ОБЛАСНІ ОЛІМПІАДИ З ІНФОРМАТИКИ: ДОСВІД КІРОВОГРАДСЬКОЇ ОБЛАСТІ

Мельник В.І., Сімоненко В.Є.

Кіровоградська обласна олімпіада з інформатики проходить у два тури. На кожному з турів учасникам було запропоновано по п'ять різнопланових задач різної складності. Перші три задачі кожного з турів були нескладними і не вимагали знань тих чи інших алгоритмів. Дві інші задачі були важчими, і вимагали від учасників добрих навичок програмування і знань певного класу алгоритмів.

ЗАВДАННЯ І ТУРУ

1. ВПІЗНАЙ ДРАКОНА

Ім'я вхідного файлу: binary.in
Ім'я вихідного файлу: binary.out
Максимальний час роботи на одному тесті: 1 секунда.

Дракони з Нетутешніх островів мають кількість голів, що є степенем двійки. Дано натуральне число N. Визначити, чи означає воно кількість голів «нетутешнього» Дракона.

Формат вхідних даних: вхідний файл містить дане число N ($1 \le N \le 20000$).

Формат вихідних даних: У вихідний файл виведіть 1 — якщо дане число є степенем двійки, та 0 — якщо число не є степенем двійки.

Приклад файлів вхідних і вихідних даних

binary.in	binary.out
16	1
20	0
2	1

Розв'язання

Дану задачу можна розв'язати двома способами.

1 спосіб. Будемо ділити дане число голів дракона N на 2 доти, доки воно ділиться на 2. Якщо, в результаті вищевказаних дій отримаємо 1, то дане число N задовольняє умову задачі, в іншому випадку — ні.

2 спосіб. На відміну від першого способу, будемо перебирати всі числа, які є степенями 2 доти, доки не отримаємо число рівне, або більше N.

```
Const File_In='binary.in';
      File_Out='binary.out';
var n: longInt;
    i: LongInt;
 begin
   Assign(Input, File_In);
   Reset(Input);
   ReadLn(n);
   Close(Input);
   Assign(Output, File_Out);
   Rewrite(Output);
   i:=1;
   While i<n do
      i:=i*2;
  If i=n then Writeln(1)
         else WriteLn(0);
   Close(Output)
 End.
```

2. РЯДКИ В КНИЖЦІ

Ім'я вхідного файлу: book.in.
Ім'я вихідного файлу: book.out.
Максимальний час роботи на одному тесті: 1 секунда.

У книжці про Драконів на одній сторінці міститься K рядків. Отже, на 1-й сторінці друкуються рядки з 1-го по K-ий, на другій — з $(K+1)$-го до $(2 \cdot K)$-го і т. д. Напишіть програму, що за номером рядка в тексті визначає номер сторінки, на якій буде надрукований цей рядок, і порядковий номер цього рядка на сторінці.

Формат вхідних даних: вхідний файл містить число K — кількість рядків, що друкується на сторінці, і число N — номер рядка ($1 \le K \le 200$, $1 \le N \le 20000$).

Формат вихідних даних: у вихідний файл виведіть два числа — номер сторінки, на якій буде надрукований цей рядок і номер рядка на сторінці.

Приклад файлів вхідних і вихідних даних

book.in	book.out
50 1	1 1
20 25	2 5
15 43	3 13

Розв'язання

Очевидно, що якщо N кратне K, то ми знаходимось на сторінці N div K, а номер рядка рівний K. В іншому випадку, номер сторінки визначається як N div $K+1$, а номер рядка N mod K.

```
Const File_In='book.in';
      File_Out='book.out';
var K,N : integer;
 begin
   Assign(Input, File_In);
   Reset(Input);
   ReadLn(K,N);
   Close(Input);
   Assign(Output, File_Out);
   Rewrite(Output);
   if N mod K = 0 then WriteLn(N div K, ' ', K)
       else WriteLn (N div K + 1, ' ', N mod K);
   Close(OUTPUT);
 end.
```

3. БИТВА ДРАКОНІВ

Ім'я вхідного файлу: dragon.in.
Ім'я вихідного файлу: dragon.out.
Максимальний час роботи на одному тесті: 1 секунда.

Відомо, що в дракона може бути кілька голів і його сила визначається числом голів. Але як визначити силу зграї драконів, у якій кілька драконів і в кожного з них певне число голів? Імовірно, ви вважаєте, що це значення обчислюється як сума всіх голів? Це далеко не так, інакше було б занадто просто обчислити силу зграї драконів. Виявляється, що шукане значення дорівнює добутку значень числа голів кожного із драконів.

Наприклад, якщо в зграї 3 дракони, у яких 3, 4 і 5 голів відповідно, то сила зграї дорівнює $3 \times 4 \times 5 = 60$.

Потрібно написати програму, яка за заданою сумарною кількістю голів зграї драконів, знайде максимально можливе значення сили цього лігвища драконів.

Формат вхідних даних: у єдиному рядку вхідного файлу записане натуральне число N ($0 < N < 100$) — кількість голів зграї драконів.

Формат вихідних даних: у єдиний рядок вихідного файлу потрібно вивести максимально можливе значення сили, що може бути в зграї драконів з N голів.

Приклад файлів вхідних і вихідних даних

dragon.in	dragon.out
6	9
8	18
13	108

Розв'язання

З математики відомо, що найбільший добуток в розкладі числа на доданки, утвориться при максимально можливій кількості 3. Отже, визначимо кількість 3, яку містить задане число N. Розглянемо залишки, які можуть утворитися, при діленні N на 3. Якщо залишок 0, то потрібно перемножити N div 3 трійок, якщо залишок 1, то, очевидно, що добуток буде більшим, якщо замінити одну трійку із залишком на 2×2, тобто отримуємо результат, перемноживши N div 3–1 трійок на 2×2. При залишку 2 N div 3 трійок слід домножити на 2.

```
Const File_In='dragon.in';
      File_Out='dragon.out';
var n: Byte;
    Count: LongInt;
    i, t: LongInt;
 begin
   Assign(Input, File_In);
   Reset(Input);
   ReadLn(n);
   Close(Input);
   Assign(Output, File_Out);
   Rewrite(Output);
   Count:= n mod 3;
   t:=n div 3;
   Case Count of
     0: count:=1;
     1: begin Count:=4; Dec(t); end;
   end;
   for i:=1 to t do
     Count:=Count*3;
   if n<3 then WriteLn(n)
          else WriteLn(Count);
   Close(Output);
 end.
```

4. РЯДОК

Ім'я вхідного файлу: string.in.

Ім'я вихідного файлу: string.out.

Максимальний час роботи на одному тесті: 1 секунда.

Дане слово, Вам необхідно в ньому переставити місцями два символи, які знаходяться на різних позиціях в цьому слові, так щоб в результаті отримати слово, лексикографічно найменше з усіх можливих.

Формат вхідних даних: у єдиному рядку вхідного файлу записане слово, яке складається з маленьких літер латинського алфавіту. Гарантується, що його довжина не буде перевищувати 100000 і не буде меншою за 2 символи.

Формат вихідних даних: у єдиному рядку вихідного файлу повинно знаходитись рівно два числа — позиції символів у даному слові, що треба переставити між собою. Якщо існує декілька варіантів відповіді — виведіть ту, у якій менше перше число, якщо таких більше однієї, то ту, у якій менше друге число.

Приклад файлів вхідних і вихідних даних

string.in	string.out
adbsab	2 5

Розв'язання

Позначимо i-ий символ даного рядку s_i. Зрозуміло, що для того щоб результуючий рядок був якнайменше лексикографічний, треба його «поліпшити» в якомога меншій позиції. Більш детально: якщо ми переставляємо місцями i-ий та j-ий символи рядку ($i<j$), то результат буде найменшим, якщо $s_i>s_j$, де i — мінімально можливий; у випадку рівних i вигідно переставити символ s_j — лексикографічно найменший з усіх можливих; якщо ж таких j теж декілька, то j — повинен бути максимально можливим. Залишилось швидко реалізувати вище наведені міркування. Будемо перебирати всі можливі індекси i з кінця даного рядка в початок, одночасно змінюючи індекс j. Якщо пара (i; j) задовольняє нашим вимогам, то запам'ятаємо її як проміжний результат, в іншому випадку, якщо $s_i<s_j$ то значення j має бути змінено на значення i на даному кроці.

У задачі є декілька особливих випадків. Якщо символи рядка знаходяться в неспадному порядку, то не можливо буде отримати рядок лексикографічно менший за даний, отже, треба вибрати два таких символи, щоб втрати були якнайменше можливими. Нехай в рядку є два рівні сусідні символи, тоді перестановка їх не змінить рядка, а, отже, результат буде оптимальним, в іншому випадку правильно буде переставити два останніх символи. Складність розв'язку O(N), де N — довжина рядка.

```
#include <iostream>
#include <cstdio>
#include <sstream>
#include <vector>
#include <set>
#include <map>
#include <queue>
#include <algorithm>
#include <numeric>
#include <functional>
#include <string>
#include <cstdlib>
#include <cmath>
#include <list>
using namespace std;
#define FOR(i,a,b) for(int i=(a),_b(b);i<_b;++i)
#define FORD(i,a,b) for(int i=(a),_b(b);i>=_b;--i)
#define REP(i,n) FOR(i,0,n)
#define ALL(a) (a).begin(),a.end()
#define SORT(a) sort(ALL(a))
#define UNIQUE(a) SORT(a),
  (a).resize(unique(ALL(a))-a.begin())
#define SZ(a) ((int) a.size())
#define pb push_back
#define VAR(a,b) __typeof(b) a=(b)
#define FORE(it,a) for(VAR(it,(a).begin());
          it!=(a).end();it++)
#define X first
#define Y second
#define DEBUG(x) cout << #x << " = " << x << endl;
#define INF 1000000000
#define CHECK
typedef vector<int> VI;
typedef vector< vector<int> > VVI;
typedef pair<int, int> PII;
typedef vector<PII> VPII;
typedef long long ll;
char s[123456];
int main() {
    freopen("string.in", "r", stdin);
```

```
    freopen("string.out", "w", stdout);
    gets (s);
#ifdef CHECK
    assert (strlen (s) >= 2 && strlen (s) <= 100000);
    for (int i = 0; s[i]; ++i)
    assert (s[i] >= 'a' && s[i] <= 'z');
#endif
    int N = strlen (s);
    int p = N - 1;
    vector <int> ans (2);
    for (int i = N - 1; i >= 0; --i) {
        if (s[i] < s[p]) p = i;
        if (s[i] > s[p]) {
            ans[0] = i;
            ans[1] = p;
        }
    }
    if (ans[0] == ans[1]) {
        ans[0] = N - 2;
        ans[1] = N - 1;
        REP (i, N - 1)
            if (s[i] == s[i + 1]) {
                ans[0] = i;
                ans[1] = i + 1;
                break;
            }
    }
    printf ("%d %d\n", ans[0]+1, ans[1]+1);
    return 0;
}
```

5. ПРОГУЛЯНКА ПО МІСТУ

Ім'я вхідного файлу: travel.in.

Ім'я вихідного файлу: travel.out.

Максимальний час роботи на одному тесті: 1 секунда.

У Кракові, недалеко від берега Вісли, стоїть Дракон: кам'яний, з пухирчастим тілом, величезними лапами, до неба піднятою головою. Місто Краків — улюблене місто туристів, які з'їзджаються сюди звідусіль. Петрик — один із них. Карта Кракова є таблицею, розміром N×M, клітинками якої є квартали, деякі з яких доступні для руху, а деякі ні. Також на карті відмічені цікаві місця, причому це також квартали, по яких можливий рух. Вася вибрав маршрут для ознайомлення з містом. Він почне з клітинки (1;1), прийде в клітинку (N;M), а потім знову повернеться туди, звідки прийшов (у клітинку (1;1)). Звичайно, Петрик хоче переглянути якнайбільше різних цікавих місць. Так як Петрик знає Краків дуже погано, та й взагалі заблукати тут дуже небажано, він вирішив, що на шляху до клітинки (N;M) буде завжди йти вправо, або вниз (відносно карти), а на зворотнім — вліво, або вверх. Допоможіть Петрикові знайти максимальну кількість різних цікавих місць, у яких він може побувати.

Формат вхідних даних: у першому рядку вхідного файлу міститься два числа N та M ($1 \le N, M \le 100$). У наступних N рядках міститься опис карти — рядок з M літер, причому «#» означає квартал недоступний для руху, «*» — місце для перегляду, а «.» (точка) — квартал доступний для руху, який слугує Васі лише клітинкою для руху. Гарантується, що клітинки (1;1) й (N;M) не будуть позначені символом «#».

Формат вихідних даних: у єдиному рядку вихідного файлу повинно міститися одне число — максимальна кількість різних місць для перегляду, у яких може побувати Петрик, протягом свого маршруту, або −1, якщо маршрут здійснити не вдасться.

Приклад файлів вхідних і вихідних даних

string.in	string.out
4 5 **.*. ..##. *#*.. *.*.*	7

Розв'язання

Давайте розіб'ємо маршрут Васі на дві частини (перша — рух з лівого верхнього кута в правий нижній, а друга навпаки), і, не втрачаючи загальності, будемо розглядати його другу частину як «перевернуту» (так щоб вона, як і перша, — прямувала з лівого верхнього кута в правий нижній). Давайте тепер допоможемо Васі, й покличемо його друга Івана. Тепер нехай Вася буде ходити по першій частині свого маршруту, а Іван по його другій, причому домовимось, що вони свої кроки будуть виконувати абсолютно одночасно. У такому випадку нескладно помітити, що, якщо деяка клітинка входить в маршрут двічі, то в ній і Вася, і Іван побуватимуть в один і той самий момент часу. Тепер вже можна використати принцип динамічного програмування, нехай $A_{x1,y1,x2,y2}$ — буде максимальна кількість різних цікавих місць, у яких можуть побувати Вася та Іван (причому одночасні потрапляння в одне й те саме цікаве місце рахується за одиницю, а не за двійку), якщо зараз Вася знаходиться в клітинці з координатами (x_1,y_1), а Іван в (x_2,y_2).

Нехай $B_{x,y}$=1, якщо клітинка (x,y) цікава та 0 навпаки. Тоді, якщо $(x_1,y_1)\neq(x_2,y_2)$, то $A_{x1,y1,x2,y2}=\text{Max}(A_{x1\text{-}1,y1,x2\text{-}1,y2}, A_{x1,y1\text{-}1,x2\text{-}1,y2}, A_{x1\text{-}1,y1,x2,y2\text{-}1}, A_{x1,y1\text{-}1,x2,y2\text{-}1})+B_{x1,y1}+B_{x2,y2}$, якщо ж $(x_1,y_1)=(x_2,y_2)$, то останній доданок треба видалити. Також треба зауважити, що тут для всіх станів, які не можна досягти $A_{x1,y1,x2,y2}$=0. Тепер порахуємо кількість станів, що і буде оцінкою складності для даного алгоритму. Очевидно, що Вася та Іван знаходяться завжди на одній діагоналі, так як всього діагоналей порядку $O(N+M)$, кількість клітинок на кожній з них теж $O(N+M)$, то загальна складність алгоритму буде порядку $O((N+M)^3)$. Насправді коефіцієнт при цій оцінці досить малий, і тому реалізація даного алгоритму працює дуже швидко.

```
#include <iostream>
#include <cstdio>
#include <sstream>
#include <vector>
#include <set>
#include <map>
#include <queue>
#include <algorithm>
#include <numeric>
#include <functional>
#include <string>
#include <cstdlib>
#include <cmath>
#include <list>
using namespace std;
#define FOR(i,a,b) for(int i=(a),_b(b);i<_b;++i)
#define FORD(i,a,b) for(int i=(a),_b(b);i>=_b;--i)
#define REP(i,n) FOR(i,0,n)
#define ALL(a) (a).begin(),a.end()
#define SORT(a) sort(ALL(a))
#define UNIQUE(a) SORT(a),
        (a).resize(unique(ALL(a))-a.begin())
#define SZ(a) ((int) a.size())
#define pb push_back
```

```
#define VAR(a,b) __typeof(b) a=(b)
#define FORE(it,a) for(VAR(it,(a).begin());
          it!=(a).end();it++)
#define X first
#define Y second
#define DEBUG(x) cout << #x << " = " << x << endl;
#define INF 1000000000
#define CHECK
typedef vector<int> VI;
typedef vector< vector<int> > VVI;
typedef pair<int, int> PII;
typedef vector<PII> VPII;
typedef long long ll;
int main() {
   freopen("tour.in", "r", stdin);
   freopen("tour.out", "w", stdout);
   int n, m;
   cin >> n >> m;
#ifdef CHECK
   assert (1 <= n && n <= 100);
   assert (1 <= m && m <= 10000);
#endif
   vector <vector <bool> > a (n, vector <bool> (n));
   int q, w;
   REP (i, m) {
     cin >> q >> w;
#ifdef CHECK
     assert (1 <= q && q <= n);
     assert (1 <= w && w <= n);
#endif
     --q, --w;
     a[q][w] = true;
   }
#ifdef CHECK
   REP (i, n)
     REP (j, i)
       assert (a[i][j] + a[j][i] > 0);
#endif
   VI b;
   VI c;
   REP (i, n)
     c.pb (i);
   random_shuffle (ALL (c));
   REP (h, n) {
     int i = c[h];
    bool ok = false;
    REP (j, SZ (b))
          if (a[i][b[j]]) {
             b.insert (b.begin() + j, i);
             ok = true;
             break;
          }
    if (!ok)
       b.pb (i);
   }
   REP (i, SZ (b)) {
    if (i)
       cout << ' ';
    cout << (b[i]+1);
   }
   cout << endl;
  return 0;
}
```

ЗАВДАННЯ ІІ ТУРУ

1. БІЛЬШЕ, МЕНШЕ

Ім'я вхідного файлу: equality.in.
Ім'я вихідного файлу: equality.out.
Максимальний час роботи на одному тесті: 1 секунда.

Дано два числа. Необхідно визначити, який знак нерівності потрібно поставити між ними.

Формат вхідних даних: вхідний файл містить один рядок, у якому записано два числа, розділених одним пропуском. Кількість цифр у кожному числі не більше 255.

Формат вихідних даних: у вихідний файл виведіть одне число: 1 — якщо перше число більше за друге, –1 — якщо перше число менше за друге, 0 — якщо числа рівні.

Приклади вхідних та вихідних даних

equality.in	equality.out
1 12	–1
123 99	1
10 10	0

Розв'язання

У даній задачі необхідно порівняти два довгих числа. Доречніше кожне число зберігати як окремий рядок. Тоді число буде більшим, якщо у нього кількість цифр більша, тобто якщо більша довжина відповідного рядка. Якщо ж довжини рівні, то рядки порівнюються як звичайні числа.

```
Var q, w: String;
     simv: Char;
     i, j: Byte;
 Begin
   Assign(Input,'equality.in');
   Reset(Input);
   Assign(Output,'equality.out');
   Rewrite(Output);
   Read(simv);
   q:='';
   While simv<>' ' do
     begin
       q:=q+simv;
       Read(simv);
     end;
   Read(simv);
   w:='';
   While NOT EOLN do
      begin
        w:=w+simv;
        Read(simv);
      end;
   w:=w+simv;
   if LengTh(q)>LengTh(w)
     then WriteLn('1')
     else if LengTh(q)<LengTh(w)
       then WriteLn('-1')
       else if q>w then WriteLn('1')
            else if q<w then WriteLn('-1')
                 else WriteLn('0');
   Close(Input);
   Close(Output)
end.
```

2. ОДИНИЧКИ

Ім'я вхідного файлу: ones.in.
Ім'я вихідного файлу: ones.out.
Максимальний час роботи на одному тесті: 1 секунда.

Дмитрик вивчив цифру 1 і сьогодні цілу годину виводив одиничку у зошиті. Старший брат Дмитрика Петрик навчається в математичному класі. Він щойно вивчив алгоритм знаходження НСД (найбільший спільний дільник) двох чисел. Тепер його цікавить, як знайти НСД двох чисел, які записав Дмитрик у своєму зошиті з одиниць у двох різних рядках.

Формат вхідних даних: вхідний файл містить два рядки, у кожному з яких записано по одному числу, кожне з яких складається тільки з одиниць. Кількість цифр у кожному числі не більше 20000.

Формат вихідних даних: у вихідний файл виведіть одне число — кількість цифр, що містить найбільший спільний дільник даних чисел.

Приклади вхідних та вихідних даних

ones.in	ones.out
1 11	1
111 11	1
11 1111	2

Розв'язання

Підрахуємо кількість одиничок у кожному з даних чисел. Нехай у першому числі їх буде a, а у другому — b. Легко побачити, що кількість цифр, що містить найбільший спільний дільник даних чисел — це найбільший спільний дільник чисел a і b.

```
Var a, b, i: Integer;
    q: Char;
 begin
   Assign(Input,'ones.in');
   Reset(Input);
   a:=0;
   While NOT EOLN do
    begin
      Read(q);
      if q='1' then Inc(a)
    end;
   ReadLn;
   b:=0;
   While NOT EOLN do
    begin
      Read(q);
      if q='1' then Inc(b)
    end;
   While (a>0) and (b>0) do
    if a>b then a:=a mod b
           else b:=b mod a;
   Assign(Output, 'ones.out');
   Rewrite(Output);
   WriteLn(a+b);
   Close(Input);
   close(Output)
 end.
```

3. НОВОРІЧНІ ІГРАШКИ

Ім'я вхідного файлу: toys.in.

Ім'я вихідного файлу: toys.out.

Максимальний час роботи на одному тесті: 1 секунда.

У кожного свята є один недолік — рано чи пізно, але воно закінчується. Ось і новорічні свята завершились і малому Дмитрику необхідно скласти іграшки у коробки. Частину іграшок він склав у одну коробку, а частину в іншу. Старший брат Дмитрика Петрик навчається в математичному класі. І його цікавить, чи можна перекласти всі іграшки в одну з коробок (кожна коробка вміщує всі іграшки), якщо з однієї коробки в іншу можна перекладати стільки іграшок, скільки в іншій коробці.

Формат вхідних даних: вхідний файл містить два числа N і M — кількість іграшок у першій та другій коробці ($1 \le N, M \le 2000000000$).

Формат вихідних даних: у вихідний файл виведіть 1 — якщо можна перекласти іграшки у одну коробку, або 0 — якщо такої можливості немає.

Приклади вхідних та вихідних даних

toys.in	toys.out
2 6	1
1 5	0
20 5	0

Розв'язання

У даній задачі можна моделювати процес перекладання іграшок з однієї коробки до іншої. Але цей процес може бути довгим і, тому матимемо програш у часі.

Оптимальним є таке рішення: скоротимо загальну кількість іграшок на їх найбільший спільний дільник. Якщо знайдений результат є степенем двійки, тоді ми можемо перекласти іграшки в одну коробку, в іншому випадку — ні.

```
Var i,k,n, m, S: LongInt;
  Begin
    Assign(Input,'TOYS.in');
    Reset(Input);
    Assign(Output,'TOYS.out');
    Rewrite(Output);
    ReadLn(n,m);
    s:=n+m;
    While (n>0) and (m>0) do
          if n>m then n:=n mod m
                 else m:=m mod n;
    s:=s div (n+m);
    While s mod 2=0 do s:=s div 2;
    If s=1 then WriteLn(1)
    else WriteLn(0);
    Close(Input);
    Close(Output);
  end.
```

4. ДВІ ПОСЛІДОВНОСТІ

Ім'я вхідного файлу: twoseq.in.

Ім'я вихідного файлу: twoseq.out.

Максимальний час роботи на одному тесті: 2 секунди.

Задано дві послідовності цілих чисел, у кожній з яких всі числа різні. Знайдіть довжину найбільшої спільної підпослідовності даних послідовностей.

Формат вхідних даних: у першому рядку вхідного файлу записано два числа N та M ($1 \le N, M \le 100000$). У другому рядку записано N цілих чисел — числа першої послідовності. У третьому — M цілих чисел — числа другої послідовності. Усі числа у вхідному файлі по модулю не перевищують 10^9.

Формат вихідних даних: у єдиному рядку вихідного файлу повинно знаходитися єдине число — відповідь на задачу.

Приклади вхідних та вихідних даних

twoseq.in	twoseq.out
4 5 2 –1 10 30 2 3 10 4 30	3

Розв'язання

Позначимо A_i — i-ий член першої послідовності, а B_i — відповідно другої. Найпростішим поліноміальним розв'язком даної задачі є динамічне програмування. Через $D_{x,y}$ позначимо довжину найбільшої спільної підпослідовності послідовностей x перших елементів першої та y елементів другої послідовності. Тоді, користуючись стандартними принципами динамічного програмування, якщо $A_x=B_y$, то $D_{x,y}=D_{x-1,y-1}+1$, інакше $D_{x,y}=Max(D_{x,y-1},D_{x-1,y})$, якщо прийняти $D_{x,y}=0$ при $x=0$ або $y=0$. Даний розв'язок має складність $O(N-M)$, що, очевидно, забагато для даних обмежень на N та M.

Треба думати над більш ефективним алгоритмом. Можна помітити, що вище наведений розв'язок не використовує той факт, що всі числа в кожній з послідовностей різні, про

що говорить умова задачі. Давайте скористаємось цим, утворимо нову послідовність C так: для кожного елемента другої послідовності знайдемо рівний йому в першій та індекс такого елементу додамо до C, пропускаючи ті елементи другої послідовності, для яких такого елемента не існує. Це зробити достатньо легко, відсортувавши елементи двох даних послідовностей, зберігши їх індекси на початку. Тепер можна побудувати взаємно однозначне відображення між множиною зростаючих підпослідовностей послідовності C, та множиною всіх можливих спільних підпослідовностей двох даних послідовностей. Отже, ми звели дану задачу до знаходження найбільшої зростаючої підпослідовності. Загальновідомим фактом є те, що така задача має розв'язок зі складністю $O(K\log K)$ (де K — довжина послідовності), що нас цілком задовольняє. Для цього будемо поелементно розглядати послідовність C, через S_h позначимо найменший елемент, яким може закінчуватись підпослідовність довжини h послідовності розглянутих елементів C. Зверніть увагу на те, що елементи масиву S будуть розташовані у зростаючому порядку і, розглядаючи черговий елемент послідовності C в S, змінюється рівно один елемент. Користуючись бінарним пошуком, приходимо до потрібної нам складності.

Загальна складність такого розв'язку $O(N+M)$.

Задача може бути трохи ускладнена, якщо в умові дозволити повторення елементів, але обмежити кількість таких повторень для кожного конкретного числа (наприклад 10). Модифікуйте вище наведений алгоритм (або придумайте свій власний) для розв'язку такої задачі.

```
#include <iostream>
#include <cstdio>
#include <sstream>
#include <vector>
#include <set>
#include <map>
#include <queue>
#include <algorithm>
#include <numeric>
#include <functional>
#include <string>
#include <cstdlib>
#include <cmath>
#include <list>
using namespace std;
#define FOR(i,a,b) for(int i=(a),_b(b);i<_b;++i)
#define FORD(i,a,b) for(int i=(a),_b(b);i>=_b;—i)
#define REP(i,n) FOR(i,0,n)
#define ALL(a) (a).begin(),a.end()
#define SORT(a) sort(ALL(a))
#define UNIQUE(a) SORT(a),
        (a).resize(unique(ALL(a))—a.begin())
#define SZ(a) ((int) a.size())
#define pb push_back
#define VAR(a,b) __typeof(b) a=(b)
#define FORE(it,a) for(VAR(it,(a).begin());
        it!=(a).end();it++)
#define X first
#define Y second
#define DEBUG(x) cout << #x << " = " << x << endl;
#define INF 1000000000
#define CHECK
typedef vector<int> VI;
typedef vector< vector<int> > VVI;
typedef pair<int, int> PII;
typedef vector<PII> VPII;
typedef long long ll;
int main() {
  freopen("twoseq.in", "r", stdin);
    freopen("twoseq.out", "w", stdout);
    int n, m;
    scanf ("%d%d", &n, &m);
    #ifdef CHECK
      assert (n >= 1 && n <= 100000);
      assert (m >= 1 && m <= 100000);
    #endif
    vector <int> a (n);
    VPII b (m);
    REP (i, n) {
      scanf ("%d", &a[i]);
      #ifdef CHECK
          assert (abs (a[i]) <= 1000000000);
      #endif
    }
    REP (i, m) {
      scanf ("%d", &b[i].X);
      b[i].Y = i;
      #ifdef CHECK
          assert (abs (b[i].X) <= 1000000000);
      #endif
    }
    #ifdef CHECK
      vector <int> check_a(a);
      UNIQUE (check_a);
      assert (SZ (check_a) == SZ (a));
      VI check_b;
      REP (i, SZ (b))
          check_b.pb (b[i].X);
      UNIQUE (check_b);
      assert (SZ (check_b) == SZ (b));
    #endif
    vector <int> c;
    SORT (b);
    REP (i, SZ (a)) {
        VPII::iterator it = lower_bound (ALL (b),
            make_pair (a[i], INT_MIN));
        if (it == b.end()) continue;
        if (it->X != a[i]) continue;
        c.pb (it->Y);
    }
    vector <int> lcs;
    REP (i, SZ (c)) {
        int j = lower_bound (ALL (lcs), c[i])—lcs.begin();
        if (j == SZ (lcs))
            lcs.pb (c[i]);
        lcs[j] = c[i];
    }
    printf ("%d\n", SZ (lcs));
    return 0;
}
```

5. TOUR DE FRANCE

Ім'я вхідного файлу: tour.in.
Ім'я вихідного файлу: tour.out.
Максимальний час роботи на одному тесті: 1 секунда.

На цей раз Вам доведеться допомогти організаторам всесвітньо відомих велоперегонів Tour De France приготувати трасу, по якій буде проходити змагання. Організатори перегонів надали вам карту. На ній відмічено N пунктів і M однонаправлених доріг, які з'єднують ці пункти. Слід зауважити, що транспортна система в Європі передбачає, що між будь-якою парою пунктів буде існувати як мінімум одна дорога (про її направленість не повідомляється). Траса повинна починатися з будь-якого пункту і має бути прокладена через всі N пунктів, а також має слідувати $N-1$ дорогами (які з'єднують відповідні N пунктів).

Формат вхідних даних: у першому рядку вхідного файлу записано два числа N ($1 \le N \le 100$) та M ($1 \le M \le 10000$). У наступних M рядках записано по два числа a_i та b_i ($1 \le a_i, b_i \le N$), що задають дорогу, яка виходить з пункту a_i й слідує до пун-

кту b_i. Номери пунктів — натуральні числа, що не перевищують N. Причому, можуть існувати дороги, які виходять і входять в один і той самий пункт, а також два пункти, можуть бути з'єднані більше ніж однією дорогою. Однак, гарантується, що для будь-яких двох різних пунктів з номерами i та j існує дорога, яка виходить з пункту i, слідує до пункту j, або ж навпаки — виходить з пункту j та слідує до пункту i.

Формат вихідних даних: у випадку існування такої траси у єдиному рядку вихідного файлу має бути записано N чисел — номери пунктів, по яких буде проходити траса, у порядку її слідування, відокремлених рівно одним пробілом. Якщо існує декілька варіантів прокладки траси — виведіть будь-яку. Якщо варіантів прокладання траси не існує, то в єдиному рядку вихідного файлу виведіть рядок «NO SOLUTION», без лапок.

Приклади вхідних та вихідних даних

tour.in	tour.out
4 6 1 2 2 3 1 4 4 3 4 2 1 3	1 4 2 3

Розв'язання

Припустимо, що ми вже проклали трасу, через деякі k пунктів: $p_1,p_2,\ldots,p_k$. Нехай x — один із пунктів, що не входить до неї. Спробуємо певним чином перебудувати трасу, так щоб вона включала даний пункт. Нехай існує дорога $x \to p_1$, тоді траса $x,p_1,p_2,\ldots,p_k$ буде задовольняти нашим вимогам, в іншому випадку, якщо є дорога $x \to p_2$, то траса $p_1,x,p_2,p_3,\ldots,p_k$ — шукана (так як не має дороги $x \to p_1$, то з умови існує дорога $p_1 \to x$). Отже, на i-ому ($2 \le i \le k$) кроці даних міркувань ми точно знаємо, що існує дорога $p_{i-1} \to x$, а отже перевіряючи існування дороги $x \to p_i$, ми або переходимо на наступний крок, або ж будуємо трасу: $p_1,p_2,\ldots,p_{i-1},x,p_i,p_{i+1},\ldots,p_k$. Зрештою, якщо не існує дороги $x \to p_k$, то точно існує $p_k \to x$ і траса $p_1,p_2,\ldots,p_k,x$ буде шуканою.

Почнемо з траси, що містить лише один пункт. Крок за кроком, додаючи по одній вершині, отримаємо трасу, що проходить через всі N пунктів. Складність розв'язку $O(N^2)$. Уважний читач мав помітити, що трасу можна побудувати для будь-якого графу, що задовольняє умови задачі.

```
#include <iostream>
#include <cstdio>
#include <sstream>
#include <vector>
#include <set>
#include <map>
#include <queue>
#include <algorithm>
#include <numeric>
#include <functional>
#include <string>
#include <cstdlib>
#include <cmath>
#include <list>
using namespace std;
#define FOR(i,a,b) for(int i=(a),_b(b);i<_b;++i)
#define FORD(i,a,b) for(int i=(a),_b(b);i>=_b;—i)
#define REP(i,n) FOR(i,0,n)
#define ALL(a) (a).begin(),a.end()
#define SORT(a) sort(ALL(a))
#define UNIQUE(a) SORT(a),
        (a).resize(unique(ALL(a))-a.begin())
#define SZ(a) ((int) a.size())
#define pb push_back
#define VAR(a,b) __typeof(b) a=(b)
#define FORE(it,a) for(VAR(it,(a).begin());
        it!=(a).end();it++)
#define X first
#define Y second
#define DEBUG(x) cout << #x << " = " << x << endl;
#define INF 1000000000
#define CHECK
typedef vector<int> VI;
typedef vector< vector<int> > VVI;
typedef pair<int, int> PII;
typedef vector<PII> VPII;
typedef long long ll;
int main() {
  freopen("tour.in", "r", stdin);
    freopen("tour.out", "w", stdout);
    int n, m;
    cin >> n >> m;
#ifdef CHECK
    assert (1 <= n && n <= 100);
    assert (1 <= m && m <= 10000);
#endif
    vector <vector <bool> > a (n, vector <bool> (n));
    int q, w;
    REP (i, m) {
        cin >> q >> w;
#ifdef CHECK
        assert (1 <= q && q <= n);
        assert (1 <= w && w <= n);
#endif
        --q, --w;
        a[q][w] = true;
    }
#ifdef CHECK
    REP (i, n)
        REP (j, i)
            assert (a[i][j] + a[j][i] > 0);
#endif
    VI b;
    VI c;
    REP (i, n)
          c.pb (i);
    random_shuffle (ALL (c));
    REP (h, n) {
        int i = c[h];
      bool ok = false;
      REP (j, SZ (b))
        if (a[i][b[j]]) {
          b.insert (b.begin() + j, i);
          ok = true; break;
          }
        if (!ok)
          b.pb (i);
    }
    REP (i, SZ (b)) {
      if (i)
          cout << ' ';
      cout << (b[i]+1);
    }
    cout << endl;
  return 0;
}
```