Администрация города Нижнего Новгорода
Департамент образования и социально-правовой защиты детства
Нижегородский государственный университет им. Н. И. Лобачевского

Седьмая нижегородская городская олимпиада школьников по информатике

3 февраля 2011 г.

Нижний Новгород
2011

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

Оригинал-макет подготовлен в системе LaTeX 2$_\varepsilon$
с использованием набора шрифтов LH.

Седьмая городская олимпиада по информатике

3 февраля 2011 г. Городской ресурсный центр физико-математического образования в лице учредителей: Департамента образования и социально-правовой защиты детства администрации города Нижнего Новгорода, Нижегородского государственного университета им. Н. И. Лобачевского, Института прикладной физики РАН и МОУ Лицей № 40 проводит седьмую 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.

Традиционно для участия в городской олимпиаде приглашаются талантливые школьники из Нижегородской области (г. Балахна, г. Саров).

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

Седьмая городская олимпиада проводится на четырёх 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 образовательное учреждение лицей № 40 (компьютерный центр) — 22 места;
- Муниципальное образовательное учреждение школа № 30 — 10 мест.

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 (ННГУ, НОЦ ИПФ РАН, лицей № 40, школа № 30)
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 и социально-правовой защиты детства администрации города Нижнего Новгорода;
Лелюх В. Д., старший преподаватель ННГУ им. Н. И. Лобачевского;
Цветков М. И., начальник отдела общего среднего образования департамента образования и социально-правовой защиты детства администрации города Нижнего Новгорода;
Бовкун И. Л., главный специалист отдела общего среднего образования департамента образования и социально-правовой защиты детства 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 образовательного учреждения лицей № 40;
Антонова Л. Л., директор муниципального образовательного учреждения школа № 30;
Евстратова Л. П., заместитель директора по информатизации МОУ лицей № 40;
Гашпар И. Л., куратор НОЦ ИПФ РАН;
Братчикова Т. А., учитель МОУ лицей № 40;
Герасимова И. Н., учитель МОУ школа № 30;
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 5 курса ВШОПФ ННГУ;
Вадимов В. Л., студент 2 курса ВШОПФ ННГУ;
Епифанов В. Ю., студент 2 курса мехмата ННГУ;
Калинин П. А., аспирант ИПФ РАН;
Круглов А. А., младший научный сотрудник ИПФ РАН;
Матросов М. В., студент 5 курса ВМК ННГУ;
Низовцев С. Н., студент 2 курса ВМК МГУ;
Разенштейн И. П., студент 4 курса мехмата МГУ;
Тимушев Р. И., старший разработчик Luxoft/UBS;
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
```

VII Городская олимпиада школьников по информатике 3 февраля 2011 г.

Задача 1. Тупики в городе

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

Компания, в которой все ещё работает ваш друг, решила выпустить новую игру для мобильных телефонов, чтобы пассажирам было не так скучно стоять в пробках. Зная вас как хорошего программиста, вам поручили написать основную часть этой игры.

Игра будет состоять в том, что игрок будет управлять автобусом, перемещающимся по городу. В первой версии игры город будет представлять собой прямоугольное поле размера $N \times M$ клеток, каждая из которых либо занята зданием, либо свободна (т. е. по ней проходит дорога). Автобус игрока может перемещаться лишь по дорогам, но не по зданиям.

Кроме того, автобус считается достаточно большим, настолько, что он не может разворачиваться в пределах одной клетки. Правда, вам пока не хочется учитывать конкретные размеры автобуса, поэтому для простоты игроку будет запрещено делать два хода подряд в противоположных направлениях, а любые другие маневры будут разрешены.

Таким образом, каждым очередным ходом игрок может переместить автобус на любую соседнюю по стороне свободную клетку, кроме той, с которой автобус только что приехал. (Первым ходом можно переместить автобус в любую сторону.)

В результате понятно, что автобус игрока может застрять в тупике, откуда ему будет некуда двигаться. Более того, ясно, что есть клетки, куда заезжать нельзя, т. к., заехав туда, в итоге игрок будет вынужден доехать до тупика.

Строго говоря, пусть игрок перемещает автобус бесконечно долго (т. е. в течение бесконечного количества ходов). Тогда несложно видеть, что в некоторых свободных клетках игрок может бывать бесконечно много раз (при условии, что начальная клетка выбрана удачно), а в некоторых — не более одного (и то лишь в начальной части игры).

Сейчас вы хотите написать программу, которая разделит все свободные клетки поля на эти два типа.

Формат входных данных

На первой строке входного файла находятся два числа N и M — количество строк и столбцов игрового поля соответственно ($1 \leqslant N, M \leqslant 500$). Далее следуют N строк, описывающих поле. В каждой строке находятся ровно M символов, каждый из которых может быть или '`#`' (решётка, ASCII-код 35), или '`.`' (точка). Решётка обозначает клетку со зданием, точка — свободную клетку.

Формат выходных данных

В выходной файл выведите N строк по M символов в каждой: для клеток, в которых находятся здания, выводите '`#`', для клеток, где игрок может побывать не более одного раза — '`X`' (латинская заглавная буква X), для остальных клеток — '`.`'.

Пример

Входной файл	*Выходной файл*
4 12 .#....#.##.. .##.#.#.##.. ......####.. .##.#.#.####	X#X...#X##.. X##.#.#X##.. XXX...####.. X##X#X#X####
3 2 ## ## ##	## ## ##

Система оценки

Если ваша программа будет работать при $N, M \leqslant 100$, то вы получите как минимум 50 баллов. Если ваша программа будет работать при $N, M \leqslant 250$, то вы получите как минимум 60 баллов.

Обратите также внимание, что для получения полного балла вам, скорее всего, придётся использовать 32-битный компилятор (Free Pascal, GNU C, GNU C++).

Решение

Будем называть *степенью* свободной клетки количество соседних с ней свободных клеток, которые ещё не были помечены как тупики. Тогда достаточно очевидно, что найти все тупики можно следующим простым алгоритмом. Пометим все свободные клетки со степенью 0 или 1 как тупики. После этого степени некоторых ещё не помеченных вершин могут стать равными 0 или 1 — отметим их тоже как тупики. И так далее, пока не останется не помеченных свободных вершин степени 0 или 1.

Строгое доказательство этого алгоритма мы здесь приводить не будем.

Элементарная реализация этого алгоритма не пройдёт по времени, и для получения полного балла необходимо было применить следующие идеи. Во-первых, заранее просчитаем начальную степень каждой вершины. Далее, организуем очередь из «обнаруженных» вершин, требующих пометки. Изначально в эту очередь поместим все вершины, у которых с самого начала степень равна 0 или 1. Будем обрабатывать вершины в этой очереди в порядке их добавления. При обработке очередной вершины пометим её как тупик, просмотрим все её соседние клетки, для каждой из них уменьшим её степень на единицу и, если степень стала 0 или 1, то добавим её в очередь. В такой реализации каждая клетка добавляется в очередь не более одного раза и обрабатывается не более одного раза, поэтому этот алгоритм работает за $O(MN)$ и набирает полный балл.

Отметим, что существуют и другие способы решения этой задачи. Например, можно было адаптировать известный алгоритм поиска в глубину; такое решение можно скачать в архиве решений жюри (`deadends_pk_correct_dfs.cpp`).

Пример правильной программы

```
const maxn=500;
      di:array[1..4] of integer=(-1,1,0,0);
      dj:array[1..4] of integer=(0,0,-1,1);
var f:text;
    a:array[-1..maxn+2,-1..maxn+2] of char;
    d:array[-1..maxn+2,-1..maxn+2] of byte;
    q:array[1..maxn*maxn] of record
       i,j:integer;
      end;
    n,m:integer;
    i,j:integer;
    l,r:integer;

procedure addp(i,j:integer);
begin
  inc(r);
  q[r].i:=i;
  q[r].j:=j;
end;

procedure decd(i,j:integer);
var ii,jj:integer;
    k:integer;
begin
  for k:=1 to 4 do begin
    ii:=i+di[k];
    jj:=j+dj[k];
    dec(d[ii,jj]);
    if (a[ii,jj]<>'#')and(d[ii,jj]=1) then
      addp(ii,jj);
  end;
end;
begin
  assign(f,'deadends.in');reset(f);
  read(f,n,m);
  readln(f);
  fillchar(a,sizeof(a),'#');
  for i:=1 to n do begin
    for j:=1 to m do
      read(f,a[i,j]);
    readln(f);
  end;
  close(f);
  fillchar(d,sizeof(d),4);
  l:=1;
  r:=0;
  for i:=0 to n+1 do
    for j:=0 to m+1 do
      if a[i,j]='#' then
        decd(i,j);
  while l<=r do begin
    i:=q[l].i;
    j:=q[l].j;
    inc(l);
    a[i,j]:='X';
    decd(i,j);
  end;
  assign(f,'deadends.out');rewrite(f);
  for i:=1 to n do begin
    for j:=1 to m do
      write(f,a[i,j]);
    writeln(f);
  end;
  close(f);
end.
```

Задача 2. Калитка в заборе

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

Дядя Фёдор, кот Матроскин и Шарик решили обновить забор вокруг своего сада в Простоквашино. Матроскин и Шарик, недолго думая, вкопали N столбов вдоль одной из сторон участка. Это очень сильно расстроило Дядю Фёдора, так как его друзья забыли о самом главном — калитка должна находиться именно на этой стороне, и для неё необходимо было оставить проём шириной как минимум W. Теперь им придётся выкапывать некоторые столбы.

Чтобы работа не пропадала даром, выкопать надо как можно меньше столбов. Помогите Дяде Фёдору определить, какие именно столбы надо выкопать. После выкапывания столбов должен найтись промежуток (между двумя оставшимися столбами, или между оставшимся столбом и концом стороны участка, или между двумя концами стороны участка) ширины больше или равной W.

Формат входных данных

Первая строка содержит два целых числа N и W — количество вкопанных столбов и минимально необходимую ширину проёма для калитки соответственно. Гарантируется, что $0 \leqslant N \leqslant 30\,000$ и что $0 \leqslant W \leqslant 60\,000$.

Будем считать, что вдоль интересующей нас стороны участка введена ось координат. Во второй строке входного файла находятся два числа L и R — координаты левого и правого конца этой стороны ($L < R$). Далее следуют N чисел — координаты вкопанных столбов. Все координаты (включая L и R) — различные целые числа, по модулю не превосходящие 30 000. Гарантируется, что все столбы вкопаны между левым и правым концами стороны.

Формат выходных данных

В первой строке выходного файла должно быть минимальное число столбов, которые надо выкопать. Далее должны следовать номера этих столбов. Столбы нумеруются в том порядке, как они указаны во входном файле, начиная с 1.

Если решений несколько, то вы можете вывести любое. Если решения нет, то выведите в выходной файл одну строку, содержащую число `-1`.

Пример

Входной файл	*Выходной файл*
3 2 2 6 3 4 5	1 2
3 2 1 6 4 3 5	0
3 5 1 7 5 3 4	3 2 1 3

Решение

Добавим два «виртуальных» столба — с координатами L и R. Поскольку выкапывать эти столбы невыгодно (их выкапывание не увеличивает ширину максимального промежутка между столбами), то ответ к задаче не изменится.

Далее для каждого столба сохраним его порядковый номер во входных данных (для «виртуальных» столбов этот номер можно задать произвольно). Отсортируем массив столбов в порядке возрастания координат, например, алгоритмом быстрой сортировки. Обозначим через $x[i]$ координату столба на позиции i отсортированного массива. Теперь задача свелась к следующей: найти пару индексов l и r ($l < r$) такую, что

$$x[r] - x[l] \geqslant W \qquad (*)$$

и разница $r - l$ минимальна (количество выкопанных столбов будет равно $r - l - 1$, поэтому достаточно минимизировать $r - l$). Для решения этой задачи для каждого l найдём наименьшее значение r, при котором $(*)$ выполнено, и выберем пару (l, r) с наименьшей разностью. Это можно сделать двумя способами:

1. Будем перебирать индексы l в порядке возрастания и хранить текущий индекс r — наименьший номер столба такой, что выполняется неравенство $(*)$. Изначально положим $l = 1$ и $r = 1$ (если столбы нумеруются с 1). Далее для каждого l (включая 1) будем увеличивать значение r, пока $(*)$ не выполнится. Если при этом в какой-то момент значение r стало равно количеству столбов, то завершаем работу алгоритма. При нахождении подходящего значения r проверяем разность $r - l$ и увеличиваем l на 1.

   Поскольку при увеличении l соответствующее значение r может только увеличиваться, то алгоритм корректен и общее суммарное количество изменений индексов l и r будет не более, чем $2(N + 2)$.

2. Для каждого индекса l будем искать бинарным поиском наименьшее значение r, удовлетворяющее $(*)$. Если такое r найдено, то проверяем разность $r - l$ на предмет оптимальности. Общая вычислительная сложность этого способа есть $O(N \log(N))$.

После того, как оптимальная пара индексов l и r найдена, для установки калитки необходимо выкопать столбы на позициях $l + 1$, $l + 2$, ..., $r - 1$ в отсортированном массиве — то есть всего $(r - l - 1)$ столб.

Если ни для какого l подходящее значение r не найдено, то следует вывести `-1`.

Пример правильной программы

```
var a : array[0..30001] of integer;
    b : array[0..30001] of integer;
    i, r, n : integer;
    w, tmp : longint;
    ans, ansl, ansr : integer;

procedure qsort(l,r:integer);
var i, j, m, tmp : integer;
begin
  i:=l;
  j:=r;
  m:=a[(l+r) div 2];
  repeat
    while a[i]<m do inc(i);
    while a[j]>m do dec(j);
    if i<=j then begin
      tmp:=a[i]; a[i]:=a[j]; a[j]:=tmp;
      tmp:=b[i]; b[i]:=b[j]; b[j]:=tmp;
      dec(j);
      inc(i);
    end;
  until i>j ;
  if i<r then qsort(i,r);
  if l<j then qsort(l,j);
end;

begin
  assign(input,'door.in');reset(input);
  assign(output,'door.out');rewrite(output);
  read(n,w);
  read(a[0],a[n+1]);
  tmp:=a[0];
  if a[n+1]-tmp>=w then begin
    for i:=1 to n do begin
      read(a[i]);
      b[i]:=i;
    end;
    qsort(1,n);
    r:=0;
    ans:=n; ansl:=0; ansr:=n+1;
    for i:=0 to n do begin
      tmp:=a[i];
      if r<=i then r:=i+1;
      while (r<=n) and(a[r]-tmp<w) do
        inc(r);
      if (a[r]-tmp>=w) and(r-i-1<ans) then begin
        ans:=r-i-1;
        ansl:=i;
        ansr:=r;
      end;
    end;
    writeln(ans);
    for i:=ansl+1 to ansr-1 do writeln(b[i]);
  end
  else writeln(-1);
  close(input);
  close(output);
end.
```

Задача 3. Множества, свободные от сумм

Входной файл	`mss.in`
Выходной файл	`mss.out`
Ограничение по времени	2 с
Максимальный балл за задачу	100

Пусть нам дано натуральное число N. Рассмотрим множество различных целых чисел $\{a_1, a_2, \ldots, a_k\}$, где каждое число лежит в интервале от 0 до $N-1$ включительно. Назовём такое множество *свободным от сумм*, если в этом множестве не найдётся таких трёх чисел, что сумма двух из них сравнима с третьим по модулю N. Строго говоря, назовём множество свободным от сумм, если для каждой тройки (не обязательно различных) индексов x, y и z ($1 \leqslant x, y, z \leqslant k$) выполняется неравенство $(a_x + a_y) \bmod N \neq a_z$, где $p \bmod q$ — остаток от деления p на q.

Например, при $N = 6$ множествами, свободными от сумм, *не* являются, например, $\{0\}$ (т. к. $(0+0) \bmod 6 = 0$), $\{1, 2\}$ (т. к. $(1+1) \bmod 6 = 2$), $\{3, 4, 5\}$ (т. к. $(4+5) \bmod 6 = 3$), но множество $\{1, 3, 5\}$ является свободным от сумм.

По заданному N определите, сколько существует множеств, свободных от сумм.

Формат входных данных

Во входном файле находится одно целое число N. Гарантируется, что $1 \leqslant N \leqslant 35$.

Формат выходных данных

В выходной файл выведите одно число — ответ на задачу.

Пример

Входной файл	*Выходной файл*
2	2
6	14

Примечание: Все множества, свободные от сумм, для $N = 6$ — это следующие: $\{5\}$, $\{4\}$, $\{3\}$, $\{3, 5\}$, $\{3, 4\}$, $\{2\}$, $\{2, 5\}$, $\{2, 3\}$, $\{1\}$, $\{1, 5\}$, $\{1, 4\}$, $\{1, 3\}$, $\{1, 3, 5\}$, $\{\}$ (последнее множество — пустое, т.е. не содержащее ни одного элемента, с $k = 0$ — тоже считается свободным от сумм).

Решение

Задача решалась аккуратно реализованным перебором с возвратом.

Будем перебирать все множества, свободные от сумм. Для всех чисел от 1 до $N-1$ будем рассматривать два варианта: либо это число входит в текущее множество, либо нет (очевидно, что ноль никогда не может входить в искомое множество). Напишем процедуру $find(i)$, которая будет делать следующее. Считая, что для всех чисел от 1 до $i-1$ уже определено, входят они в текущее множество или нет, процедура переберёт два варианта для числа i (включать его в текущее множество или нет) и для каждого запустит рекурсивно $find(i+1)$. Естественно, рекурсивный запуск следует производить, только если текущее множество все ещё является свободным от сумм. Для ускорения работы следует по ходу рекурсии особо отмечать те числа, которые уже нельзя добавлять в текущее множество, а именно, числа, равные сумме или разности двух уже имеющихся в множестве чисел. Эту информацию легко обновлять при добавлении нового числа в множество.

Программа, аккуратно реализующая изложенный алгоритм, укладывается в ограничения по времени; из деталей аккуратной реализации отметим только то, что при добавлении нового числа i следует особо проверить, не имеется ли уже в множестве число $2i \bmod N$, а также то, что стандартные функции взятия остатка некорректно работают с отрицательными числами, поэтому перед взятием остатка разности двух чисел необходимо к разности прибавлять N.

Пример правильной программы

```
const maxN=35;
type tarr=array[0..maxN] of integer;
var n:integer;
    can:tarr;
    ans:longint;
    f:text;

procedure check;
begin
  inc(ans);
end;

procedure find(i:integer);
var old:tarr;
    j:integer;
begin
  if i>=n then begin
    check;
    exit;
  end;
  if can[i]=-1 then begin
    find(i+1);
    exit;
  end;
  can[i]:=0;
  find(i+1);
  if can[(i+i) mod n]=1 then
    exit;
  old:=can;
  can[i]:=1;
  for j:=1 to i do
    if can[j]=1 then begin
      can[(j+i) mod n]:=-1;
      can[(j-i+n) mod n]:=-1;
      can[(i-j+n) mod n]:=-1;
    end;
  find(i+1);
  can:=old;
end;

begin
  assign(f,'mss.in');reset(f);
  read(f,n);
  close(f);
  fillchar(can,sizeof(can),0);
  ans:=0;
  find(1);
  assign(f,'mss.out');rewrite(f);
  writeln(f,ans);
  close(f);
end.
```

Задача 4. Переливание

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

На досуге вы любите почитать сборники занимательных задач по математике. Недавно вы наткнулись в одном из таких сборников на следующую задачу:

> Есть бесконечный резервуар с водой и два пустых сосуда объёмом 5 и 12 литров. Можно: наливать воду из резервуара в любой сосуд до его заполнения, переливать воду из одного сосуда в другой до заполнения второго или опустошения первого (смотря что будет раньше) и выливать воду из сосуда на землю до полного опустошения сосуда. Как таким образом можно отмерить 3 литра?

Вы решили написать программу, которая будет решать подобные задачи для произвольных объёмов сосудов.

Формат входных данных

Во входном файле находятся три целых числа — V_1, V_2 и V — объёмы двух сосудов и объем воды, который нужно отмерить. Гарантируется, что $1 \leqslant V_1, V_2 \leqslant 32767$ и $0 \leqslant V \leqslant \max(V_1, V_2)$.

Формат выходных данных

В первую строку выходного файла выведите одно число — количество действий в вашем решении. Далее выведите соответствующее количество строк, описывающих действия в вашем решении. Для каждого действия выведите два числа:

- если это действие — переливание из одного сосуда в другой, то первое число должно быть номером сосуда, откуда надо переливать воду, а второе — номером сосуда, куда переливать;
- если это действие — набор воды из резервуара, то первое число должно быть нулём, а второе — номером сосуда, куда наливать;
- если это действие — выливание воды «на землю», то первое число должно быть номером сосуда, а второе — нулём.

После выполнения всех операций хотя бы в одном сосуде должна находиться вода в объёме V.

Если существует несколько решений, то вы можете вывести любое. Ваше решение не обязано быть оптимальным, единственное ограничение — размер выходного файла не должен превосходить 3 Мб.

Если решений не существует, выведите одно число `-1`.

Пример

Входной файл	*Выходной файл*

`5 12 3`	`10`
	`0 1`
	`1 2`
	`2 1`
	`1 2`
	`0 1`
	`1 0`
	`0 1`
	`1 2`
	`0 1`
	`1 2`

Решение

Эта задача имеет довольно простое решение, однако доказательство корректности решения несколько более сложное. Будем считать, что $V_1 \leqslant V_2$, в противном случае переименуем сосуды (только не забудем это учесть при выводе решения в выходной файл). В этом случае задача решается следующим алгоритмом. Сначала особо рассмотрим случай, когда требуемый объём строго равен V_2: здесь решение очевидно. В противном случае наливаем доверху первый сосуд и полностью выливаем его во второй, опять наливаем доверху первый сосуд и полностью выливаем его

во второй и т. д. В очередной момент мы не сможем вылить *всё* содержимое первого сосуда во второй — в таком случае переливаем из первого во второй сколько влезет, выливаем содержимое второго сосуда «на землю», и продолжаем переливание из первого во второй.

Строго говоря, получаем следующий алгоритм:

1. если требуется набрать V_2, то вывести очевидное решение; иначе:
2. налить первый сосуд доверху;
3. перелить из первого сосуда во второй сколько влезет;
4. если второй сосуд полон, то вылить его «на землю» и перелить остатки из первого сосуда во второй (они точно влезут, т. к. $V_1 \leqslant V_2$);
5. вернуться к п. 2.

Оказывается, что, во-первых, в некоторый момент после окончания п. 4 *оба* сосуда будут пусты, т. е. алгоритм зациклится, и во-вторых, если задача разрешима и $V \neq V_2$, то в некоторый момент по окончании п. 4 во втором сосуде будет требуемый объём. Таким образом, для решения задачи достаточно было просто запрограммировать этот алгоритм. Если в некоторый момент по окончании п. 4 во втором сосуде получился требуемый объём, то мы нашли решение; если же за одно повторение цикла решения не нашлось, то выводим -1.

Докажем оба утверждения. Во-первых, заметим, что после любой допустимой последовательности операций объём, находящийся в каждом из сосудов, будет делиться на НОД(V_1, V_2). Действительно, изначально это утверждение выполняется (в обоих сосудах нулевые объёмы), и несложно проверить, что, если перед некоторой операцией это утверждение выполнялось, то оно будет выполняться и после неё. Обозначим $V_0 = \text{НОД}(V_1, V_2)$. Таким образом, если требуемый объём V не делится на V_0, то решения точно не существует.

По условию $0 \leqslant V \leqslant V_2$; случай $V = V_2$ рассмотрен отдельно — осталось рассмотреть случаи $0 \leqslant V < V_2$. Заметим, что среди таких ровно V_2/V_0 разрешимых (а именно, 0, V_0, $2V_0$, $\ldots$, $V_2 - V_0$). Кроме того, несложно заметить, что после k-ого выполнения п. 4 во втором сосуде объём будет ровно $X_k = kV_1 \bmod V_2$ (действительно, во второй сосуд мы перелили в общей сложности ровно kV_1 жидкости и вылили несколько раз по V_2 так, что осталось меньше V_2). Найдём длину периода последовательности X_k. Если $k'V_1 \bmod V_2 = k''V_1 \bmod V_2$ и $k' > k''$, то $(k' - k'')V_1 \bmod \bmod V_2 = X_{k'-k''} = 0$, т. е., во-первых, через $k' - k''$ итераций алгоритма после п. 4 второй сосуд станет пустым. Во-вторых, это обозначает, что $(k' - k'')V_1 = lV_2$, где l целое. Разделив на V_0, получим, что $(k' - k'')(V_1/V_0) = l(V_2/V_0)$, но V_1/V_0 и V_2/V_0 взаимно просты, значит, $(k' - k'')$ делится на V_2/V_0. С другой стороны, несложно проверить, что при $k' - k'' = V_2/V_0$ равенство выполнится. Следовательно, длина периода последовательности объёмов равна V_2/V_0, а поскольку в ней может быть только V_2/V_0 различных чисел, то все они действительно появляются. Следовательно, все разрешимые объёмы появятся в некоторый момент во втором сосуде после выполнения п. 4. Корректность алгоритма доказана.

На самом деле, известно, что диофантово уравнение $kV_1 - lV_2 = V$, равносильное (при условии $0 \leqslant V < V_2$) уравнению $kV_1 \bmod V_2 = V$, разрешимо в целых числах тогда и только тогда, когда V делится на НОД(V_1, V_2), причём соответствующие k и l легко ищутся с помощью расширенного алгоритма Евклида без необходимости полного перебора всех k.

Но, поскольку в задаче всё равно требуется вывести способ получения искомого объёма, то приведённый выше алгоритм проходит по времени.

Пример правильной программы

```
const id:array[0..2] of char=('0','1','2');
var f:text;
    t,v1,v2,v:integer;
    c:longint;
    nn:longint;

procedure out(i,j:integer);
begin
  writeln(f,id[i],' ',id[j]);
end;

begin
  assign(f,'flow.in');reset(f);
  read(f,v1,v2,v);
  close(f);
  if v1>v2 then begin
    t:=v1;v1:=v2;v2:=t;
    id[1]:='2';
    id[2]:='1';
  end;
  assign(f,'flow.out');rewrite(f);
  if v=v2 then begin
    writeln(f,1);
    out(0,2);
    close(f);
    halt;
  end;
  c:=0;
  nn:=0;
  while c<>v do begin
    c:=c+v1;
    inc(nn,2);
    if c>=v2 then begin
      c:=c-v2;
      inc(nn,2);
    end;
    if c=0 then begin
      writeln(f,-1);
      close(f);
      halt;
    end;
  end;
  writeln(f,nn);
  c:=0;
  while c<>v do begin
    c:=c+v1;
    out(0,1);
    out(1,2);
    if c>v2 then begin
      c:=c-v2;
      out(2,0);
      out(1,2);
    end;
  end;
  close(f);
end.
```

Задача 5. Пересечение

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

Компания, производящая оборудование для сотовой связи, обратилась к вам с просьбой написать программу, оценивающую качество организации сети. Одним из важных параметров является то, пересекаются ли зоны покрытия передатчиков, работающих на одинаковых частотах. Для простоты будем считать, что область покрытия каждого передатчика представляет собой многоугольник на плоскости (не обязательно выпуклый). Две области покрытия будем считать пересекающимися, если у них есть хотя бы одна общая точка (возможно, лежащая на границе одной или даже обеих областей). Ваша программа должна принимать на вход набор пар многоугольников, описывающих зоны покрытия передатчиков, и выводить про каждую пару информацию о том, пересекаются ли эти зоны.

Формат входных данных

На первой строке входного файла находится число K — количество тестов во входном файле. Далее идёт описание K тестов. Каждый тест задаётся описанием двух многоугольников, которые надо проверить на пересечение. Каждый многоугольник задаётся в следующем формате: сначала указывается одно число N_i —

число вершин этого многоугольника, — после чего идут N_i строк, каждая из которых содержит два разделённых пробелом числа x_{ij} и y_{ij} — координаты j-й вершины этого многоугольника. Вершины перечислены в порядке обхода многоугольника.

Число пар многоугольников в одном тесте $1 \leqslant K \leqslant 10$, число вершин каждого многоугольника $3 \leqslant N_i \leqslant 100$, координаты вершин — целые числа, $|x_{ij}|, |y_{ij}| \leqslant 10\,000$.

Формат выходных данных

Для каждой пары многоугольников выведите в выходной файл на отдельной строке одно слово: 'YES', если многоугольники пересекаются, и 'NO', если нет.

Пример

Входной файл	*Выходной файл*
2 3 0 0 -1 0 0 -1 3 1 1 2 1 1 2 4 0 0 2 0 2 2 0 2 4 1 1 3 1 3 3 1 3	NO YES

Решение

Поскольку каждый входной файл по сути представляет собой набор их K независимых подтестов, мы будем рассматривать задачу о пересечении двух многоугольников, не обращая внимания на остальные пары.

Несложно увидеть, что если два многоугольника пересекаются, то возможны всего два существенно различных варианта: либо один из многоугольников полностью лежит внутри другого, либо у них есть общая точка, лежащая на границе обоих многоугольников.

Для начала разберёмся с первой возможностью. Возьмём по одной вершине каждого многоугольника и проверим её на принадлежность второму. Если такая точка найдена, то ответ YES и дальнейшие проверки не требуются. Существует несколько алгоритмов проверки принадлежности точки многоугольнику, в частности, в приведённом ниже решении проверяется чётность количества пересечений исходящего

из точки горизонтального луча со сторонами многоугольника. При реализации этого алгоритма следует обратить внимание на то, чтобы не посчитать проход луча через вершину многоугольника за два пересечения.

Если первая проверка не увенчалась успехом, то попытаемся найти общую точку границ многоугольников. Для этого достаточно попробовать найти точку пересечения каждой стороны первого многоугольника с каждой стороной второго. Если такая точка найдена, то ответ на задачу `YES`, в противном случае ответ `NO`.

Получившееся решение имеет сложность $O(KN^2)$ и без проблем укладывается в ограничение по времени. Следует так же отметить, что обе вышеописанные проверки не требуют деления, и все вычисления могут проводиться в рамках целочисленной арифметики достаточной размерности (для ограничений из условия достаточно 64-битного типа данных).

Пример правильной программы

```
const MAXN = 100;
var ax, ay, bx, by : array[1..MAXN+1] of int64;
    k, s, an, bn, i : integer;

function inside(x,y:int64;
  vx,vy:array of int64; vn:integer) : boolean;
var c : boolean;
    i : integer;
begin
  c:=false;
  for i:=1 to vn do begin
    if (vy[i]<y) and (y<=vy[i+1]) and
         ( (vx[i+1]-vx[i])*(y-vy[i])>
           (x-vx[i])*(vy[i+1]-vy[i]) )
       then c:=not c;
    if (vy[i+1]<y) and (y<=vy[i]) and
        ( (vx[i+1]-vx[i])*(y-vy[i])<
          (x-vx[i])*(vy[i+1]-vy[i]) )
       then c:=not c;
  end;
  inside:=c;
end;

function max(x,y:int64) : int64;
begin
  if x>y then max:=x
  else max:=y;
end;

function min(x,y:int64) : int64;
begin
  if x<y then min:=x
  else min:=y;
end;

function intersect(i,j:integer) : boolean;
var p, q : int64;
begin
  intersect:=false;
  if (max(ax[i],ax[i+1])<min(bx[j],bx[j+1])) or
     (max(ay[i],ay[i+1])<min(by[j],by[j+1])) or
     (min(ax[i],ax[i+1])>max(bx[j],bx[j+1])) or
     (min(ay[i],ay[i+1])>max(by[j],by[j+1]))
       then exit;
  p:=(ax[i+1]-ax[i])*(by[j]-ay[i])
       -(bx[j]-ax[i])*(ay[i+1]-ay[i]);
  q:=(ax[i+1]-ax[i])*(by[j+1]-ay[i])
       -(bx[j+1]-ax[i])*(ay[i+1]-ay[i]);
  if p*q>0 then exit;
  p:=(bx[j+1]-bx[j])*(ay[i]-by[j])
       -(ax[i]-bx[j])*(by[j+1]-by[j]);
  q:=(bx[j+1]-bx[j])*(ay[i+1]-by[j])
       -(ax[i+1]-bx[j])*(by[j+1]-by[j]);
  if p*q>0 then exit;
  intersect:=true;
end;

function solve(): boolean;
var i, j : integer;
begin
  solve:=true;
  if inside(ax[1],ay[1],bx,by,bn) then exit;
  if inside(bx[1],by[1],ax,ay,an) then exit;
  for i:=1 to an do
    for j:=1 to bn do
      if intersect(i,j) then exit;
  solve:=false;
end;

begin
  assign(input,'sect.in'); reset(input);
  assign(output,'sect.out'); rewrite(output);
  readln(k);
  for s:=1 to k do begin
    readln(an);
    for i:=1 to an do readln(ax[i],ay[i]);
    ax[an+1]:=ax[1]; ay[an+1]:=ay[1];
    readln(bn);
    for i:=1 to bn do readln(bx[i],by[i]);
    bx[an+1]:=bx[1]; by[an+1]:=by[1];
    if solve() then writeln('YES')
    else writeln('NO');
  end;
  close(output);
  close(input);
end.
```

Задача 6. Собрать треугольник

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

Вы по-прежнему работаете под руководством д. б. н., проф. О. Б. Ломова и изучаете интеллект обезьян. Ваши подопечные уже очень далеко ушли от столь элементарной задачи, как сбор квадрата. Теперь вы работаете над тем, чтобы обучить их намного более сложной задаче. Вы по-прежнему даёте обезьянам набор из N палочек, но на этот раз вы хотите, чтобы они собрали из этих палочек треугольник.

Конечно, решить эту задачу в элементарном варианте — выбрать три палочки и собрать из них треугольник — ваши подопечные могут без каких-либо проблем; вы же хотите их обучить, чтобы они собирали один большой треугольник из *всех* выданных им палочек сразу. Таким образом, они должны разбить палочки на три группы так, чтобы, сложив палочки каждой группы в один большой отрезок, получить три отрезка, из которых можно собрать треугольник. Полученный треугольник должен быть невырожденным, т. е. его площадь должна быть строго больше нуля.

Как и в прошлый раз, вам понадобилась программа, которая определит, разрешима ли задача для данного набора палочек.

Формат входных данных

На первой строке входного файла находится одно натуральное число N — количество палочек в наборе ($1 \leqslant N \leqslant 16\,000$). На второй строке находятся N натуральных чисел — длины палочек. Гарантируется, что суммарная длина палочек не превосходит 100 000 000.

Формат выходных данных

Если решения не существует, то в первую строку выходного файла выведите одно слово '`no`' (без кавычек). В противном случае в первую строку выведите одно слово '`yes`', а в следующие три строки выведите какой-нибудь способ собрать треугольник из данных палочек. Каждая из этих трёх строк должна описывать очередную сторону получающегося треугольника: в каждой строке сначала должно идти количество палочек, из которых состоит эта сторона, а потом длины этих палочек. Каждую палочку, конечно, можно использовать только один раз.

Если есть несколько способов собрать треугольник из данных палочек, выведите любой.

Пример

Входной файл	*Выходной файл*
5 1 2 3 4 5	yes 2 4 3 1 5 2 1 2
5 1 2 3 4 100	no

Решение

Для удобства рассуждений поделим длину каждого отрезка на суммарную длину всех отрезков, так что сумма новых длин будет равна единице; это, очевидно, не изменит ответа.

Чтобы из трёх отрезков можно было собрать невырожденный треугольник, необходимо и достаточно, чтобы выполнялись неравенства треугольника: каждая сторона должна быть строго больше суммы двух других. С учётом того, что сумма длин сторон у нас всегда будет равна единице, несложно показать, что неравенства треугольника равносильны требованию того, чтобы каждая сторона была строго короче $1/2$. Таким образом, нам необходимо разбить данные нам отрезки на три группы так, чтобы суммарная длина отрезков в каждой группе была бы строго меньше $1/2$.

Сначала отметим два случая, когда решения точно не существует. Один случай очевиден: если среди данного набора имеется отрезок длины $\geqslant 1/2$, то решения точно нет. Второй случай менее очевиден, но несложно видеть, что если нам даны четыре отрезка, каждый длиной ровно $1/4$, то решения тоже нет.

Во всех остальных случаях решение есть, что мы докажем, приведя алгоритм построения такого решения.

Алгоритм состоит в следующем. Отсортируем отрезки по уменьшению длины. Далее пройдём по этому массиву и наберём первую сторону: возьмём самый длинный отрезок и добавим его к текущей стороне. Возьмём следующий отрезок и, если его добавление к текущей стороне не сделает длину этой стороны $\geqslant 1/2$, то добавим его, иначе пропустим. Аналогично поступим с третьим отрезком и т. д., в итоге получим первую сторону. Пройдём по массиву ещё один раз и аналогичным образом наберём вторую сторону (при этом, естественно, будем пропускать отрезки, уже взятые на первую сторону). После этого все оставшиеся отрезки отнесём к третьей стороне.

Докажем корректность этого алгоритма. А именно, предположим, что не имеет места ни один из двух описанных выше случаев, когда задача не имеет решения, и докажем, что тогда в результате работы этого алгоритма все три стороны получатся короче $1/2$. Обозначим длины получившихся сторон: первую a, вторую b и третью c. По построению, $a < 1/2$ и $b < 1/2$, поэтому нам надо лишь доказать, что $c < 1/2$. Для этого докажем, что $a > 1/4$ и $b \geqslant 1/4$, откуда с учётом условия $a + b + c = 1$ будет следовать, что $c < 1/2$.

Докажем, что $a > 1/4$. Во-первых, a не может быть нулём, т. к. по крайней мере самый первый отрезок мы включим в a (все имеющиеся у нас отрезки строго короче $1/2$, иначе решения бы не существовало). Предположим, что $0 < a \leqslant 1/4$. Значит, в a мы включили какой-то отрезок, который не длиннее $1/4$. С другой стороны, очевидно, что хоть какой-то отрезок мы не включили в a. Каждый такой отрезок безусловно не короче $1/4$, т. к. его добавление к a делает a длиннее $1/2$ (а иначе мы этот отрезок включили бы). Но самый первый отрезок не длиннее $1/4$, т. к. мы его взяли в a, а остальные не длиннее его, т. к. отрезки упорядочены по длине. Следовательно, во-первых, самый первый отрезок строго равен $1/4$, во-вторых, только его мы и взяли в a, в-третьих, все отрезки, которые мы не взяли в a, тоже строго равны $1/4$. Значит, все вообще отрезки строго равны $1/4$, а это —

описанный выше случай, когда решения не существует. Мы же считаем, что этот случай не имеет места, противоречие, следовательно, $a > 1/4$.

Аналогично доказывается, что $b \geqslant 1/4$. Действительно, пусть $b < 1/4$, тогда самый длинный отрезок, оставшийся после a, короче $1/4$. Но безусловно есть отрезок, который мы не взяли ни в b, ни в a (т. к. $a < 1/2$ и $b < 1/2$, значит, $a + b < 1$), тогда этот отрезок тем более короче $1/4$ и мы должны были добавить его к b, противоречие. Отметим, что вопрос о том, может ли b строго равняться $1/4$, не столь тривиален (в отличие от первой стороны, при наборе второй некоторые отрезки уже использованы), но нам достаточно нестрогого неравенства $b \geqslant 1/4$.

Таким образом, мы доказали, что если не имеют места два приведённых выше случая, когда решения не существует, то наш алгоритм строит две стороны a и b такие, что $1/4 < a < 1/2$ и $1/4 \leqslant b < 1/2$, и, значит, третья сторона c строго короче $1/2$. Таким образом, этот алгоритм действительно строит решение.

Реализация алгоритма не составляет затруднений; заметим только, что при реализации удобно не выполнять явного деления отрезков на их суммарную длину (что создаст проблемы из-за необходимости работы с вещественными числами), а просто во всех проверяемых неравенствах и т. п. умножать правую сторону на суммарную длину отрезков, что и сделано в примере программы.

Возможны и другие способы решения задачи. Например, можно было действовать следующим образом: будем идти по массиву отрезков и набирать все три стороны одновременно: очередной отрезок будет присоединять к наиболее короткой на данный момент стороне. Если перебирать стороны в случайном порядке, то этот алгоритм в общем случае не работает (например, на тесте, состоящем из одного отрезка длины L и $L+1$ отрезка длины 1); но можно доказать, что если перебирать отрезки в порядке уменьшения их длины, то алгоритм будет корректным.

Отметим также ещё один возможный подход к решению. Можно было написать программу, которая действовала бы следующим образом: она пробовала бы составить треугольник случайным образом, т.е. для каждого отрезка случайным образом определяла бы, к какой стороне его отнести. Если получилось корректное решение, то программа выводит его в файл, иначе собирает треугольник ещё раз случайным образом и так далее, до тех пор, пока или не найдётся решение, или не кончится время. Этот алгоритм в большинстве случаев работает и потому набирает неплохой балл, но на некоторых тестах вероятность случайным образом собрать треугольник крайне мала (например, на приведённом выше тесте из одного длинного и многих коротких отрезков вероятность правильно собрать треугольник не больше $(2/3)^L$, что при больших L составляет крайне малую величину), поэтому этот алгоритм не получает полного балла.

Пример правильной программы

```
{$M 65520,0,1024}
var a:array[1..16000] of longint;
    n:integer;
    s:longint;
    f:text;
    i:integer;

procedure sortall;
var aa:array[1..16000] of longint;

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];
      inc(i1);
    end
    else begin
      aa[i]:=a[i2];
      inc(i2);
    end;
  for i:=l to r do
    a[i]:=aa[i];
end;

begin
  sort(1,n);
end;

procedure outno;
begin
  assign(f,'triangle.out');rewrite(f);
  writeln(f,'no');
  close(f);
  halt;
end;

procedure outline;
var i:integer;
    nn,ss:longint;
    ans:array[1..16000] of longint;
begin
  ss:=0;
  nn:=0;
  for i:=1 to n do
    if (a[i]>0)and((ss+a[i])*2<s) then begin
      ss:=ss+a[i];
      inc(nn);
      ans[nn]:=a[i];
      a[i]:=-1;
    end;
  write(f,nn,' ');
  for i:=1 to nn do
    write(f,ans[i],' ');
  writeln(f);
end;

begin
  assign(f,'triangle.in');reset(f);
  read(f,n);
  s:=0;
  for i:=1 to n do begin
    read(f,a[i]);
    s:=s+a[i];
  end;
  close(f);
  sortall;
  if n<3 then outno;
  if a[1]*2>=s then
    outno;
  if (a[1]*4=s)and(a[2]*4=s)
      and(a[3]*4=s)and(a[4]*4=s) then
    outno;
  assign(f,'triangle.out');rewrite(f);
  writeln(f,'yes');
  outline;
  outline;
  outline;
  close(f);
end.
```

Содержание