Palabras

El profesor Oak tiene dos palabras de longitud n, a_1 y a_2. Ambas palabras están formada por letras de la a a la z minúsculas. Algunas de las letras no se pueden leer y el profesor las ha sustituido por un *?*.

El profesor se pregunta si es posible sustituir los *?* por letras minúsculas para que la palabra a_1 sea estrictamente más pequeña lexicográficamente que a_2.

Input Format
La entrada consiste en varios casos. Cada caso empieza con un entero n seguido de las palabras a_1 y a_2.

Constraints
La suma de todas las n es menor que 10000000.

Output Format
Para cada caso escribid una línea con la respuesta 'si' si se puede y 'no' en caso contrario.

Solución en Python

```python
import sys

for line in sys.stdin:
    a = input()
    b = input()

    a = a.replace('?', 'a')
    b = b.replace('?', 'z')

    if (a < b): print("si")
    else: print("no")
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

```cpp
#include <iostream>

using namespace std;

int main() {
	int n;
	while (cin >> n) {
		string s,t;
		cin >> s >> t;
		for (int i = 0; i < n; ++i) {
			if (s[i] == '?') s[i] = 'a';
			if (t[i] == '?') t[i] = 'z';
```

```
            }
            if (s < t) cout << "si" << endl;
            else cout << "no" << endl;
        }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Islas y Montañas

En el país de Enelogonia tienen un problema, cada año sube el nivel del mar. Esto ha hecho que el país quede dividido en islas. El gobierno quiere saber el número de islas en las que estará dividido el país al final de cada año.

Inicialmente el país es una recta dividida en n segmentos, cada segmento tiene una altura de ai metros. Al inicio del primer año el nivel del mar está a 0, a finales del año i-ésimo el nivel está a m_i. Se cumple que si $i < j$ entonces $m_i < m_j$.

Dadas las n alturas de los segmentos y los niveles del mar al final de cada año, decid para cada final de año cuántas islas hay.

Cuando el segmento i-ésimo pasa a estar bajo el agua los segmentos $(i+1)$-ésimo y $(i-1)$-ésimo pasan a formar parte de islas distintas porque están separados por agua (esto en caso de que ambos segmentos estén por encima del nivel del mar).

Un segmento pasa a estar bajo el agua cuando el nivel del mar es igual o mayor a su altura.

Input Format

La entrada consiste en varios casos. Cada caso empieza con una línea con los enteros n y k, en la segunda línea hay los n enteros a_i y en la tercera los k enteros m_i.

Constraints

$1 \leq a_i, m_i \leq 1000000$

Caso 1: 30 puntos
$1 \leq n \leq 1000$
$k = 1$

Caso 2: 30 puntos
$1 \leq n \leq 1000$
$1 \leq k \leq 1000$

Caso 3: 40 puntos
$1 \leq n \leq 100000$
$1 \leq k \leq 100000$

Output Format

Para cada caso escribid una línea con k enteros separados por un espacio, donde el entero i-ésimo es el número de islas a finales del año i-ésimo.

Solución en Python

```python
import sys

for line in sys.stdin:
    n, k = line.split(' ')
    n, k = int(n), int(k)

    alturas = input().split(' ')
    for i in range(n): alturas[i] = (int(alturas[i]), i)

    queries = input().split(' ')
    for i in range(k): queries[i] = int(queries[i])

    alturas = sorted(alturas)

    D = [1 for i in range(n)]

    illes = 1
    it = 0
    ans = []
    for j in range(k):
        while (it < n and alturas[it][0] <= queries[j]):
                i = alturas[it][1]
                it += 1

                sm = 0
                D[i] = 0
                if (i > 0): sm += D[i-1]
                if (i < n-1): sm += D[i+1]

                if (sm == 2): illes += 1
                elif (sm == 0): illes -= 1

        ans.append(illes);

    for i in range(len(ans)):
        if (i != 0): print(end=' ')
        print(ans[i], end='')

    print('')
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef vector<int> vi;
```

```
typedef pair<int, int> pi;
typedef vector<pi> vpi;

int main() {
    int n, k;
    while (cin >> n >> k) {
        vi  M(k);
        vpi A(n);
        for (int i = 0; i < n; ++i) {
               cin >> A[i].first;
               A[i].second = i;
        }
        sort(A.begin(), A.end());
        for (auto& m : M) cin >> m;
        vi D(n, 1);
        int res = 1;
        int ind = 0;
        for (int j = 0; j < k; ++j) {
               while (ind < n and A[ind].first <= M[j]) {
                      int i = A[ind++].second;
                      int sum = 0;
                      D[i] = 0;
                      if (i > 0) sum += D[i-1];
                      if (i < n-1) sum += D[i+1];
                      if (sum == 2) ++res;
                      else if (sum == 0) --res;
               }
               if (j) cout << ' ';
               cout << res;
        }
        cout << endl;
    }
}
```

Autor de la solución: Cesc Folch (Miembro del comité organizador)

Horarios

Baq es un estudiante muy aplicado que se encuentra ante la difícil tarea de escoger las asignaturas para el próximo año. Como está muy interesado en aprender matemáticas quiere elegir el máximo número de asignaturas posible. Además, no le gustaría tener que saltarse ninguna clase, así que no quiere que se le solapen asignaturas (aunque no tiene problema en hacer diversas asignaturas consecutivamente). Las clases deben empezar y acabar en horas en punto, pero como Baq vive en un universo un tanto extraño los días no tienen necesariamente 24 horas.

Conociendo la hora a la que empieza y acaba cada asignatura, ¿podéis determinar el número máximo de asignaturas que va a poder hacer Baq?

Input Format
La entrada consiste en varios casos. Cada caso empieza con n, el número de asignaturas ofertadas, y m, el número de horas en un día. Siguen n líneas con dos enteros a_i b_i, con las horas de inicio y final de la i-ésima asignatura. Se satisface $0 \leq a_i < b_i \leq m$.

Constraints
A (5 puntos): Casos en los que n = 2, 1 ≤ m ≤ 10.
B (5 puntos): Casos en los que 0 ≤ n ≤ 10, m = 2.
C (15 puntos): Casos en los que 0 ≤ n ≤ 10, 1 ≤ m ≤ 10.
D (30 puntos): Casos en los que 0 ≤ n ≤ 1000, 1 ≤ m ≤ 1000.
E (45 puntos): Casos en los que $0 \leq n \leq 5 \cdot 10^5$, $1 \leq m \leq 5 \cdot 10^5$.

Output Format
Para cada caso, imprimid una línea con el número máximo de asignaturas que se pueden cursar.

Solución en Python

```
import sys

for line in sys.stdin:
    n, m = line.split(' ')
    n, m = int(n), int(m)

    D = [0 for i in range(m+1)]
    A = [-1 for i in range(m+1)]

    for i in range(n):
        x, y = input().split(' ')
        x, y = int(x), int(y)

        A[y] = max(x, A[y])

    for i in range(1, m+1):
        s = 0
```

```
        if (A[i] != -1): s = D[A[i]] + 1

        D[i] = max(D[i-1], s)

    print(D[m])
```

Autor de la solución: Javier López-Contreras (Miembro del comité organizador)

Solución en C++

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

typedef vector<int> vi;

int main() {
    int n, m;
    while (cin >> n >> m) {
        vi D(m + 1, 0);
        vi A(m + 1, -1);
        for (int i = 0; i < n; ++i) {
               int x, y;
               cin >> x >> y;
               A[y] = max(x, A[y]);
        }
        for (int i = 1; i <= m; ++i)
               D[i] = max(D[i-1], (A[i] == -1 ? 0 : D[A[i]] + 1));
        cout << D[m] << endl;
    }
}
```

Autor de la solución: Martí Oller (Miembro del comité organizador)

Pickle Rick

Tras una noche de incesante consumo de sustancias estupefacientes, Pickle Rick debe llegar hasta su aeronave lo antes posible para abandonar el planeta, puesto que su insolencia le ha hecho ganarse media docena de peligrosos enemigos. Pickle Rick es un pepinillo, y podemos considerar que su forma es la de un cuboide de dimensiones *1x1x2*. Se encuentra de pie sobre la casilla *(0,0)* de una cuadrícula infinita de casillas *1x1*. Cuando decimos de pie, nos referimos a que reposa sobre cualquiera de sus dos lados cuadrados de dimensiones *1x1*.

Su aeronave se encuentra en la posición *(x,y)* de la cuadrícula y quiere llegar hasta ella de pie y en el menor número de movimientos. Para moverse, Pickle Rick elige una de sus aristas en contacto con el suelo y rota alrededor de ésta hasta chocar de nuevo contra el suelo. Por ejemplo, desde su posición inicial puede rotar alrededor de la arista de la derecha hasta tumbarse estirado sobre las casillas *(1,0)* y *(2,0)* y rotar de nuevo alrededor de la arista de la derecha para ponerse de pie sobre la casilla *(3,0)*. Esto es un total de dos movimientos.

Debido a su estado de embriaguez, Pickle Rick no razona nítidamente. Ayúdalo. Debes calcular el mínimo número de movimientos que debe hacer para llegar de pie a la casilla en la que se encuentra su aeronave.

Input Format

La entrada empieza por un entero *t*, el número de casos. Siguen *t* líneas, una para cada caso, con las coordenadas *x* e *y* de la casilla de la aeronave.

Constraints

$1 \leq t \leq 10^5$
10 puntos: $0 \leq x,y < 1000$ y ambos son múltiplos de *3*
20 puntos: $0 \leq x,y < 10$
20 puntos: $0 \leq x,y < 100$
40 puntos: $0 \leq x,y < 1000$
10 puntos: $0 \leq x,y < 10^9$

Output Format

Para cada caso, imprime una línea con un único entero: el mínimo número de movimientos que debe hacer Pickle Rick para llegar de pie a su aeronave.

Solución en C++

```cpp
#include <iostream>
#include <vector>
#include <queue>

using namespace std;
```

```
const int INF = 1e8;
vector <vector <int> >dist;
priority_queue <pair <int, pair <int,int> > >q;

void move2(int x, int y, int d) {
  if (x < 0 || x >= 1030 || y < 0 || y >= 1030 || dist[x][y] <= d) return;
  dist[x][y] = d;
  q.push({-d, {x, y}});
}

void move(int curx, int cury, int d1, int d2, int d) {
  move2(curx+d1, cury+d2, d);
  move2(curx+d1, cury-d2, d);
  move2(curx-d1, cury+d2, d);
  move2(curx-d1, cury-d2, d);
  move2(curx+d2, cury+d1, d);
  move2(curx+d2, cury-d1, d);
  move2(curx-d2, cury+d1, d);
  move2(curx-d2, cury-d1, d);
}

int main() {
  ios::sync_with_stdio(0);
  cin.tie(0);
  dist = vector <vector <int> >(1030, vector <int>(1030, INF));
  q.push({0, {15, 15}});
  dist[15][15] = 0;
  while (!q.empty()) {
       pair <int, pair <int,int> >p = q.top();
       q.pop();
       p.first = -p.first;

       move(p.second.first, p.second.second, 0, 2, p.first+4);
       move(p.second.first, p.second.second, 0, 3, p.first+2);
       move(p.second.first, p.second.second, 1, 3, p.first+3);
       move(p.second.first, p.second.second, 0, 1, p.first+3);
       move(p.second.first, p.second.second, 2, 3, p.first+4);
  }
  int t;
  cin >> t;
  while (t--) {
       int x, y;
       cin >> x >> y;
       if (x < 1000 && y < 1000) cout << dist[x+15][y+15] << '\n';
       else {
       int ans = 2e9+50;
       for (int i = x%3; i < 30; i += 3) {
       for (int j = y%3; j < 30; j += 3) {
       ans = min(ans, dist[i][j]+2*((x+y+30-i-j)/3));
       }
       }
       cout << ans << '\n';
       }
  }
```

```
}
```

Autor de la solución: Jordi Castellví (Miembro del comité organizador)