K \leqslant 15$) — количество шоколадок у Васи и количество друзей, которых Вася пригласил в гости. Во второй строке содержатся N натуральных чисел — веса шоколадок. Ни один из весов не превосходит 1000.

Формат выходных данных

Выведите в выходной файл одно число — количество способов раздать шоколадки друзьям.

Пример

Входной файл	*Выходной файл*
5 4 1 2 1 1 1	24
3 2 1 1 2	4

Примечание: Во втором примере возможные распределения шоколадок следующие:

1) Первому другу дать шоколадку номер 1, второму — номер 2;
2) Первому другу дать шоколадку номер 2, второму — номер 1;
3) Первому другу дать шоколадку номер 3, второму — шоколадки номер 1 и 2;
4) Первому другу дать шоколадки номер 1 и 2, второму — номер 3.

Решение

При решении задачи нам пригодятся значения суммы весов для каждого подмножества шоколадок. Каждое из этих подмножеств можно представить в виде одного числа — маски $mask$ из N бит (если в бите i стоит 1, то считаем, что шоколадка i входит в данное подмножество). Сосчитать все значения $sum[mask]$ можно просто перебрав все подмножества, а для каждого подмножества — все шоколадки, и просуммировав веса шоколадок, соответствующих разрядам, в которых стоит 1.

Далее для нахождения ответа воспользуемся методом динамического программирования. Для каждой маски $mask$ и числа друзей $M \leqslant K$ обозначим через $ans[M, mask]$ количество способов распределить все шоколадки подмножества $mask$ между последними M друзьями, считая, что мы обязаны распределить *все* шоколадки из маски $mask$. Понятно, что для любого значения $mask > 0$ выполняется

$ans[0, mask] = 0$ (мы никому не можем отдать неиспользованные шоколадки), а $ans[0, 0] = 1$. Пусть теперь $M > 0$. Нам необходимо определиться, какой набор шоколадок cur мы отдадим другу M. Набор cur будет допустимым, если, во-первых, он является подмножеством $mask$, во-вторых, вес шоколадок этого набора равен $sum[mask]/M$ (поскольку мы должны поделить все шоколадки из рассматриваемого множества $mask$ поровну между M друзьями).

Таким образом, для нахождения значения $ans[M, mask]$ можно перебрать все подмножества cur множества $mask$ и для тех наборов, которые обладают требуемым весом шоколадок, увеличить значение $ans[M, mask]$ на $ans[M-1, mask-cur]$ — отдав другу M набор cur, мы должны отдать оставшимся $M - 1$ друзьям набор $mask - cur$. Перебрав подобным образом значения M в порядке возрастания $1 \leqslant \leqslant M \leqslant K$, а для каждого значения M перебрав все маски, мы найдём значения $ans[K, mask]$ для каждого подмножества $mask$. Тогда ответом на задачу будет сумма значений $ans[K, mask]$ для всех подмножеств $mask > 0$ ($mask$ — непустой набор шоколадок, который мы хотим раздать K друзьям). При этом для подсчёта результата $ans[M, mask]$ мы выполним порядка 2^t итераций цикла перебора подмасок, где t — количество единичных бит в числе $mask$ (количество шоколадок в подмножестве). Несложно проверить, что сумма значений 2^t по всем маскам $mask$, состоящим из N бит, равна 3^N. Значит, всего для расчёта значений ans будет выполнено порядка $K \cdot 3^N$ операций. При этом вычисление значений таблицы ans можно значительно ускорить, поскольку можно не рассматривать состояния $ans[M, mask]$ если $sum[mask]$ не делится на M — значение будет равно 0 (в этом случае мы не можем поделить шоколадки поровну). Ещё порядка $N \cdot 2^N$ операций будет выполнено для нахождения значений sum, и порядка 2^N операций — для нахождения итогового ответа.

Примечание 1: несложно заметить, что перебрать все подмножества cur множества $mask$ за время порядка 2^t можно следующим образом

```
cur:=mask;
repeat
  cur:=((cur-1) and mask)
until cur=0;
```

Примечание 2: можно найти значения sum, выполнив не $N \cdot 2^N$, а порядка 2^N операций, но в данной задачи это существенной роли не играет, поскольку вычисление значений массива ans все равно занимает заметно более долгое время.

Пример правильной программы

```
const maxn=15;
      maxm=15;
var f:text;
    n,m:longint;
    i,j,k:longint;
    a:array[0..maxn-1] of longint;
    sum:array[0..1 shl maxn -1] of longint;
    ans:array[0..maxm,0..1 shl maxn-1]
                                of int64;
    s:longint;
    res:int64;

begin
assign(f,'choco.in');reset(f);
read(f,n,m);
for i:=0 to n-1 do
    read(f,a[i]);
fillchar(sum,sizeof(sum),0);
for i:=0 to 1 shl n -1 do
    for j:=0 to n do
        if (i and (1 shl j)<>0) then
           sum[i]:=sum[i]+a[j];
```

```
fillchar(ans,sizeof(ans),0);
for i:=0 to 1 shl n-1 do
    ans[0,i]:=1;
for i:=1 to m do begin
    for j:=0 to 1 shl n -1 do begin
        s:=sum[j];
        if s mod i<>0 then continue;
        s:=s div i;
        k:=j;
        repeat
          if (sum[k]=s) then begin
             ans[i,j]:=ans[i,j]+
                  ans[i-1,j xor k];
          end;
          k:=((k-1) and j);
        until k=0;
    end;
end;
res:=0;
for i:=1 to 1 shl n-1 do//from 1, not from 0!
    res:=res+ans[m,i];
assign(f,'choco.out');rewrite(f);
writeln(f,res);
close(f);
end.
```

Задача 6. Два капитана

Входной файл	`treasure.in`
Выходной файл	`treasure.out`
Ограничение по времени	1 с
Максимальный балл за задачу	100

Капитаны Флинт и Джек Воробей нашли клад и хотят поделить его. Клад находится в шкатулке и состоит из чётного числа драгоценных камней. Капитан Флинт оценил i-ый камень в a_i пиастров, а Джек Воробей — в b_i долларов. Теперь они действуют следующим образом. Джек Воробей достаёт из шкатулки два камня, после чего Флинт забирает себе один из них (естественно, тот, у которого больше a_i). Оставшийся камень достаётся Воробью. После этого Джек Воробей достаёт ещё пару камней, и так далее: каждым ходом Воробей достает из шкатулки два камня, Флинт забирает себе камень с бóльшим a_i, оставшийся камень достается Воробью.

Джек Воробей знает все a_i, все b_i, а также может, доставая очередные два камня, подглядеть в шкатулку и выбрать, какие именно камни надо доставать. Помогите ему действовать так, чтобы доля Воробья была максимально возможной (т.е. чтобы сумма b_i полученных Воробьём камней была как можно больше).

По сравнению с камнями шкатулка ничего не стоит, поэтому её можно не учитывать при дележе.

Формат входных данных

Первая строка входного файла содержит одно целое число N — количество камней в кладе. Гарантируется, что N чётное и что $2 \leqslant N \leqslant 5000$. Далее следуют две строки по N целых чисел в каждой: сначала заданы все a_i, потом — все b_i. Гарантируется, что все a_i различны (т.е. что действия Флинта всегда однозначно определены). Гарантируется, что все a_i и все b_i положительны и не превосходят 400 000.

Формат выходных данных

В выходной файл выведите $N/2$ строк по два числа в каждой — пары камней, в том порядке, как их должен доставать из шкатулки Джек Воробей. Камни нумеруются начиная с 1.

Числа в пределах каждой пары можете выводить в произвольном порядке. Если есть несколько оптимальных решений, выводите любое.

Пример

Входной файл	*Выходной файл*
6 6 10 11 18 5 14 1 7 6 12 15 16	5 1 2 3 6 4
6 6 44 2 43 7 48 6 44 2 43 7 48	3 1 5 4 2 6

Примечание: Среди тестов будут такие, в которых каждый камень оба капитана оценивают одинаково: $a_i = b_i$ для каждого i (как во втором тесте из примера); суммарная стоимость таких тестов будет 40 баллов.

Решение

Для решения данной задачи надо было в первую очередь обратить внимание на тот факт, что конкретные значения величин a_i не важны, важен лишь их порядок. Поэтому отсортируем все камни по возрастанию a_i и далее на конкретные значения a_i обращать внимания не будем: капитан Флинт всегда будет просто выбирать камень с бóльшим номером.

Заметим кроме того, что конкретный порядок пар в выходном файле также не имеет значения. Поэтому задача на самом деле формулируется так: есть чётное количество (N) камней, стоимость i-ого камня — b_i. Надо сгруппировать камни в $N/2$ пар так, чтобы, выбрав в каждой паре камень с меньшим номером и просуммировав стоимости выбранных камней (эту сумму будем называть стоимостью решения), получить как можно больший результат.

В каждой такой паре камень с меньшим номером будем называть *левым*, а камень с бóльшим номером — *правым*. Тогда стоимость решения равна сумме стоимостей всех «левых» камней. Рассмотрим несколько первых камней — камни с номерами от 1 до k включительно. Очевидно, что для любого корректного решения «левых» камней среди них будет не меньше, чем «правых»: если среди этих камней какой-то является «правым», то соответствующий ему «левый» тоже должен находиться в этом множестве камней.

Докажем обратное утверждение. А именно, пусть мы каким-то (произвольным) образом $N/2$ камней назвали «левыми», а оставшиеся камни назвали «правыми», пока не объединяя их в пары никоим образом. Тогда если для каждого k выполняется следующее условие: среди первых k камней «левых» не меньше, чем «правых» (назовём это условие условием $*$), — то камни можно объединить в пары «левый-правый» так, чтобы в каждой паре камень с меньшим номером был «левым», а камень с бóльшим номером был «правым», — т. е. получить корректное решение со стоимостью, равной сумме стоимостей «левых» камней. Действительно, будем двигаться от камней с меньшими номерами к камням с бóльшими номерами, и каждому встреченному «левому» камню будем ставить в соответствие ещё не использованный «правый» камень с минимальным номером. Пусть в некоторый момент работы этого алгоритма некоторому «левому» камню (обозначим его номер j) мы поставим в соответствие «правый» камень с меньшим номером ($< j$). Положим $k = j - 1$. Тогда тот «правый» камень, который наш алгоритм поставил в соответствие камню

j, находится среди первых k камней. Кроме того, все «левые» камни, находящиеся среди первых k камней, нами уже обработаны, и парные к ним «правые» камни тоже находятся среди первых k камней (потому что мы используем «правые» камни в порядке возрастания их номеров, камень, который мы поставили в соответствие камню j, находится среди первых k камней — значит, и все использованные ранее «правые» камни находятся там же). Таким образом, количество «правых» камней среди первых k как минимум на один больше, чем количество «левых», что противоречит условию $*$. Следовательно, если это условие выполнено, то алгоритм построит корректное решение.

Таким образом, нам осталось выбрать $N/2$ «левых» камней так, чтобы условие $*$ было выполнено, и их сумма была максимальна. Эта задача уже легко решается методом динамического программирования. Для каждых i и j решим следующую задачу: из первых i камней выберем некоторые «левыми», остальные «правыми» так, чтобы «левых» камней было на j больше, чем «правых», чтобы условие $*$ выполнялось для всех $k < i$, и чтобы сумма стоимостей выбранных «левых» камней была максимальна. (Данная задача имеет решение не для всех пар (i, j), мы будем её рассматривать только для таких значений i и j, для которых требуемый выбор возможен.) Обозначим $ans[i, j]$ ответ на эту задачу. Тогда если $j = 0$, то несложно видеть, что i-ый камень должен быть «правым» (иначе условие $*$ не будет выполняться при $k = i - 1$), и ответ на задачу равен $ans[i - 1, 1]$; в противном случае есть два варианта: либо сделать последний камень «правым» (тогда ответ на задачу будет равен $ans[i - 1, j + 1]$), или сделать последний камень «левым» (ответ $ans[i - 1, j - 1] + b[i]$). Из этих двух вариантов надо выбрать дающий наибольшую стоимость — это и будет $ans[i, j]$.

По указанному алгоритму легко вычислить все значения матрицы ans. Отметим, что введя строку $i = 0$ и столбец $j = -1$ матрицы ans и заполнив их большими отрицательными значениями, и положив $ans[0, 0] = 0$, можно обойтись без особого рассмотрения случая $j = 0$ в программе. Ответ на основную задачу (стоимость решения) будет находиться в $ans[N, 0]$. После этого стандартными методами динамического программирования можно восстановить, какие камни надо выбрать как «левые», какие — как «правые», чтобы получить такую стоимость решения. Применив алгоритм разбиения на пары, описанный выше, можно построить ответ на исходную задачу — набор пар камней, которые должен доставать из шкатулки Джек Воробей. Естественно, в выходной файл надо выводить не номера камней, какими они стали после сортировки, а номера камней в том порядке, как они были заданы во входном файле.

Отметим, что в случае, когда для каждого i выполняется условие $a_i = b_i$, задачу можно решить и проще. А именно, достаточно отсортировать камни по убыванию (или по возрастанию) их оценок, после чего объединить в пары первый и второй камень, третий и четвёртый и т.д. Действительно, пусть два самых дорогих камня сто́ят m и n (при этом $m > n$), и пусть в некотором решении они не объединены в одну пару. Пусть они объединены с камнями стоимости p и q соответственно, тогда в этом решении Воробью из этих двух пар достаются камни суммарной стоимости $p + q$. Если же мы изменим решение — объединим в одну пару камни m и n, а в другую — камни p и q (а оставшиеся пары не изменим), то Воробью достанется

$n+\min(p,q)$. Поскольку n — второй по ценности камень, то $n > p$ и $n > q$, и несложно видеть, что $p + q < n + \min(p,q)$, следовательно, изначальное решение не было оптимальным. Следовательно, в оптимальном решении нам необходимо объединять два самых дорогих камня в одну пару. После этого мы остаёмся с такой же задачей, только с $N - 2$ камнями, поэтому легко видеть, что и далее нам надо объединять два самых дорогих оставшихся камня и т.д.

По условию, тесты, в которых $a_i = b_i$, были оценены суммарно в 40 баллов; указанное решение набирало слегка больше (43 балла) за счёт того, что оно проходило также некоторые другие тесты.

Пример правильной программы

```
const maxN=5000;
      maxA=400000;
var n:integer;
    aa,bb,a,b:array[1..maxN] of integer;
    iid,id:array[1..maxN] of integer;
    ans:array[0..maxN,-1..maxN+1] of integer;
    used:array[1..maxN] of (_none,_my,_his);
    i,j:integer;
    f:text;

procedure sort(l,r:integer);
var i,i1,i2,o:integer;
begin
  if l>=r then exit;
  o:=(l+r) div 2;
  sort(l,o);
  sort(o+1,r);
  i1:=l;
  i2:=o+1;
  for i:=l to r do
    if (i2>r)or((i1<=o)and(a[i1]<a[i2]))
    then begin
      aa[i]:=a[i1];
      bb[i]:=b[i1];
      iid[i]:=id[i1];
      inc(i1);
    end
    else begin
      aa[i]:=a[i2];
      bb[i]:=b[i2];
      iid[i]:=id[i2];
      inc(i2);
    end;
  for i:=l to r do begin
    a[i]:=aa[i];
    b[i]:=bb[i];
    id[i]:=iid[i];
  end;
end;

procedure out(i,j:integer);
begin
  if i=0 then exit;
  if ans[i-1,j-1]+b[i]>ans[i-1,j+1] then begin
    used[i]:=_my;
    out(i-1,j-1);
  end
  else out(i-1,j+1);
end;

begin
  assign(f,'treasure.in');reset(f);
  read(f,n);
  for i:=1 to n do
    read(f,a[i]);
  for i:=1 to n do
    read(f,b[i]);
  for i:=1 to n do
    id[i]:=i;
  sort(1,n);
  fillchar(ans,sizeof(ans),$80);{Big negative}
  ans[0,0]:=0;
  for i:=1 to n do
    for j:=0 to n do begin
      ans[i,j]:=ans[i-1,j+1];
      if ans[i-1,j-1]+b[i]>ans[i,j] then
        ans[i,j]:=ans[i-1,j-1]+b[i];
    end;
  fillchar(used,sizeof(used),ord(_none));
  out(n,0);
  assign(f,'treasure.out');rewrite(f);
  for i:=1 to n do begin
    if used[i]=_my then begin
      j:=i+1;
      while (j<=n)and(used[j]<>_none) do
        inc(j);
      used[j]:=_his;
      writeln(f,id[i],' ',id[j]);
    end;
  end;
  close(f);
end.
```

Содержание