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

Одиннадцатая нижегородская городская олимпиада школьников по информатике имени В. Д. Лелюха

31 января 2015 г.

Нижний Новгород
2015

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.

Одиннадцатая городская олимпиада по информатике

31 января 2014 г. Городской ресурсный центр физико-математического образования в лице учредителей: Департамента образования администрации города Нижнего Новгорода, Нижегородского государственного университета им. Н.И. Лобачевского и МБОУ Лицей № 40 проводит одиннадцатую 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.

Спонсорами городской олимпиады школьников по информатике являются НПП «Прима», компания «Мера НН», ННГУ им. Н.И. Лобачевского.

Одиннадцатая городская олимпиада проводится в ННГУ им. Лобачевского на базе факультета ВМК (корпус 2).

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).

Для участия в городской олимпиаде приглашаются талантливые школьники из Нижегородской области.

Официальные разрешённые среды программирования — Free Pascal, GNU C/C++ (MinGW), Free Basic, Python 3.

Регламент проведения олимпиады

9:30—10:00	Регистрация участников
10:00—10:15	Открытие олимпиады, информация от жюри
10:15—15:15	Решение задач олимпиады
15:15—16:00	Обед
16:00—17:00	Работа жюри
17:00—18:00	Приветствие участников олимпиады, выступления учредителей, спонсоров. Подведение итогов олимпиады. Поздравление победителей и призёров.

Состав оргкомитета олимпиады

Сидоркина С. Л., первый заместитель директора департамента образования администрации города Нижнего Новгорода;
Цветков М. И., начальник отдела общего среднего образования департамента образования администрации города Нижнего Новгорода;
Тинькова Е. В., главный специалист отдела общего среднего образования департамента образования администрации города Нижнего Новгорода;
Авралев Н. В., проректор по связям с общественностью ННГУ им. Н.И. Лобачевского;
Борисов Н. А., доцент кафедры МО ЭВМ ННГУ им. Н.И. Лобачевского;
Сысоев А. В., ассистент кафедры МО ЭВМ ННГУ им. Н.И. Лобачевского;
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 классов НОЦ ИПФ РАН;

Состав предметной комиссии (жюри)

Председатель — Калинин П. А., старший инженер по разработке программного обеспечения, ЗАО «Интел А/О», к.ф.-м.н.;

Члены комиссии:
Евстратова Л. П., заместитель директора МОУ лицей № 40, ответственный секретарь комиссии;
Вадимов В. Л., студент 6 курса ВШОПФ ННГУ;
Демидов А. Н., инженер, НПП «ПРИМА»;
Калинин Н. А., студент 1 курса ВШОПФ ННГУ;
Лазарев Е. А., старший инженер по разработке программного обеспечения, ЗАО «Интел А/О», к.т.н.;
Лопаткин М. А., программист, Яндекс;
Матросов М. В., инженер-программист, НПП «АВИАКОМ»;
Шмелёв А. С., инженер-программист, НПП «ПРИМА».

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
```

Python 3

```
inf = open("example.in", "r")
a,b = map(int, inf.read().split())
ouf = open("example.out", "w")
ouf.write(str(a+b))
```

XI Городская олимпиада школьников по информатике 31 января 2015 г.

Задача 1. 23 февраля

Входной файл	`23feb.in`
Выходной файл	`23feb.out`
Ограничение по времени	0.5 секунды
Ограничение по памяти	256 мегабайт

Сегодня 23 Февраля, поэтому Малыш решил устроить парад своих солдатиков. Он уже достал их из коробки, расположил в одну линию и теперь хочет построить их по росту в порядке невозрастания слева направо.

В честь праздника Малыш решил, что будет менять местами тех и только тех солдатиков, между которыми стоят ровно 2 или ровно 3 других солдатика.

Малышу совершенно не обязательно построить солдатиков за минимально возможное количество перестановок, но ему обязательно нужно это сделать не более чем за 23 000 перестановок, иначе он не успеет до сна.

Помогите Малышу справиться с этой ответственной задачей.

Формат входных данных

В первой строке входного файла указано целое число N ($2 \leqslant N \leqslant 100$) — количество солдатиков у Малыша. Во второй строке через пробел указаны N положительных целых чисел, каждое из которых не превосходит 2000. K-ое число во второй строке задает рост K-ого солдатика в исходном построении. Солдатики нумеруются числами от 1 до N слева направо.

Формат выходных данных

Если расположить солдатиков требуемым образом невозможно, то в выходной файл требуется вывести единственное слово «`NO`» без кавычек.

Если построение осуществимо, то в первой строке выведите единственное слово «`YES`» без кавычек, а во второй строке выведите требуемое количество действий K ($0 \leqslant K \leqslant 23\,000$). Далее нужно вывести K строк, каждая из которых должна содержать два числа X и Y, означающих, что Малышу нужно поменять местами солдатиков, стоящих сейчас на X-ом и Y-ом местах.

Если есть несколько решений, выведите любое. Обратите внимание, что минимизировать число действий не требуется, главное — чтобы их было не больше 23 000.

Пример

Пример

Входной файл	*Выходной файл*
10 10 9 4 7 6 5 1 3 2 8	YES 2 10 7 3 7
2 1500 1700	NO

Примечание: Решения, работающие при $N \leqslant 8$, будут набирать 50 баллов.

Решение

Формализуем задачу. У нас есть массив из N чисел, нужно отсортировать его, переставляя элементы $a[i]$ с $a[i{+}3]$ или $a[i]$ с $a[i{+}4]$. Далее для удобства перестановку элементов массива $a[x]$ и $a[y]$ будем обозначать как (x, y).

Если бы не было ограничений на возможные перестановки, то задача просто бы решалась любой сортировкой, например, сортировкой «пузырьком», в которой используются только перестановки соседних элементов $(i, i+1)$. Если немного поэкспериментировать с ручкой и бумажкой, то можно заметить, что перестановка $(i, i+1)$ представима как последовательно выполненные перестановки $(i, i+4)$, $(i+1, i+4)$, $(i, i+4)$. Поэтому, чтобы отсортировать массив, будем сортировать его «пузырьком», но каждый раз, когда алгоритму потребуется поменять местами два соседних элемента, будем менять их чуть более сложно за несколько разрешенных перестановок. Обычной сортировке «пузырьком» для N элементов требуется не более, чем $N(N-1)/2$ перестановок. Нашей модифицированной сортировке нужно не более, чем $3N(N-1)/2$, что меньше 23 000 для $N \leqslant 100$.

Это казалось бы верное решение, но есть несколько тонких мест. Во-первых, если нам во время сортировки понадобится поменять местами элементы, например, $(N-1, N)$, то описанным выше способом этого сделать нельзя, иначе мы выйдем за границы массива. Поэтому около правого края массива соседние элементы $(i-1, i)$ нужно менять другим набором разрешенных перестановок: $(i-4, i)$, $(i-4, i-1)$, $(i-4, i)$.

Во-вторых, что более серьезно, для $N \leqslant 6$ у нас вообще описанным способом решить не получится, поэтому эти случаи нужно разбирать отдельно. Тут в силу маленького N есть множество способов сделать это. Опишем некоторые из них.

Можно представить каждую перестановку из N элементов как вершину графа. Для $N \leqslant 6$ в графе будет не более $6! = 720$ вершин. В этом графе ребрами соединим те и только те вершины, если соответствующие им перестановки переходят друг в друга за одно разрешенное действие $(i, i+3)$ или $(i, i+4)$. Тогда исходная задача сведется к поиску пути из одной вершины графа к другой, а это можно легко сделать поиском в глубину или ширину.

Можно в случайном порядке пытаться случайно поменять допустимым образом элементы массива. Если в ходе этих случайных допустимых перестановок решение находится, то выводить его, а если за 23 000 перестановок решение не нашлось, то выводить «NO». До этого можно интуитивно догадаться, но доказательство того, что при данных ограничениях это действительно решение, к сожалению, далеко не элементарно.

Пример правильной программы

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <algorithm>
#include <iostream>
#include <vector>

using namespace std;

typedef vector<int>    vi;
typedef pair<int, int> pii;

vi              a, b, sx, sy, sz;
int             n, q;
vector< pii >   ans;
vector< vi  >   o;
```

```
void save_swap(int x, int y) {
    swap(a[x], a[y]);
    ans.push_back( make_pair(x + 1, y + 1) );
}

void mil_swap(int x) {
    if (x >= 3) {
        save_swap(x - 3, x);
        save_swap(x - 3, x + 1);
        save_swap(x - 3, x);
    } else {
        save_swap(x + 4, x);
        save_swap(x + 1, x + 4);
        save_swap(x + 4, x);
    }
}

void add(vi &b, int x, int y, int z) {
    for(int i=0; i<o.size(); ++i)
        if (o[i] == b) return;
    o.push_back(b);
    sx.push_back(x);
    sy.push_back(y);
    sz.push_back(z);
}

void make_ans(int step) {
    if (step == 0) return;
    make_ans(sz[step]);
    ans.push_back(
        make_pair(sx[step] + 1, sy[step] + 1)
    );
    swap(a[sx[step]], a[sy[step]]);
}

int main() {
    freopen("23feb.in", "r", stdin);
    freopen("23feb.out", "w", stdout);
    scanf("%i", &n);
    for(int i=0; i<n; ++i) {
        scanf("%i", &q);
        a.push_back(q);
    }

    if (n > 6) {
        for(int i=0; i<n; ++i)
            for(int j=n-1; j>i; --j)
                if (a[j - 1] < a[j])
                    mil_swap(j - 1);
    } else {
        o.push_back(a);
        sx.push_back(0);
        sy.push_back(0);
        sz.push_back(0);
        int tail = 0;
        while (tail < o.size()) {
            b = o[tail];
            bool ok = true;
            for(int i=1; i<n; ++i)
                if (b[i - 1] < b[i])
                    ok = false;
            if (ok) {
                make_ans(tail);
                break;
            }

            for(int i=0; i + 4 < n; ++i) {
                swap(b[i], b[i + 4]);
                add(b, i, i + 4, tail);
                swap(b[i], b[i + 4]);
            }

            for(int i=0; i + 3 < n; ++i) {
                swap(b[i], b[i + 3]);
                add(b, i, i + 3, tail);
                swap(b[i], b[i + 3]);
            }
            tail++;
        }
    }

    bool ok = true;
    for(int i=1; i<n; ++i)
        ok = ok && (a[i] <= a[i - 1]);

    if (ok) {
        printf("YES\n%i\n", ans.size());
        for(int i=0; i<ans.size(); ++i)
            printf("%i %i\n", ans[i].first,
                              ans[i].second);
    } else {
        printf("NO\n");
    }
}
```

Задача 2. Оптимальное расписание

Входной файл	buses.in
Выходной файл	buses.out
Ограничение по времени	0.5 секунды
Ограничение по памяти	256 мегабайт

Как вы помните, ваш друг работает главным диспетчером в одной из компаний, владеющей сетью маршрутных такси. Недавно директор поставил перед ним задачу оптимизировать расписание маршруток на некотором маршруте.

Экономический отдел предоставил вашему другу прогноз пассажиропотока на следующие n часов. Теперь необходимо составить расписание маршруток, а именно для каждого часа определить, выйдет маршрутка на линию в этот час или нет. Известно, что если в i-ый час на маршрут выйдет маршрутка, то она принесет прибыль a_i рублей, при этом a_i может быть как положительным, так и нулевым или отрицательным. Также, в связи с ограниченным размером автопарка и в соответствии с требованиями департамента транспорта, маршрутки должны работать примерно $2/3$ всего времени. Более формально, для любого i ($1 \leqslant i \leqslant n$) величина

$$b = \frac{t_{work}}{2} - t_{skip}$$

должна лежать в пределах от $-k$ до k включительно. Здесь t_{work} — количество часов до i-ого включительно, в которые маршрутки выходили на линию, а t_{skip} — количество часов до i-ого включительно, в которые маршрутки не выходили на линию.

Помогите вашему другу найти максимальную суммарную прибыль маршрута с учетом всех требований.

Формат входных данных

В первой строке входных данных находится количество часов n, на которые необходимо составить расписание, и число k ($1 \leqslant n \leqslant 10^5$, $1 \leqslant k \leqslant 10$). На второй строке находятся n чисел: прогнозируемая прибыль для каждого часа a_i ($|a_i| \leqslant 10^9$).

Все числа во входном файле целые.

Формат выходных данных

Выведите одно число — максимальную суммарную прибыль в рублях.

Пример

Пример

Входной файл	*Выходной файл*
5 1 2 1 3 4 -5	9
5 2 2 1 3 4 -5	10
5 1 5 5 -10 5 5	20

Примечание: В первом примере оптимально выпустить маршрутки на линию в первый, третий и четвертый часы, таким образом, прибыль составит $2 + 3 + 4 = 9$ рублей. Величины t_{work}, t_{skip} и b после каждого часа представлены в таблице:

№ часа	1	2	3	4	5
t_{work}	1	1	2	3	3
t_{skip}	0	1	1	1	2
b	0.5	-0.5	0	0.5	-0.5

Решение, в котором маршрутки выходят на линию в первый, второй, третий и четвертый часы, неправильно, потому что в таком случае уже после третьего часа $t_{work} = 3$, $t_{skip} = 0$, а значит, $b = 1.5$, что недопустимо по условию задачи.

Однако, для второго примера это решение допустимо, так как максимальное значение b равно 2, что удовлетворяет ограничению. Это решение и является оптимальным.

В третьем примере необходимо выпустить маршрутки на линию во все часы, кроме третьего.

Решения, правильно работающие при $n \leqslant 20$, оцениваются не менее чем в 20 баллов.

Решения, правильно работающие при $n \leqslant 1000$, оцениваются не менее чем в 50 баллов.

Решение

Задача решается методом динамического программирования. Заметим, что величина $t_{work} - 2t_{skip}$, назовем ее балансом, должна находиться в пределах от $-2k$ до $2k$ включительно, и может принимать максимум 41 значение (от -20 до 20). Тогда заведем матрицу `ans`, и значение `ans[i][b]` будет содержать максимальную прибыль за первые i часов, если после этого величина баланса равна b. Будем заполнять матрицу в порядке увеличения i, изначально `ans[0][0]` $= 0$, а `ans[0][`$b \neq 0$`]` $= -\infty$, так как изначально баланс 0. По величинам i и b можно найти величину $j = t_{work} = (2i + b)/3$, и использовать динамическое программирование с «просмотром вперед». Существует два перехода: если маршрутка выйдет на линию в $(i + 1)$-ый час, и если не выйдет. В обоих случаях нужно проверить выполнение требований на величину баланса и обновить соответствующую ячейку в матрице `ans`. Итоговый ответ — максимум из `ans[n][b]` среди всех возможных величин b.

Пример правильной программы

```
uses Math;
const maxn = 100005;
const maxk = 20;
const inf = 1000000000000000000;
var ans: array[0..maxn, -maxk..maxk] of int64;
    a: array[1..maxn] of longint;
    n, k: integer;
    answer: int64;
    i, j, b: integer;
    fin, fout: text;

begin
assign(fin, 'buses.in');
reset(fin);
assign(fout, 'buses.out');
rewrite(fout);
read(fin, n, k);
for i := 1 to n do read(fin, a[i]);
k := k * 2;
for i := 0 to n do
  for b := -k to k do ans[i][b] := -inf;
ans[0][0] := 0;
for i := 0 to n - 1 do begin
  for b := -k to k do
    if (b + 2 * i) mod 3 = 0 then begin
      j := (b + 2 * i) div 3;
      if (j < 0) or (j > i) then continue;
      if abs(3 * j - 2 * (i + 1)) <= k then
        ans[i + 1][3 * j - 2 * (i + 1)] :=
          max(ans[i + 1][3 * j - 2 * (i + 1)],
            ans[i][b]);
      if abs(3 * (j + 1) - 2 * (i + 1)) <= k
      then
        ans[i + 1][3*(j + 1) - 2 * (i + 1)]:=
          max(ans[i+1][3*(j+1)-2*(i+1)],
            ans[i][b] + a[i + 1]);
    end;
end;
answer := -inf;
for i := -k to k do
  answer := max(answer, ans[n][i]);
writeln(fout, answer);
close(fout);
end.
```

Задача 3. Часы с кукушкой

Входной файл	`cuckoo.in`
Выходной файл	`cuckoo.out`
Ограничение по времени	0.5 секунды
Ограничение по памяти	64 мегабайта

У Васи на кухне висят часы с кукушкой. Часы устроены так: в каждый ровный час кукушка кукует столько раз, сколько сейчас часов (от 1 до 12), например, ровно в 7:00 кукушка кукует 7 раз. Кроме того, в 30 минут каждого часа (в 0:30, 1:30, 2:30 и т.д.) кукушка кукует ровно один раз.

Васе очень нравится смотреть на то, как кукует кукушка, он любит считать, сколько раз она прокуковала. Но, к сожалению, сегодня мама отправила Васю в магазин за покупками, и поэтому он пропустил несколько моментов, в которые куковала кукушка. Определите, сколько раз всего куковала кукушка за время отсутствия Васи.

Считайте, что кукушка кукует очень быстро. Например, даже в 11:00 она успевает прокуковать 11 раз быстрее, чем за одну минуту, т.е. к моменту 11:01 кукушка уже закончила куковать.

Формат входных данных

Входные данные содержат четыре целых числа H_1, M_1, H_2 и M_2 — время ухода (H_1 часов M_1 минут) и время возвращения (H_2 часов M_2 минут) Васи. Гарантируется, что $0 \leqslant H_{1,2} < 12$ и что $0 < M_{1,2} < 60$. Гарантируется, что момент ухода Васи следует до момента его возвращения (т.е. или $H_1 < H_2$, или $H_1 = H_2$, но $M_1 < M_2$), и что кукушка не кукует ни в момент ухода, ни в момент возвращения (т.е. что $M_{1,2} \neq 0$ и $M_{1,2} \neq 30$).

Вася уходил и приходил в одной и той же половине суток, т.е. между его моментом ухода и моментом прихода не было ни полудня, ни полуночи.

Формат выходных данных

Выведите одно число — сколько раз кукушка куковала за время отсутствия Васи.

Пример

Пример

Входной файл	*Выходной файл*
`2 20 2 40`	`1`
`2 31 4 1`	`8`
`8 10 8 20`	`0`

Примечание: В первом примере кукушка кукует один раз — в момент времени 2:30.

Во втором примере кукушка кукует три раза в момент времени 3:00, один раз в момент времени 3:30, и еще четыре раза в момент времени 4:00 — итого 8 раз.

В третьем примере кукушка не куковала ни разу.

Решение

Это была самая простая задача олимпиады, и она допускала несколько решений.

Самое легкое в реализации решение состояло в следующем. Переведем оба заданных во входном файле момента времени в минуты, прошедшие с начала суток, т.е. вычислим $T_1 = 60H_1 + M_1$ и $T_2 = 60H_2 + M_2$. По условию, первый момент времени идет до второго, поэтому получится $T_1 < T_2$. Теперь пройдем циклом от T_1 до T_2, для каждого промежуточного момента времени определим, кукует ли кукушка в этот момент времени и сколько раз, и просуммируем все эти количества. Это решение наберет полный балл, т.к. минут в половине суток всего 720, и цикл отработает очень быстро.

Как определить, кукует ли кукушка в конкретный момент времени t и, если да, то сколько раз? Это достаточно просто. Кукушка кукует каждые полчаса, т.е. в моменты времени, делящиеся на 30. Причем, если t не делится на 60, то это середина часа и кукушка кукует один раз, а если t делится на 60, то $t/60$ как раз и будет соответствующим количеством часов, т.е. кукушка прокукует $t/60$ раз.

Отметим, что можно было реализовать и еще более быстрое решение, не содержащее цикла по моментам времени. Проще всего было научиться вычислять функцию $f(h, m)$ — сколько раз кукушка куковала начиная с начала суток до момента времени h:m, после чего ответом на задачу будет $f(H_2, M_2) - f(H_1, M_1)$. Реализовать функцию f можно, например, так: в целые часы кукушка куковала $1 + 2 + \ldots + h = h(h + 1)/2$ раз, а в получасы кукушка куковала h раз (в моменты времени 0:30, 1:30, …, $(h-1)$:30) и, возможно, еще раз, если $m > 30$. Можно реализовать и другие подобные решения, различные их варианты см. в архиве решений жюри.

Пример правильной программы

```
var h1,m1,h2,m2:integer;
    i:integer;
    ans:integer;
    f:text;
begin
assign(f,'cuckoo.in');reset(f);
read(f,h1,m1,h2,m2);
close(f);
ans:=0;
for i:=h1*60+m1 to h2*60+m2 do
    if i mod 60=0 then
       ans:=ans+i div 60
    else if i mod 30=0 then inc(ans);
assign(f,'cuckoo.out');rewrite(f);
writeln(f,ans);
close(f);
end.
```

Задача 4. Угадайка

Входной файл	`module.in`
Выходной файл	`module.out`
Ограничение по времени	0.5 секунды
Ограничение по памяти	256 мегабайт

Вася прошел на уроке математики в школе операцию деления с остатком. Чтобы закрепить пройденный материал, он решил потренироваться в выполнении этого действия. Придя домой, Вася попросил маму назвать N произвольных натуральных чисел X_i ($1 \leqslant i \leqslant N$), не превосходящих 10^9. Затем он придумал некоторое натуральное число M, не превосходящее наибольшего из X_i, и вычислил остатки Y_i от деления всех чисел на него.

Вечером к Васе зашел его друг Леша и увидел результаты вычислений. Подумав несколько минут, он радостно сообщил товарищу, что придумал еще одно значение M, удовлетворяющее всем соотношениям. Вася был шокирован. Он не мог поверить, что это возможно, и обратился к Вам с просьбой о помощи.

Помогите Васе найти все возможные значения M, удовлетворяющие всем соотношениям.

Несмотря на то, что и Вася, и его друг Леша — прилежные и умные ученики, они могли ошибиться в вычислениях, и ни одного подходящего M может не существовать.

Формат входных данных

В первой строке находится одно натуральное число N — количество чисел, с которыми Вася проводил операцию деления ($1 \leqslant N \leqslant 1000$).

Далее следуют N строк, в i-ой из которых находятся два целых числа X_i, Y_i, разделенные пробелом ($1 \leqslant X_i \leqslant 10^9$, $0 \leqslant Y_i \leqslant X_i$).

Формат выходных данных

В первой строке должно находиться целое число K — количество различных M, удовлетворяющих всем соотношениям.

Во второй строке должны находиться K чисел, разделенные пробелами — все возможные значения M. Числа можно выводить в произвольном порядке.

Пример

Пример

Входной файл	*Выходной файл*
2 9 4 6 1	1 5
3 14 2 7 1 12 0	2 6 3

Примечание: В первом примере существует единственное число $M = 5$. Остаток от деления 9 на 5 есть 4, а остаток от деления 6 на 5 есть 1.

Во втором примере в качестве M можно взять 3 и 6.

Решения, работающие при $X_i \leqslant 200\,000$, будут набирать 30 баллов.

Решение

Наивное решение, заключающееся в переборе всех возможных значений M и проверке верности всех условий, будет работать слишком медленно, поэтому придумаем более быстрое решение.

Рассмотрим пару чисел X_i и Y_i. По определению остатка от деления число X_i можно представить в виде $X_i = Z_i \cdot M + Y_i$, где Z_i — это некоторое неотрицательное

целое число. Отсюда можно выразить $Z_i = (X_i - Y_i)/M$. Таким образом, число M должно быть делителем всех разностей $X_i - Y_i$. Вычислим наибольший общий делитель всех разностей $X_i - Y_i$ с помощью алгоритма Евклида и обозначим полученное число gcd. Искомые M будут делителями gcd. Число gcd может быть порядка 10^9, поэтому необходимо уметь быстро находить все его делители.

Для того, чтобы их найти, переберем все числа j от 1 до $\sqrt{gcd}$ и проверим, являются ли они делителем gcd. Если ответ положительный, то числа j и gcd/j могут быть искомыми M. Для того, чтобы они попали в ответ, еще необходимо, чтобы они были больше чем наибольшее из Y_i, так как, по определению, остаток от деления меньше соответствующего делителя.

Пример правильной программы

```
Uses math;
var i, n, x, y, g, maxy, cnt, sq: longint;
    ans:array [0..5000] of longint;
    f:text;

function GCD(a,b:longint):longint;
begin
  if b=0 then GCD:=a
  else GCD := GCD(b,a mod b);
end;

begin
  assign(f,'module.in'); reset(f);
  read(f, n);
  g := 0; cnt := 0; maxy := 0;
  for i := 0 to n - 1 do begin
    read(f, x, y);
    if i = 0 then g := x - y
    else g := GCD(g, x - y);
    if (y > maxy) then maxy := y;
  end;
  close(f);
  sq := floor(sqrt(g));
  for i := 1 to sq do begin
    if (g mod i = 0) then begin
      if (i > maxy) then begin
        ans[cnt] := i;
        inc(cnt);
      end;
      if ((g div i) > maxy)
          and ((g div i) <> i) then begin
        ans[cnt] := g div i;
        inc(cnt);
      end;
    end;
  end;
  assign(f,'module.out'); rewrite(f);
  writeln(f, cnt);
  for i := 0 to cnt - 1 do
      write(f, ans[i], ' ');
  close(f);
end.
```

Задача 5. Миша против древних берляндцев

Входной файл	`string.in`
Выходной файл	`string.out`
Ограничение по времени	0.5 секунды
Ограничение по памяти	256 мегабайт

Миша изучает работы по расшифровке древнеберляндских текстов. Древние берляндцы пользовались иероглифичной письменностью. Поскольку число иероглифов очень велико, в трудах, изучаемых Мишей, каждый иероглиф записывается в виде последовательности строчных латинских символов. Древнеберляднское слово представляет собой последовательность из одного или нескольких иероглифов, записанных один за другим слева направо без разделителей. Известно, что древние берляндцы очень любили палиндромы, поэтому все древнеберляндские слова являются палиндромами: первый иероглиф слова совпадает с последним, второй иероглиф — с предпоследним и так далее.

К сожалению, Миша не нашел правила сопоставления иероглифов последовательностям латинских символов, поэтому чтение трудов дается ему очень тяжело. Отчаявшись найти эти правила, Миша решил попробовать сам угадать разбиение

слов на иероглифы, для чего он придумал следующий алгоритм: он выписывает каждое отдельное слово и пытается разбить его на наибольшую по длине последовательность иероглифов так, чтобы эта последовательность при объединении давала исходное слово и при этом являлась палиндромом. Но поиск такого разбиения оказался слишком сложным для Миши, поэтому он обратился за помощью к Вам.

Формат входных данных

В первой и единственной строке записано слово, состоящее из не более чем $4 \cdot 10^5$ строчных латинских букв — древнеберляндское слово, не разбитое на иероглифы.

Формат выходных данных

В первой строке вывести число N — количество иероглифов, в последующих N строках вывести последовательности латинских символов, соответствующие иероглифам исходного слова. При этом объединение всех иероглифов должно давать исходное слово, а последовательность иероглифов должна быть палиндромом. Количество N должно быть максимально возможным для всех корректных разбиений.

Пример

Пример

Входной файл	*Выходной файл*
abacaba	7 a b a c a b a
nnoi	1 nnoi
abcab	3 ab c ab

Примечание: Гарантируется, что решение, работающее для слов, состоящих из не более чем 150 000 латинских букв, набирает 60 баллов.

Решение

Данная задача может быть решена жадным алгоритмом. Пусть s — древнеберляндское слово, записанное латинскими буквами, тогда первым иероглифом должен быть минимальный префикс строки s, который совпадает с ее суффиксом. После

нахождения такого префикса, его можно «отрезать» с начала и с конца, а потом точно так же продолжить разбивать на иероглифы оставшееся слово.

Докажем корректность этого алгоритма. Допустим, что нам известен правильный ответ — последовательность иероглифов h_i, $1 \leqslant i \leqslant k$. Построим последовательность, полученную жадным методом: $\tilde{h}_i$, $1 \leqslant i \leqslant \tilde{k}$. Очевидно, что найдется такое j, что $h_j \neq \tilde{h}_j$, причем без потери общности можно считать что $j = 1$, то есть первые иероглифы отличаются. Понятно, что $|h_j| > |\tilde{h}_j|$, поскольку последовательность $\tilde{h}_i$ построена жадным способом. Поскольку обе подстроки h_1 и $\tilde{h}_1$ являются одновременно префиксами и суффиксами s, то $\tilde{h}_1$ является префиксом и суффиксом строки h_1. Рассмотрим два случая:

1. $2|\tilde{h}_1| < |h_1|$, тогда h_1 представимо в виде объединения трех строк $\tilde{h}_1 + x + \tilde{h}_1$, где x — некоторая строка. Но в этом случае правильный ответ можно улучшить на 4, если вместо h_1 взять $\tilde{h}_1$, x и $\tilde{h}_1$, то есть последовательность h_i не оптимальна.

2. $2|\tilde{h}_1| = |h_1|$, этот случай отличается от предыдущего только отсутствием строки x между вхождениями $\tilde{h}_1$ в h_1, поэтому снова последовательность h_i не оптимальна.

3. $2|\tilde{h}_1| > |h_1|$, в этом случае h_1 снова может быть разбита на три строки $t+u+v$, где $h_1 = t+u = u+v$. Легко показать, что u является префиксом и суффиксом h_1, но тогда u является префиксом и суффиксом s и при этом, очевидно $|s| < |\tilde{h}_1|$, что не может быть верным, поскольку $\tilde{h}_1$ — минимальная строка, являющаяся префиксом и суффиксом.

Тривиальная проверка подстрок строки s на совпадение не проходит по времени, поэтому для такой проверки нужно использовать хеши.

Пример правильной программы

```
{$Q-}
const
    nmax = 400000;
    q = 239;

var
    s: string;
    i, j, curl, curr, il, ir: integer;
    l, r: array [1 .. nmax] of integer;
    hash, qq: array [0 .. nmax] of int64;

function gethash(l, r: integer): int64;
begin
result:= hash[r - 1] - hash[l - 1] * qq[r - l];
end;

begin
assign(input,'string.in');reset(input);
assign(output,'string.out');rewrite(output);
readln(s);

hash[0] := 0;
qq[0] := 1;
for i := 1 to length(s) do begin
    hash[i] := hash[i - 1] * q + ord(s[i]);
    qq[i] := qq[i - 1] * q;
end;

curl := 1;
curr := length(s) + 1;
il := 1;
ir := 1;
l[il] := curl;
r[ir] := curr;
while curl < curr do begin
    for i := 1 to nmax do begin
        if curl + i > curr - i then begin
            curl := curr;
            break;
        end;
        if gethash(curl, curl + i) =
            gethash(curr - i, curr) then begin
            inc(curl, i);
            dec(curr, i);

            inc(il);
```

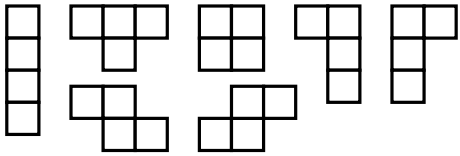
```
            l[il] := curl;
            if curr <> curl then begin
                inc(ir);
                r[ir] := curr;
            end;
            break;
        end;
    end;
end;
while ir > 0 do begin
    inc(il);
    l[il] := r[ir];
    dec(ir);
end;
writeln(il - 1);
for i := 1 to il - 1 do begin
    for j := l[i] to l[i + 1] - 1 do
        write(s[j]);
    writeln;
end;
end.
```

Задача 6. Площадь Пажитнова

Входной файл	`tetris.in`
Выходной файл	`tetris.out`
Ограничение по времени	0.5 секунды
Ограничение по памяти	256 мегабайт

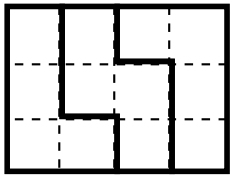
В прошлом году всемирно известной игре Тетрис исполнилось 30 лет. В Берляндии очень любят Тетрис, и поэтому решили назвать одну из городских площадей в столице в честь изобретателя игры, Алексея Леонидовича Пажитнова. Дизайн площади, разумеется, должен напоминать всем о Тетрисе. Поэтому она будет замощена плитками в виде фигурок игры.

Как известно, фигурки Тетриса состоят из четырех «клеток». Берляндский завод тротуарной плитки может выпускать плитку в виде любых фигурок Тетриса. Размер каждой «клетки» такой плитки составляет 1×1 метр. Все возможные фигурки Тетриса представлены на рисунке ниже.



Площадь Пажитнова имеет вид прямоугольника длиной N и шириной M метров. Для упрощения замощения она вся расчерчена на «клетки» размером 1×1 метр, то есть представляет собой сетку. При замощении «клетки» плитки должны совпадать с «клетками» площади. Плитки можно использовать только целиком, они не должны перекрываться или выступать за пределы площади, но их можно поворачивать на углы, кратные 90°. Разумеется, плитки должны покрывать площадь полностью. Размерами промежутков между плитками, покрывающими соседние «клетки», можно пренебречь.

Например, площадь длиной 3 и шириной 4 метра можно покрыть тремя плитками, как показано на рисунке ниже.



Напишите программу, которая поможет берляндцам найти подходящее покрытие.

Формат входных данных

В первой и единственной строке записаны два натуральных числа N и M ($1 \leqslant \leqslant N, M \leqslant 100$), разделенные пробелом — длина и ширина площади соответственно.

Формат выходных данных

В первой строке выведите число K — минимальное количество плиток, нужное для того, чтобы замостить площадь. Если площадь замостить невозможно, то выведите одно число `-1`.

Если замощение существует, то выведите его, начиная со второй строки. А именно, считайте, что плитки, используемые в замощении, занумерованы натуральными числами от 1 до K. Выведите N строк по M чисел, разделенных пробелами, в каждой. Эти числа должны соответствовать номерам плиток, покрывающим соответствующие «клетки» площади.

Если существует несколько замощений с минимальным количеством плиток, то выведите любое из них.

Пример

Пример

Входной файл	*Выходной файл*

`3 4`	`3`
	`1 2 3 3`
	`1 2 2 3`
	`1 1 2 3`

`3 3`	`-1`

Примечание: Первый пример соответствует картинке из условия.

Решение

Довольно очевиден факт, что если площадь можно замостить плитками Тетриса, то площадь площади $N \cdot M$ делится на 4. В самом деле, площадь плитки всегда равна 4, и можно использовать только целые плитки без наложений и перекрытий, значит, покрываемая ими площадь кратна площади одной плитки. Это необходимое условие существования покрытия. Отсюда же, кстати, следует, что количество плиток одинаково во всех покрытиях.

Является ли это условие достаточным? Если $N \cdot M$ кратно 4, то что это может нам сказать о самих N и M? Тут возможны два варианта:

1. N кратно 2 и M кратно 2;

2. либо N, либо M кратно 4.

В первом случае можно замостить площадь квадратами 2×2. Во втором случае можно воспользоваться «палками» 1×4, разместив их длинную сторону параллельно тому измерению, которое и кратно 4.

Пример правильной программы

```
var n,m:integer;
    i,j:integer;
    f:text;
begin
assign(f,'tetris.in');reset(f);
read(f,n,m);
close(f);
assign(f,'tetris.out');rewrite(f);
if n*m mod 4<>0 then writeln(f,-1)
else begin
  writeln(f,n*m div 4);
  for i:=0 to n-1 do begin
      for j:=0 to m-1 do begin
          if n mod 4=0 then
              write(f,j*(n div 4)+i div 4+1)
          else if m mod 4=0 then
              write(f,i*(m div 4)+j div 4+1)
          else
              write(f,(i div 2)*(m div 2)+
                  j div 2+1);
          write(f,' ');
      end;
      writeln(f);
  end;
end;
close(f);
end.
```

Содержание