FES ACATLAN UNAM

GUAPA

Grupo Universitario de Algoritmia y Programación Competitiva

Compendio de Algoritmia

Autor:
Silverio Flores Moroni

Ciclo:
105 A.C. - 2015 D.C

March 11, 2018

Contents

1 Estructura de Datos **2**
1.1 Estructura de datos lineales 2
1.2 Estructura de datos no lineales 15

2 Teoría de Números **19**

3 Combinatoria **20**

4 Teoría de Gráficas **23**

5 Paradigmas de programación **34**
5.1 Greedy 34

6 Concurso de programación **38**

7 Cuarto concurso de programación **69**

8 concurso de programación **91**

9 5to concurso Interno **102**

10 concurso de programación **120**

1 Estructura de Datos

1.1 Estructura de datos lineales

—D - Dark Souls*

—Límite de tiempo: 1 segundo

Dificultad: $\star$
Temas: Implementación
Complejidad: $O(nm)$

¿Alguna vez has jugado Dark Souls? ¡Se dice que es uno de los juegos mas difíciles que hay! Sus "boss ghts" son casi imposibles, normalmente suele ser un enemigo enorme con muchos puntos de vida, pero cuando aparecen enemigos mas chicos, aparecen en mayor cantidad e incluso puede ser mas difícil que un solo enemigo enorme. Un ejemplo de esto es cuando estlos Dragones y te atacan muchos al mismo tiempo. Cuando pasa esto tienes que buscar un punto ciego donde el ataque de los dragones no te alcance. En esta batalla hay dos tipos de dragones: dragones de fuego y dragones eléctricos. La forma de atacar de los dragones se ilustra en las siguientes cuadriculas.

Los ataques se hacen en la dirección hacia donde los dragones estan volteando (arriba, abajo, izquierda o derecha).
Tu trabajo es, dada una cuadricula de N $\times$ M, la cantidad de dragones de fuego y electricos y sus posiciones, encontrar si existe una casilla que no es alcanzada por los ataques. No se puede estar en la misma casilla en la que esta un dragón.

Entrada

-Un entero T ≤ 100, el número de casos.
-Dos enteros $2 \leq N, M \leq 30$, la dimensión de la cuadrícula.
-Dos enteros F y E, la cantidad de dragones de fuego y eléctronicos respectivamente.
Después siguen F líneas y E líneas con las siguientes especificaciones: Dos enteros, X y Y y un carácter C, donde X y Y son las coordenadas del dragón: Arriba, Abajo, Izquierda y Derecha, representados por los caracteres U, D, L, R, respectivamente.

Salida

Por cada caso de entrada se debe imprimir "Pelear" si existe una casilla que no es

*Hernán Tellechea Garduño - Grupo de Algorimia Avanzada y Programación Competitiva

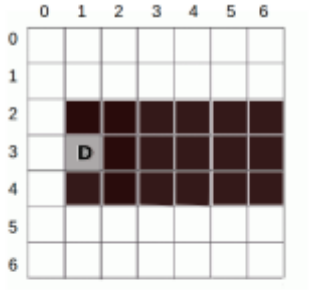


Figura 1: Dragon de fuego

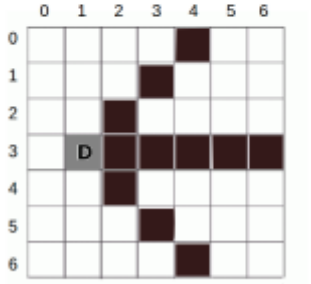


Figura 2: Dragon eléctrico

alcanzada por los ataques o imprimir "Escapar" en el caso contrario.

Entrada Ejemplo

2
9 9
3 0
0 1 D
8 4 u
0 7 D
3 4
1 2
0 1 D
0 0 R
1 0 R

Salida Ejemplo

Escapar
Pelear

Entendiendo el problema

En este problema te encuentras en una cuadrícula de $n \times m$ $(2 \leq n, m \leq 30)$, y se te pide saber si necesitas escapar, o no, después de que ciertas casillas de la cuadrícula sean atacadas por los dragones de la cuadrícula. Siempre que existe al menos una casilla en la cuadrícula que no sea atacada por ningún dragón, no es necesario escapar.

Solución

Es posible marcar todas las casillas que ataca cada dragón. Al terminar con todos los dragones se debe recorrer toda la cuadrícula y revisar si existe al menos una casilla no atacada.

Código

```
#include <iostream>
#include <cstring>
#define For(i, a, b) for(int i=a; i<b; ++i)
#define FOR(i, a, b) for(int i=a; i>b; --i)
using namespace std;

bool mundo[35][35];
int N, M;

void fuego(int x, int y, char d)
{
    if (d == 'U')
        FOR(i, y, -1)
        {
            mundo[i][x] = 1;
            if (x-1 >= 0) mundo[i][x-1] = 1;
            if (x+1 < M) mundo[i][x+1] = 1;
        }
    else if (d == 'D')
        For(i, y, N)
        {
            mundo[i][x] = 1;
            if (x-1 >= 0) mundo[i][x-1] = 1;
            if (x+1 < M) mundo[i][x+1] = 1;
        }
    else if (d == 'L')
        FOR(i, x, -1)
        {
            mundo[y][i] = 1;
            if (y-1 >= 0) mundo[y-1][i] = 1;
            if (y+1 < N) mundo[y+1][i] = 1;
        }
    else
```

```
34          For(i, x, M)
35          {
36              mundo[y][i] = 1;
37              if (y-1 >= 0) mundo[y-1][i] = 1;
38              if (y+1 < N) mundo[y+1][i] = 1;
39          }
40  }
41
42  void electrico(int x, int y, char d)
43  {
44      mundo[y][x] = 1;
45      if (d == 'U')
46      {
47          FOR(i, y-1, -1) mundo[i][x] = 1;
48          for(int i=y-1, j=x-1; i >= 0 and j >= 0; i--, j--) mundo[i][j] = 1;
49          for(int i=y-1, j=x+1; i >= 0 and j < M; i--, j++) mundo[i][j] = 1;
50      }
51      if (d == 'D')
52      {
53          For(i, y+1, N) mundo[i][x] = 1;
54          for(int i=y+1, j=x-1; i < N and j >= 0; i++, j--) mundo[i][j] = 1;
55          for(int i=y+1, j=x+1; i < N and j < M; i++, j++) mundo[i][j] = 1;
56      }
57      if (d == 'L')
58      {
59          FOR(i, x-1, -1) mundo[y][i] = 1;
60          for(int i=y-1, j=x-1; i >= 0 and j >= 0; i--, j--) mundo[i][j] = 1;
61          for(int i=y+1, j=x-1; i < N and j >= 0; i++, j--) mundo[i][j] = 1;
62      }
63      if (d == 'R')
64      {
65          For(i, x+1, M) mundo[y][i] = 1;
66          for(int i=y-1, j=x+1; i >= 0 and j < M; i--, j++) mundo[i][j] = 1;
67          for(int i=y+1, j=x+1; i < N and j < M; i++, j++) mundo[i][j] = 1;
68      }
69  }
70
71  int main()
72  {
73      int T;
74      cin>>T;
75      while(T--)
76      {
77          int F, E;
78          cin>>N>>M>>F>>E;
79
80          For(i, 0, N)
81              memset(mundo[i], 0, sizeof(mundo[i]));
82
```

```
83          For(i, 0, F)
84          {
85              char d;
86              int x, y;
87              cin>>y>>x>>d;
88              fuego(x, y, d);
89          }
90          For(i, 0, E)
91          {
92              char d;
93              int x, y;
94              cin>>y>>x>>d;
95              electrico(x, y, d);
96          }
97          bool escapar = false;
98          For(i, 0, N)
99              For(j, 0, M)
100                 if (!mundo[i][j])
101                     escapar = true;
102
103         if (!escapar)
104             cout<<"Escapar"<<endl;
105         else
106             cout<<"Pelear"<<endl;
107     }
108     return 0;
109 }
```

—E - Escribiendo mensajes[†]

—Límite de tiempo: 3 segundos

Dificultad: ⋆
Temas: Implementación
Complejidad: $O(n)$

El flojo fue víctima de un asalto, le robaron su celular y ahora tiene un viejo celular que tiene un teclado como el siguiente:

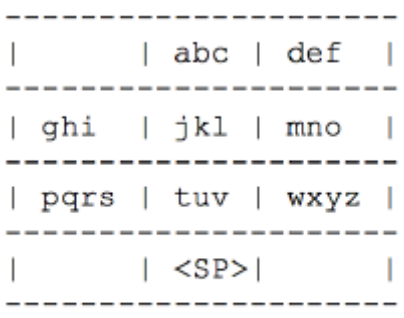


Figura 1: Teclado

Al flojo Mau no le gusta ese celular porque para escribir una letra tiene que presionar varias veces una misma tecla. Por ejemplo, para escribir la "a", él tiene que presionar la misma tecla una vez, pero para escribir la "b" tiene que presionar la misma tecla dos veces, y para la "c", tres veces. Para escribir un espacio, basta con presionar una vez la tecla $< SP >$. El fl ojo Mau tiene que mandar un mensaje de texto con su celular, pero como es muy flojo no quiere mandar el mensaje si debe presionar muchas teclas. Tu tarea es ayudar a contar el número de teclas que el ojo Mau debe presionar para poder escribir un mensaje.

Entrada

Un entero T, el número de casos de prueba. Las siguiente T lńeas contienen sólo espacios y caracteres en minúscula.

Salida

Para cada caso de prueba debes imprimir una línea especificando el número del caso seguido del número de teclas que el ojo Mau debe presionar para escribir el mensaje de ese caso.

[†]UVA Online Judge

Entrada Ejemplo
2
el flojo mau es muy flojo
come frutas y verduras

Salida Ejemplo
Case #1: 52
Case #2: 48

Entendiendo el problema
Dada una cadena de caracteres, se te pide encontrar el número de tecleos necesarios para escribir dicha cadena usando un teclado multitap.

Solución
Lo único que necesitamos es tener el número de tecleos, $T[x]$, necesarios para escribir el caracter x. Con esto podemos recorrer la cadena s e ir sumando $T[s[i]]$ a nuestro resultado.

Código

```
#include <iostream>
#include <cstdio>
#define For(i, a, b) for(int i=a; i<b; ++i)
using namespace std;

int main()
{
    int T[26];
    for (int i=0, v=0; i<26; i++)
    {
        if (i+'a' == 's' or i+'a' == 'z')
            T[i] = 4;
        else
        {
            T[i] = v+1;
            v = (v+1)%3;
        }
    }

    int casos;
    string basura;
    cin>>casos;
    getline(cin, basura);
    For(j, 0, casos)
```

```
25      {
26          int total = 0;
27          string L = "";
28          getline(cin, L);
29          For(i, 0, L.length())
30              if (L[i] == ' ') total++;
31              else total += T[L[i]-'a'];
32          cout<<"Case #"<<j+1<<": "<<total<<endl;
33      }
34      return 0;
35  }
```

—F - ¿Y el problema F?[‡]

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆
Temas: Cadenas, Implementación, Timewaster, Ordenamiento
Complejidad: $O(n \log n)$

Se tiene una enorme cadena de caracteres la cual está codificada y tiene ciertas contraseñas escondidas, dependiendo del tipo y cantidad de contraseñas que contenga la cadena original se le asigna un valor, sabemos que podemos dividir dicha cadena en N subcadenas que son propuestas como útiles, estas N subcadenas pueden caer en uno de los siguientes 4 tipos de cadenas: cadenas palíndromos regulares, cadenas espejo, cadenas palíndromos espejo y cadenas basura, cada una de estas cadenas tiene un significado especial y un valor :

1. Palíndromos Regulares-Valor 1: Cadena de caracteres que se lee igual hacia atr'as que hacia adelante. Ejemplo: "RECONOCER".
2. Cadena Espejo - Valor 2: Cadena de caracteres que, cuando se lee de atrás hacia adelante, cada caracter tiene su respectivo "caracter espejo". Ejemplo: "3SI2E".
   Aquí se muestra la tabla de caracteres con sus correspondientes espejos, no todos los caracteres tiene un espejo.
3. Cadena Palíndromo Espejo - Valor 3: Existen caracteres que son su mismo espejo, por ejemplo el espejo de "A" es "A" y el espejo de "8" es "8", una cadena palíndromo espejo es un palíndromo compuesto únicamente por caracteres que son su mismo espejo. Ejemplo: "A8OMO8A".
4. Cadena Basura - Valor 0: Toda cadena que no cumple con alguna de las condiciones anteriores.

Ĺas cadenas palíndromo espejo tengan son las mas especiales debido a que los investigadores estan seguros de que son contraseñas de cuentas bancarias codificadas, sabiendo esto, podemos obtener el valor de la cadena orginal, que es la suma de los

[‡]Mauricio Eduardo Montalvo Guzmán - Grupo de Algorimia Avanzada y Programación Competitiva

Caracter	Espejo	Caracter	Espejo	Caracter	Espejo
A	A	M	M	Y	Y
B		N		Z	5
C		O	O	1	1
D		P		2	S
E	3	Q		3	E
F		R		4	
G		S	2	5	Z
H	H	T	T	6	
I	I	U	U	7	
J	L	V	V	8	8
K		W	W	9	
L	J	X	X		

Figura 1: Tabla de caracteres

valores de sus subcadenas. Tu tarea es, dado un grupo de M cadenas obtener cual es la que tiene mayor valor.

Entrada

La primer lÍnea de entrada sera un número entero N, que indica el numero de casos de prueba. Cada caso de prueba comienza con dos enteros M, $(1 \leq M \leq 10)$ y P, $(1 \leq P \leq 50)$, que indican el número de cadenas a evaluar y el número de subcadenas que contiene cada cadena, respectivamente. Cada cadena a evaluar esta compuesta por dos líneas, la primera contiene la cadena y la segunda contiene P enteros que son los ndices donde termina cada subcadena, la subcadena 1 comienza en el índice 0. La longitud de cada cadena no sobrepasa los 10000 caracteres.

Salida

Para cada caso de prueba debes imprimir una línea especificando el número del caso seguido del número de teclas que el ojo Mau debe presionar para escribir el mensaje de ese caso.

Entrada Ejemplo

1
2 3
NOTAPALINDROMEISAPALINILAPASIMIRRORED
13 28 36
2A3MEASATOYOTACOSAS
6 13 18

Salida Ejemplo

1 5
2A3MEAS
ATOYOTA

Entendiendo el problema

Se te es dada una serie de cadenas y cada una de ellas está delimitada en subcadenas. Debes decidir cuál cadena es la cadena que tiene el mayor puntaje total; donde el puntaje total de cada cadena es la suma del puntaje de sus subacadenas, los puntajes para cada tipo de subacadena se definen en el problema. Una vez obtenida la cadena con mayor puntaje se deben imprimir, en orden lexicográfico, sus subcadenas que tienen un puntaje mayor a cero.

Solución

El problema anterior es trivial de resolver manteniendo un vector de vectores con las subcadenas útiles para cada cadena y al final del procesamiento se obtendría cuál es la que tuvo el mayor valor total de manera lineal para después solo ordenar sus subcadenas útiles, por lo que solo nos quedaría programar subrutinas que nos digan si una cadena es palíndroma regular (una cadena s es palíndroma si $s_i = s_{n-i-1}$ $\forall\ i = 0, \ldots, n-1$), es espejo (una cadena s es espejo si $s_i = \text{mirror}(s_{n-i-1})$ $\forall\ i = 0 \ldots n-1$, donde, mirror(c) es una función que regresa el caracter espejo de c) o ambas.

Por lo tanto la complejidad del problema sería $O(nm \log n)$, donde, n es la cantidad de cadenas y m es la longitud de la subcadena más grande de la cadena con el mayor valor.

Código

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <map>
#define For(i, a, b) for(int i=a; i<b; ++i)
using namespace std;

bool isPalindromo(string A)
{
        int m1 = A.length()/2, m2 = A.length()/2;
        if (!(A.length()%2)) m1--;

        for(int i = m1, j = m2; i >= 0 and j < A.length(); i--, j++)
                if (A[i] != A[j]) return false;
        return true;
```

```
16  }
17
18  bool isEspejo(string A)
19  {
20          char letra[] = {'A', 'E', 'H', 'I', 'J', 'L', 'M', 'O', 'S', 'T', 'U',
             ↪  'V', 'W', 'X', 'Y', 'Z', '1', '2', '3', '5', '8'};
21          char espej[] = {'A', '3', 'H', 'I', 'L', 'J', 'M', 'O', '2', 'T', 'U',
             ↪  'V', 'W', 'X', 'Y', '5', '1', 'S', 'E', 'Z', '8'};
22          map <char, char> letraEsp;
23          For(i, 0, 21)
24                  letraEsp[letra[i]] = espej[i];
25
26          int m1 = A.length()/2, m2 = A.length()/2;
27          if (!(A.length()%2)) m1--;
28
29          for(int i = m1, j = m2; i >= 0 and j < A.length(); i--, j++)
30                  if (A[j] != letraEsp[A[i]]) return false;
31          return true;
32  }
33
34  bool isPalEsp(string A)
35  {
36          char letra[] = {'A', 'H', 'I', 'M', 'O', 'T', 'U', 'V', 'W', 'X', 'Y',
             ↪  '1', '8'};
37          bool sirve[26];
38          For(i, 0, 13)
39                  sirve[letra[i]-'A'] = true;
40
41          int m1 = A.length()/2, m2 = A.length()/2;
42          if (!(A.length()%2)) m1--;
43          for(int i = m1, j = m2; i >= 0 and j < A.length(); i--, j++)
44                  if (A[i] != A[j] or (A[i] == A[j] and !sirve[A[i]-'A'])) return
                     ↪  false;
45          return true;
46  }
47
48  int main()
49  {
50          int T;
51          cin>>T;
52          while(T--)
53          {
54                  int M, P, valor[15] = {0}, nEsp[15] = {0};
55                  vector <vector <string> > utiles;
56                  string palabra[15];
57                  cin>>M>>P;
58                  utiles.assign(M, vector<string> ());
59                  For(i, 0, M)
60                  {
```

```
61                      int indice[60];
62                      cin>>palabra[i];
63                      For(k, 0, P)
64                              cin>>indice[k];
65                      int pos = 0;
66                      For(k, 0, P)
67                      {
68                              string A = "";
69                              For(j, pos, indice[k]+1)
70                                      A = A + palabra[i][j];
71                              //cout<<A<<endl;
72                              pos = indice[k]+1;
73                              if (isPalEsp(A))
74                              {
75                                      valor[i] += 3;
76                                      utiles[i].push_back(A);
77                                      nEsp[i]++;
78                              }
79                              else if (isEspejo(A))
80                              {
81                                      valor[i] += 2;
82                                      utiles[i].push_back(A);
83                              }
84                              else if (isPalindromo(A))
85                              {
86                                      valor[i] += 1;
87                                      utiles[i].push_back(A);
88                              }
89                      }
90              }
91              int mayor = 0;
92              For(i, 0, M)
93                      if (valor[i] > valor[mayor])
94                              mayor = i;
95              sort(utiles[mayor].begin(), utiles[mayor].end());
96              cout<<palabra[mayor]<<endl;
97              cout<<nEsp[mayor]<<" "<<valor[mayor]<<endl;
98              For(j, 0, utiles[mayor].size())
99                      cout<<utiles[mayor][j]<<endl;
100     }
101     return 0;
102 }
```

1.2 Estructura de datos no lineales

—H - Haciendo Anagramas[§]

—Límite de tiempo: 4 segundos

Dificultad: ⋆⋆
Temas: Ordenamiento, Cadenas
Complejidad: $O(n^2 * m \log m)$

Muchos aficionados a los crucigramas suelen utilizar anagramas (grupos de palabras con las mismas letras en diferente orden) por ejemplo: AMOR, ROMA, OMAR, MORA, RAMO, ARMO y MARO. Sin embargo, no todas las palabras cumplen con este atributo, ya que no importa como trates de ordenar sus letras no podrías formar otra palabra. Estas palabras se llaman ananagramas, un ejemplo es SEXY. Es obvio que la definición anterior depende del dominio con el que nosotros estemos trabajando, tú podrías pensar que APARCAMIENTO es un ananagrama, pero cualquier médico te podría desmentir fácilmente ya que el te diría que existe METACARPIANO. Un posible dominio podría ser todo el idioma español, pero esto podría acarrear algunos problemas. Nosotros podemos restringir el dominio, por ejemplo, Geografía, en cuyo caso NEPAL se vuelve un anagrama relativo (PANEL no esta en el mismo dominio), pero QUERETARO no lo es ya que puede transformarse a TERRAQUEO. Tú debes escribir un programa que lea un diccionario de un dominio restringido de palabras y determinar todos los posibles ananagramas relativos. Debes notar que las palabras de una sola letra son, ipso facto, ananagramas relativos ya que no se pueden "reorganizar"de ningún otra forma. Nuestro diccionario no contendría más de mil palabras.

Entrada

La entrada consiste en una serie de lineas. Ninguna línea tendra mas de 80 caracteres de largo, pero puede contener cualquier número de palabras. Las palabras consisten de hasta 20 caracteres (mayúsculas y minúsculas) letras y no se cortaran entre lineas. Los espacios pueden aparecer libremente entre las palabras y por lo menos un espacio separa múltiples palabras en la misma línea. Tenga en cuenta que las palabras que contienen las mismas letras pero con diferente capitalización se consideran anagramas entre si, por ejemplo CoNsErVaDorA y cOnvErsAdOra son anagramas. El archivo de entrada termina con una linea consistente del sḿbolo #.

[§]Waterloo Local Contest 2005 September 24 (Richard Krueger)

Salida
La salida consistiría de una serie de lineas. Cada línea consistiría de una sola palabra que sería un ananagrama relativo en el diccionario de entrada. Las palabras deberían ser puestas en orden lexicográfico (sensible a capitalización). Se asegura que siempre habrá por lo menos unananagrama relativo.

Entrada Ejemplo
ladder came tape soon leader acme RIDE lone Dreis peat
ScAlE orb eye Rides dealer NotE derail LaCeS drIed
noel dire Disk mace Rob dries

Salida Ejemplo
Disk
NotE
derail
drIed
eye
ladder
soon

Entendiendo el problema
Un ananagrama es una palabra que no es un anagrama, dado un diccionario debemos decir cuáles no son anagramas entre sí.

Solución
Una primera solución ingenua sería ver si cada palabra contra todas las demás es anagrama relativo y las que no lo son marcarlas para después imprimirlas, esto nos dejaría con el problema de ver si una palabra es anagrama de otra, esto se puede hacer ordenando los caracteres de ambas y si coinciden en todas sus letras estamos ante dos anagramas, esto nos deja ante una solución con complejidad $O(n^2 * m \log m)$ donde n es la cantidad de palabras y m es el tamaño de la palabra más grande, para este concurso este era la complejidad mínima esperada para pasar los casos de prueba del juez.

Un segudo enfoque es ordenar cada cadena e insertarlas en un árbol binario de búsqueda balanceado; para cada palabra checaremos si es que existe dentro de la estructura de datos así evitando la fuerza bruta cuadrática, esta solución tiene una complejidad de $O(n * m * \log(n) * \log(m))$.

Código

```cpp
#include <bits/stdc++.h>
using namespace std;

set<string> ananagrama;
vector<string> arr;

bool compare(string a, int apos, string b, int bpos)
{
    if(apos != bpos)
    {
        transform(a.begin(), a.end(), a.begin(), ::tolower);
        transform(b.begin(), b.end(), b.begin(), ::tolower);
        sort(a.begin(), a.end());
        sort(b.begin(), b.end());
        if(a == b)
            return true;
        else
            return false;
    }
    return false;
}

int main()
{
    string palabra;
    while((cin >> palabra) and palabra != "#")
    {
        arr.push_back(palabra);
    }
    bool bandera=false;
    for(int i=0; i<(int)arr.size(); ++i)
    {
        bandera = false;
        for(int j=0; j<(int)arr.size(); ++j)
        {
            if ((bandera = compare(arr[i],i, arr[j],j)) and bandera)
                break;
        }
        if( not bandera)
            ananagrama.insert(arr[i]);
    }
    for(set<string>::iterator  i=ananagrama.begin(); i!=ananagrama.end(); ++i)
    {
        cout << *i << "\n";
    }
    return 0;
```

```
47 }
```

2 Teoría de Números

3 Combinatoria

—C - Cortando Pizza[¶]

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆
Temas: Matemáticas
Complejidad: $O(1)$

Ricardo dice ser muy inteligente y que su inteligencia y perspicacia le ayudarán en todo. Él dice que si su inteligencia le permite hacer una actividad realizando un esfuerzo físico menor ¿Por que debe de cansarse más de la cuenta? También él dice que si usa su cerebro para realizar un menor esfuerzo no es por holgazanería sino una muestra elegante de su superioridad intelectual. En alguna ocasión Ricardo se preguntó cómo cortar una pizza en siete rebanadas y repartirlas entre sus amigos, para ello el tamaño de las rebanadas puede no ser el mismo. Pensó un poco y lleg o a la conclusión de que pod a obtener las siete rebanadas realizando sólo tres cortes -de orilla a orilla- a la pizza, con un cortador de pizza. Aquí mostramos la forma en que Ricardo cortó aquella pizza:

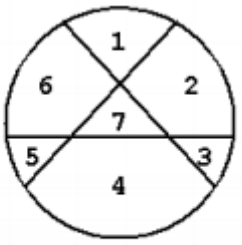


Figura 1: 7 rebanadas

Uno de sus amigos, quien nunca creyó en la inteligencia de Ricardo, pensó "Si Ricardo puede hacerlo, ¿Por qué no podría hacerlo mi computadora?" Así que este amigo intento hacer algo similar (pero no exactamente igual a él porque Ricardo lo criticaría por haberle robado la idea) con ayuda de su computadora. Él escribió un programa que dado el número de cortes en la pizza dice el número máximo de rebanadas que se pueden obtener con exactamente ese número de cortes. Tu trabajo aquí es escribir un programa similar. Es seguro que el amigo de Ricardo no te criticará por hacer el mismo trabajo que él.

[¶]UVA Online Judge

Entrada

El archivo de entrada contiene un entero por línea "....Ecuación... n (0 ¡= n ¡= 4, 295, 113, 503" que representa el número de cortes que se deben hacer de lado a lado de la pizza. Un número negativo con entrada.

Salida

La salida debe ser un entero, el número máximo de rebanadas que se pueden producir.

Entrada Ejemplo

5
10
-100

Entrada Ejemplo

16
56

Entendiendo el problema

El problema se puede ver como el máximo número de regiones en las que el plano puede ser cortado con n líneas, el cual es un problema muy conocido.

Solución

Definamos nuestro resultado como $f(n)$. Claramente $f(1) = 2$, y dado que cada nueva línea nos crea n nuevas secciones, ya que divide cada sección en dos, entonces $f(n) = f(n-1) + n$. Por lo tanto:

$$\begin{aligned} f(n) &= n + f(n-1) \\ &= n + n - 1 + f(n-2) \\ &= n + n - 1 + ... + 2 + f(1) \\ &= n + n - 1 + ... + 2 + 1 + 1 \\ &= \sum_{i=1}^{n} i + 1 \\ &= \frac{n(n+1)}{2} + 1 \end{aligned}$$

Código

```cpp
#include <iostream>
using namespace std;
int main()
{
    long long quesos = 0, n;
    cin >> n;
    while(n >= 0)
    {
        quesos = n*(n+1)/2 + 1;
        cout << quesos << endl;
        cin >> n;
    }
    return 0;
}
```

4 Teoría de Gráficas

—B - Buscando Oro[‖]

—Límite de tiempo: 3 segundos

Dificultad: ⋆⋆⋆⋆
Temas: Grafos, Programación dinámica
Complejidad: $O(k^2 2^k)$

Has descubierto la Ciudad de Oro. Como te encanta el oro, has empezado a recolectarlo. Pero hay tanto oro que te estás cansando mientras lo recolectas, así que quieres saber cuál es el mínimo esfuerzo necesario para obtener todo el oro. La ciudad está descrita con una cuadrícula 2D, donde tu posición inicial está marcada con una 'x'. Un espacio vacío se denotará con un '.'. Y los lugares que contienen oro se denotarán con una 'g'. En cada movimiento puedes desplazarte a todos los 8 lugares adyacentes dentro de la ciudad.

Entrada

La primera línea será un entero T (1 ¡= T ¡= 100), que denota el número de casos. Cada caso empezará con dos enteros, m y n (0 ¡ m; n ¡= 20) denotando los renglones y columnas de la ciudad, respectivamente. Las siguientes m líneas contienen n caracteres que describen las ciudad. Habrá sólo una 'x' en la ciudad y a lo más 15 posiciones de oro.

Salida

Para cada caso de prueba debes imprimir un entero indicando el mínimo número de pasos requeridos para recolectar todo el oro de la ciudad y regresar a la posición inicial.

Entrada Ejemplo

2
5 5
x....
g....
g....

[‖] LightOJ Online Judge

.....
g....
5 5
x....
g....
g....
.....
.....

Salida Ejemplo

Case 1: 8
Case 2: 4

Entendiendo el problema

Dada una cuadrícula de $n \times m$ $(1 \leq n, m \leq 20)$, donde existen k $(1 \leq k \leq 15)$ posiciones especiales, y una posición inicial, se pide encontrar el mínimo número de pasos necesarios para pasar por todas las k posiciones especiales y regresar a la posición inicial.

Solución

Para resolver el problema hay que notar que lo que necesitamos es un ciclo hamiltoniano mínimo. El conjunto de vértices del grafo serán las k posiciones especiales más la posición inicial, y dos vértices u, v, estarán unidos por una arista con peso igual al número mínimo de pasos para llegar de u a v. Debido a que para cada posición de la cuadrícula es posible moverse a los 8 vecinos adyacentes, la distancia mínima entre dos posiciónes es la norma uniforme entre ellas. Una vez construido el grafo hay que encontrar el ciclo hamiltoniano mínimo utilizando programación dinámica.

Código

```
1 #include <iostream>
2 #include <cstring>
3 #include <cstdlib>
4 #include <cstdio>
5 #include <vector>
6 #include <map>
7 #define For(i, a, b) for(int i=a; i<b; ++i)
8 using namespace std;
```

```
9
10  struct punto
11  {
12      int x, y;
13  };
14
15  struct nodo
16  {
17      string S;
18      int pos[20];
19  };
20
21  int tabla[30][35000];
22  nodo I[35000];
23  int dist[30][30], N = 1;
24  map <string, int> Ind;
25
26  int max(int a, int b)
27  {
28      return a > b ? a : b;
29  }
30
31  void getSubStrings(int t, int x, int n, int* pos, string A)
32  {
33      A = A + (char)(x+'0');
34      if (t == n)
35      {
36          Ind[A] = *pos;
37          I[*pos].S = A;
38          For(k, 1, A.length())
39          {
40              string B = A;
41              int u = int(B[k]-'0');
42              B.replace(k, 1, "");
43              I[*pos].pos[u] = Ind[B];
44          }
45          (*pos)++;
46          return;
47      }
48
49      for( ; x < N-1; x++)
50          getSubStrings(t+1, x+1, n, pos, A);
51  }
52
53  void iniTabla()
54  {
55      int pos = 0;
56      For(i, 0, N)
57          getSubStrings(0, 0, i, &pos, "");
```

```
58  }
59
60  void HamCycle()
61  {
62      For(i, 0, N)
63          tabla[i][0] = dist[i][0];
64
65      For(j, 1, 1<<(N-1))
66      {
67          For(i, 0, N)
68          {
69              if (j == (1<<(N-1))-1 and i) return;
70              int min = -1;
71              For(k, 1, I[j].S.length())
72              {
73                  int u = int(I[j].S[k]-'0'), S_u = I[j].pos[u];
74                  if (tabla[u][S_u] + dist[u][i] < min or min == -1)
75                      min = tabla[u][S_u] + dist[u][i];
76              }
77              tabla[i][j] = min;
78          }
79      }
80  }
81
82  int main()
83  {
84      //freopen("entrada.in","r",stdin);
85      punto pos[20];
86      string mundo[20];
87      int T;
88      cin>>T;
89      For(casos, 0, T)
90      {
91          int R, C;
92          cin>>R>>C;
93          For(i, 0, R)
94              cin>>mundo[i];
95          For(i, 0, R)
96              For(j, 0, C)
97                  if (mundo[i][j] == 'x')
98                  {
99                      pos[0].x = j;
100                     pos[0].y = i;
101                 }
102                 else if (mundo[i][j] == 'g')
103                 {
104                     pos[N].x = j;
105                     pos[N].y = i;
106                     N++;
```

```
107                 }
108         For(i, 0, N)
109             For(j, i+1, N)
110                 dist[i][j] = dist[j][i] = max(abs(pos[i].x - pos[j].x),
                     ↪  abs(pos[i].y - pos[j].y));
111 
112         iniTabla();
113         HamCycle();
114         cout<<"Case "<<casos+1<<": "<<tabla[0][(1<<(N-1))-1]<<endl;
115         Ind.clear();
116         N = 1;
117     }
118     return 0;
119 }
```

—G - Guiando a Mau por el Bosque**

—Límite de tiempo: 3 segundos

Dificultad: ⋆⋆⋆⋆
Temas: Teoría de Grafos, Programación Dinámica
Complejidad: $O(E + V \log V)$

El flojo Mau se estresa mucho en su trabajo. Para relajarse después de un día difícil a él le gusta caminar a casa. Para hacer las cosas aún mejores, su ofcina está en un lado del bosque y su casa del otro. Una caminata tranquila por el bosque viendo los pájaros y las ardillas es muy agradable. El bosque es hermoso y el fl ojo Mau quiere tomar una ruta diferente cada día. Él también quiere llegar a su casa antes del anochecer, por lo que siempre toma un camino para avanzar hacia su casa. Él toma un camino de A a B si existe una ruta desde B a su casa que es más corta que cualquier ruta posible desde A. Calcula cuántas rutas diferentes puede tomar el fl ojo Mau desde su oficina a la casa.

Entrada

La entrada contiene varios casos de prueba seguidos por una línea que contiene un 0. El fl ojo Mau ha numerado cada intersección de diferentes caminos, empezando con el 1. Su oficina está numerada con el 1 y su casa con el 2. La primera línea de cada caso contiene el n umero de intersecciones $N(1 < N >= 10^3)$, y el número de caminos M. Las siguientes M líneas contienen, cada una, un par de intersecciones y un entero d (1 ¡= d ¡= 1000) indicando que hay un camino de longitud d entre dichas intersecciones. El fl ojo Mau puede atravezar un camino en cualquier dirección. Hay a lo más un camino entre cada par de intersecciones.

Salida

Para cada caso de prueba debes imprimir un sólo entero módulo 1000000009 indicando el número de rutas diferentes que el fl ojo Mau puede tomar.

Entrada Ejemplo

5 6
1 3 2
1 4 2

**Waterloo Local Contest 2005 September 24 (Richard Krueger)

```
3 4 3
1 5 12
4 2 34
5 2 24
7 8
1 3 1
1 4 1
3 7 1
7 4 1
7 5 1
6 7 1
5 2 1
6 2 1
0
```

Salida Ejemplo

```
2
4
```

Entendiendo el problema

Dado un grafo, encontrar la cantidad de rutas distintas que hay entre el origen s y el destino t con la restricción de que sólo se puede ir de un nodo u a un nodo v si la distancia mínima de v a t es menor que la distancia mínima de u a t.

Solución

Se puede observar que el grafo generado por sólo seguir dichas aristas es un grafo dirigido acíclico (DAG por sus siglas en inglés), que se puede generar fácilmente sacando todas las distancias desde t a todos los demás nodos con un algoritmo Dijkstra en $O(E + V \log V)$ y después con un pase a la lista de adyacencia formamos nuestro DAG.

Sobre un DAG la respuesta al número de caminos posibles desde un origen es bien conocido y se obtiene vistando los nodos en un orden topológico (DFS o BFS para obtener el orden $O(V + E)$) de manera que cuando visitemos un nodo ya habremos visitado todos los nodos que nos llevan a él usando la siguiente recurrencia:

$$f(s,u) = \begin{cases} 1 & u = s \\ \sum_{e=(v,u)\in E} f(s,v) & \text{cualquier otro caso} \end{cases}$$

done la línea de abajo nos dice que sumemos todas las aristas que llegan a u desde v, es decir, la cantidad de caminos que llegan a un nodo es la suma de la cantidad

de caminos que llegan él.

Por último puesto que la respuesta puede ser muy grande usamos aritmética modular para presentar la solución.

Código

```
#include <bits/stdc++.h>
using namespace std;
//look at my code my code is amazing
#define FOR(i, a, b) for (int i = int(a); i < int(b); i++)
#define FOREACH(it, a) for (typeof(a.begin()) it = (a).begin(); it != (a).end();
 ↪  it++)
#define ROF(i, a, b) for (int i = int(a); i >= int(b); i--)
#define REP(i, a) for (int i = 0; i < int(a); i++)
#define INF 1000000000
#define INFLL 1000000000000000000LL
#define ALL(x) x.begin(), x.end()
#define MP(a, b) make_pair((a), (b))
#define X first
#define Y second
#define EPS 1e-9
#define DEBUG(x)   cerr << #x << ": " << x << " "
#define DEBUGLN(x) cerr << #x << ": " << x << " \n"
typedef pair<int, int> ii;
typedef vector<int> vi;
typedef long long ll;
typedef vector<bool> vb;
//ios_base::sync_with_stdio(0);//fast entrada/salida ;-)
//cin.tie(NULL); cout.tie(NULL);
int V, E;
vector< vector<pair<ll,ll> > > AdjList;
vector< vector<ll> > daglist;
vi dist, paths;
vb visited;
vi topo;

void toposort(ll u)
{
        visited[u] = true;
        REP(i, daglist[u].size())
        {
                ll v = daglist[u][i];
                if(not visited[v])
                {
                        toposort(v);
                }
```

```
40          }
41          topo.push_back(u);
42  }
43
44  void solve()
45  {
46          AdjList.assign(V, vector<pair<ll,ll> >());
47          daglist.assign(V, vector<ll>());
48          visited.assign(V, false);
49          topo.assign(0, 0);
50          REP(i, E)
51          {
52                  ll u, v, w;
53                  cin >> u >> v >> w;
54                  u--, v--;
55                  AdjList[u].push_back(pair<ll,ll> (v, w));
56                  AdjList[v].push_back(pair<ll,ll> (u, w));
57          }
58          set<ii> cola;
59          dist.assign(V, INF);
60          paths.assign(V, 0);
61          paths[0] = 1;
62          dist[1] = 0;
63          cola.insert(ii(0, 1));
64          while(not cola.empty())
65          {
66                  ll u = (*(cola.begin())).Y;
67                  ll d = (*(cola.begin())).X;
68                  cola.erase(cola.begin());
69                  if(d > dist[u])
70                          continue;
71                  REP(i, AdjList[u].size())
72                  {
73                          ll v = AdjList[u][i].X;
74                          ll w = AdjList[u][i].Y;
75                          if(d+w <= dist[v])
76                          {
77                                  dist[v] = d+w;
78                                  cola.insert(ii(dist[v], v));
79                          }
80                  }
81          }
82          REP(u, AdjList.size())
83          {
84                  REP(i, AdjList[u].size())
85                  {
86                          int v = AdjList[u][i].X;
87                          if(u != v and dist[v] < dist[u])
88                          {
```

```
89                                  daglist[u].push_back(v);
90                          }
91                  }
92          }
93          toposort(0);
94          ROF(i, topo.size()-1, 0)
95          {
96                  ll u = topo[i];
97                  REP(j, daglist[u].size())
98                  {
99                          ll v = daglist[u][j];
100                         paths[v] = (paths[v] + paths[u]) % 1000000009LL;
101                 }
102         }
103         cout << paths[1] << '\n';
104 }
105
106 int main()
107 {
108         ios_base::sync_with_stdio(0);//fast entrada/salida ;-)
109         cin.tie(NULL); cout.tie(NULL);
110         while(cin >> V and V)
111         {
112                 cin >> E;
113                 solve();
114         }
115         return 0;
116 }
```

5 Paradigmas de programación

5.1 Greedy

—A - Abejas[††]

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆
Temas: Geometría, Ordenamiento, Algoritmos Voraces
Complejidad: $O(n \log n)$

Era un hermoso viernes al medio día en la FES Acatlán, los jóvenes estudiantes descansaban tranquilamente en los pastos y las bancas de las áreas comunes de la FES, cuando de pronto, un cumulo de abejas comenzó a atacar a las personas, las abejas salán de todos lados y picaban a cualquier persona que se pusiera en su camino, todos comenzaron a correr tratando de huir de las abejas pero para algunos estudiantes les fue imposible escapar, extraño ¿no? Despues de varios días de investigación, los integrantes del Grupo de Algoritmia resolvieron el gran misterio acerca del evento de las abejas, un estudiante de MAC, llamado Peter, fue el culpable del acontecimiento. Peter era un chico muy extraño y callado, descubrieron que tená un odio irracional hacia las personas y por eso aventó un panal de abejas aquel dá. A pesar de todo, Peter era un chico muy inteligente y no hacía las cosas al azar, descubrieron que, basandose en sus conocimientos de matemáticas y computación, logro desarrollar un sistema perfectamente elaborado para molestar a la gente. Peter veía a la FES como una gran cuadrćula (similar al plano cartesiano) donde podía identicar la posición de cualquier objeto con sus coordenadas en X y Y , así es como logró identificar las posiciones de todos los panales de abejas que habá en la FES. Ya que tena identificados los panales sólo esperaba a que hubiera una hora pico donde cierto número N de personas se encontraran cerca de un panal, en ese momento comenzaba su proceso para molestar. Peter sabá que si golpeba el panal con cierta fuerza F las abejas saldrían muy enojadas y picarían a todo aquel que se encontrara en un radio R alrededor del panal, muchas veces era imposible poder afectar las N personas cerca del panal asque Peter elegá un nÚumero K, (K $\leq$ N), de personas a las que al menos querá afectar. Jámas se logró descubrir como es que Peter calculaba el nivel de fuerza F con el que era necesario golpear un panal. La historia de Peter

[††]Mauricio Eduardo Montalvo Guzmán - Grupo de Algorimia Avanzada y Programación Competitiva

es muy interesante, tan interesante que tú estás obsesionado con poder emular el proceso que tená Peter para molestar a la gente, así que desarrollarás un programa al que dadas las coordenadas de un panal, el número de personas cerca del panal y el número de personas a las que Peter quería molestar, calcule el area mínima de la zona afectada por las abejas para poder molestar al menos al número de personas que Peter eligió.

Entrada

La primer línea de entrada sería un entero C, $(1 < C \leq 500)$ que indica el número de casos de prueba. Cada caso de prueba comienza con una línea que contiene dos números flotantes PX y PY , $(-500 \leq PX, PY \leq 500)$ que indican las coordenadas del panal, la siguiente línea contiene dos enteros N y K, $(1 < K \leq N < 200)$ que indican el número de personas cerca del panal y el número de personas que Peter quiere molestar respectivamente, de ahí siguen N líneas donde cada una contiene dos números flotantes X_i, Y_i, $(-500 \leq X_i; Y_i \leq 500)$ que indican las coordenadas de la persona i, cada caso de prueba termina con un .

Salida

Por cada caso de entrada se debe imprimir "Pelear" si existe una casilla que no es alcanzada por los ataques o imprimir "Escapar" en el caso contrario.

Entrada Ejemplo

2
9 9
3 0
0 1 D
8 4 u
0 7 D
3 4
1 2
0 1 D
0 0 R
1 0 R

Salida Ejemplo

Escapar
Pelear

Entendiendo el problema

Dado un punto base y n puntos adicionales, quieres saber cuál es el área mínima del círculo que cubre al menos k puntos adicionales con centro en el punto base.

Solución

Siguiendo una estrategia *greedy*, o voraz, podemos ver que el círculo que buscamos es aquél que tiene como radio la distancia al k-ésimo punto más cercano al centro.

Código

La dificultad recae en implementar un algoritmo de ordenamiento cuya complejidad sea $O(n \log n)$ y ordenar por distancias, pero aprovechando las bondades de la `STL` de `C++` tenemos lo siguiente.

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#define For(i, a, b) for(int i=a; i<b; ++i)
#define PI 3.14159265359
using namespace std;

struct punto
{
        double x, y, dist;
};

bool myfunction(punto A, punto B){ return A.dist < B.dist; }

int main()
{
        int T;
        cin>>T;
        while (T--)
        {
                double Cx, Cy;
                int N, K;
                string basura;
                punto puntos[200];
                cin>>Cx>>Cy>>N>>K;
                For(i, 0, N)
                {
                        cin>>puntos[i].x>>puntos[i].y;
                        puntos[i].dist = (Cy-puntos[i].y)*(Cy-puntos[i].y) +
                         ↪  (Cx-puntos[i].x)*(Cx-puntos[i].x);
                }
                cin>>basura;
                sort(puntos, puntos+N, myfunction);
```

```
33                  printf("%.2lf\n",PI*puntos[K-1].dist);
34          }
35          return 0;
36  }
```

6 Concurso de programación

XII semana de MAC

—A - Asombroso League of Legends[‡‡]

—Límite de tiempo: 3 segundos

Dificultad: $\star$
Temas: Ad-hoc
Complejidad: $O(1)$

League of Legends es un juego clasificación MOBA (Multiplayer Online Battle Arena), en el que combaten varios campeones en enfrentamientos épicos, en los que el equipo que juegue mejor en equipo gana. En éste problema nos centraremos en 2 campeones, Lux y Volibear; Volibear está persiguiendo a Lux, la cuál tiene poca vida, tal que si Volibear alcanza a Lux, ésta morirá sin poder hacer nada al respecto. Afort unadamente, Lux tiene un hechizo que le permite hacer que Volibear deje de moverse durante cierto tiempo(T); el hechizo lo puede usar cada C segundos (a C también se le llama "enfriamiento" de hechizo). Tu tarea consiste en, sabiendo las velocidades de los campeones (Lux y Volibear), el tiempo que el hechizo de Lux deja atrapado a Volibear, y el enfriamiento del hechizo de Lux, determinar si Lux será alcanzada por Volibear o no.

Entrada

La primera línea contendrá un n umero N ($0 < N \leq 100$), siendo N el número de casos de prueba. Cada una de las siguientes N líneas tendrá 4 enteros, V1, V2, T y C, ($0 < V1, V2 \leq 500$)($0 < T, C <= 10$) las velocidades de Volibear, Lux, el tiempo que el hechizo de Lux atrapa a Volibear, y el enfriamiento de su hechizo respectivamente. (Las velocidades V1 y V2 están dadas en unidades/segundo, y T,C están dados en segundos. También supon- dremos que Volibear y Lux empiezan en puntos diferentes, Lux est a adelante de Volibear, y ambos corren en línea recta)

Salida

Para cada caso de prueba, se tendrá que imprimir una línea, imprimiendo "Se muere" si Volibear alcanza a Lux, y "Se salva" en caso contrario.

[‡‡]Sergio Adrián Lagunas Pinacho - Grupo de Algorimia Avanzada y Programación Competitiva

Entrada Ejemplo

2
10 300 1 2
300 100 1 10

Salida Ejemplo

Se salva
Se muere

Entendiendo el problema

Hay dos objetos que se mueven sobre una recta infinita, cuya posición al inicio es distinta una del otro, y quieres saber si el objeto a puede alcanzar al objeto b en algún momento en el tiempo; sabiendo que, el objeto a se mueve a una velocidad v_a durante c segundos y se mantiene inmóvil durante t segundos, mientras que el objeto b nunca para de moverse a una velocidad constans v_b.

Solución

Si observamos lo que pasa durante un ciclo de tiempo de tamaño $c+t$ podemos ver que el objeto a (el que persigue) solo se moverá $c*v_a$ unidades de distancia mientras que el objeto b se moverá $(c+t)*v_b$ unidades de distancia, por lo tanto si $c*v_a \leq (c+t)*v_b$ con cada ciclo de tiempo se irá alejando o manteniendo la misma distancia a b y en caso contrario b será alcanzado,sin importar donde empiecen.

Código

```
#include <iostream>
#include <cstdio>
using namespace std;

int main()
{
//        freopen("DataLoL.out","w",stdout);
//        freopen("DataLoL.in","r",stdin);
        int n,c1,c2,T,C;
        cin>>n;
        while(n--)
        {
                cin>>c1>>c2>>T>>C;
                c2 *= C;
                c1 = (C - T)*c1;
                if(c1 > c2)
```

```
17                  {
18                          cout<<"Se muere"<<endl;
19                  }
20                  else
21                  {
22                          cout<<"Se salva"<<endl;
23                  }
24          }
25  }
```

—B - Boby EL Minotauro*

—Límite de tiempo: 3 segundos

Dificultad: ⋆
Temas: Ad-hoc, Cadenas
Complejidad: $O(n)$

Desafortunadamente para el "flojo Mau", un día se topó con el malvado minotauro devorador de hombres conocido como ... Boby, y cayó directo en su trampa. Boby no es cualquier minotauro, es un minotauro que habla, pero el sólo entiende frases que son palíndromos. Mau se dió cuenta que si le hablaba con palíndromos a Boby, él lo dejaría ir. Afortunadamente Mau tiene consigo un teléfono inteligente con un programa que identifica frases palindrómicas, el cual tú codificaste para él ... ¿o no?.

Entrada

Te serán dadas muchas frases. Cada frase sólo contendrá letras mayúsculas de la 'A' hasta la 'Z' y los siguientes caracteres: '.', ',', '!', '?'. El final de la entrada será una línea conteniendo la palabra "HECHO", que no deberá ser procesada. Cada frase tendrá a lo más 200 caracteres.

Salida

Para cada frase imprime una línea con la palabra "NO SERAS COMIDO" si la frase es un palíndromo, o "OH NO!" si no es un palíndromo.

Entrada Ejemplo

ROMA TIBI SUBITO MOTIBUS IBIT AMOR.
ME DEJARIAS IR?
ARRIBA LA BIRRA
TRAIGAN AL MINOTAURO!
HECHO

*Modificado de un problema original de UVA Online Judge

Salida Ejemplo

NO SERAS COMIDO
OH NO!
NO SERAS COMIDO
OH NO!

Anotaciones: Un palíndromo es una frase que se lee igual de atrás hacia adelante y de adelante hacia atrás. Tienes que determinar si son palíndromos o no, ignorando signos de puntuación.

Entendiendo el problema

El problema te pide saber si una cadena es un palíndromo.

Solución

Se dice que una cadena es un palíndromo si para elemento s_i de la cadena s se cumple que $s_i = s_{n-i+1}$ donde n es el tamaño de cadena e $i = 0, 1, \ldots, n-1$. Podemos checar esto eliminando los caracteres especiales y haciendo la comprobación lineal caracter por caracter, aunque con hacer la comprobación para una mitad es suficiente.

Código

```
#include <cstdio>
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <algorithm>
#include <string>
#include <set>
#include <cctype>
#include <vector>
#include <map>
#include <cctype>
using namespace std;
typedef long long ll;

string frase ;

void solve()
{
        int n = frase.size();
        string frase_bonita = "";
```

```
21
22          for (int i = 0; i < n; ++i)
23          {
24                  if(not (frase[i] == '.' or frase[i] == ',' or frase[i] == '!' or
                     ↪  frase[i] == '?' or frase[i] == ' ' or frase[i] == '\n') )
25                  {
26                  //if(isalpha(frase[i]))
27                          frase_bonita += frase[i];
28                  }
29          }
30          n = frase_bonita.size();
31          for (int i = 0, j = n-1; i < (n/2)+1; ++i, --j)
32          {
33                  if(frase_bonita[i] != frase_bonita[j])
34                  {
35                          //cout << frase_bonita[i] << " " << frase_bonita[j] <<
                             ↪  '\n';
36                          printf("OH NO!\n");
37                          return;
38                  }
39          }
40          printf("NO SERAS COMIDO\n");
41  }
42
43  int main()
44  {
45      //int k = 1;
46          while(getline(cin, frase) and frase != "HECHO")
47          {
48                  solve();
49          }
50          return 0;
51  }
```

—C - Coleccionista*

—Límite de tiempo: 3 segundos

Dificultad: ⋆
Temas: Implementación
Complejidad: $O(n)$

Hernán es un joven apasionado por los videojuegos, le gustan de todo género, desde shooters, plataformas, carreras y estrategia hasta llegar a los de aventura. Uno de sus juegos favoritos es Pokémon. Pokémon es un juego en el que tu objetivo es conseguir todos los monstruos disponibles(llamados pokémon) y las medallas de todos los gimnasios para así demostrar que has vencido a los mejores.

Aunque a Hernán le encantan todas las versiones de este juego, desde Rojo y Azul hasta X y Y, él no siempre quiere capturar a todos los pokémon disponibles en algunas versiones, ya sea porque considera que algunos son débiles o pequeños, e incluso feos.

Debido a esto(y a que no le gusta hacer algo tan fácil), te ha pedido que le ayudes con una sencilla tarea.

Dados el número total de pokémon disponibles en cada versión y dos listas de pokémon, la primera será la lista de los que ya posee y la segunda la lista de los que no le interesan, determina las siguientes dos cosas:

- Cuántos y cuáles pokémon le faltan por capturar (solo de los que le interesan).
- Cuantos y cuales pokémon que posee puede cambiar, ésto es, los pokémon que ya tiene y no le interesan.

Los pokémon se identificarán por enteros para facilitar su procesamiento.

Es sabido que Hernán no posee mas de una especie del mismo pokémon por versión.

Entrada

La entrada consta de varios casos de prueba. Cada caso consta de 3 líneas, la primera línea ser´a un entero N, ($10 \leq N \leq 1000$) que indica el total de pokémon

*Maximiliano Vera Luna - Grupo de Algoritmia Avanzada y Programación Competitiva

en la versión del juego.

La segunda línea contiene un entero A, ($0 \leq A \leq N$) el número de pokémon que Hernán ha capturado en esa versión, seguido por una lista de A enteros que representan a estos pokémon. Todos estos n´umeros están separados por un espacio.

La tercera línea contiene un entero X, ($0(\leq X \leq N$) el total de pokémon que no le interesan a Hernán, seguido por X enteros que representan a los mismos. Todos estos números están separados por un espacio. Para facilitar su identificación, los pokémon estan representados por números desde 1 hasta N.

Salida

Para cada caso imprime dos líneas:

La primera línea será un entero F que representará el número de pokémon que le interesa capturar a Hernán y aún no tiene, seguido por F enteros que representan estos pokémon impresos en orden ascendente.

La segunda constará de un entero C que representará el número de pokémon que Hernán puede cambiar, seguido por C enteros que serán los pokémon cambiables impresos en orden ascendente.

Entrada Ejemplo

11
5 3 5 9 10 11
6 2 3 4 5 9 11

Salida Ejemplo

4 1 6 7 8
4 3 5 9 11

Entendiendo el problema

Dadas la cantidad de pokémon existentes, los pokémon que ya capturaste y los pokémon que no te interesan tener, debes obtener dos cosas:

- Cuántos y cuáles pokémon que sí te interesan te faltan obtener.
- Cuántos y cuáles pokémon de los que ya tienes no te interesan.

Solución

Para la solución necesitamos poder almacenar cuáles pokémon tenemos, y de esos cuáles no me interesan. Dado que el número total de pokémon es pequeño (≤ 1000), es posible crear arreglos de booleanos, `yatengo[]` y `nomeinteresan[]` para este propósito. Para responder la primer pregunta necesitamos contar los pokémon x tales que `yatengo[x] == false` y `nomeinteresan[x] == false`. Para la segunda, tenemos que contar aquellos pokémon tales que `yatengo[x] == true` y `nomeinteresan[x] == true`.

Código

```
#include <iostream>
#include <vector>
#include <cstdio>
using namespace std;
int main()
{
    int estampas,aux,tengo,desinteres,cont1,cont2,cont=0;;
    vector<int> yatengo;
    vector<int> nomeinteresan;
    while(cin>>estampas)
    {
        cont1=0;
        cont2=0;
        cont++;
        yatengo.assign(estampas+1,0);
        nomeinteresan.assign(estampas+1,0);
        cin>>tengo;
        for(int i=0;i<tengo;i++)
        {
            cin>>aux;
            yatengo[aux]=1;
        }
        cin>>desinteres;
        for(int i=0;i<desinteres;i++)
        {
            cin>>aux;
            nomeinteresan[aux]=1;
        }
        for(int i=1;i<estampas+1;i++)
        {
            if(nomeinteresan[i]==0 && yatengo[i]==0)
            {
                cont1++;
            }
        }
        cout<<cont1;
```

```
37          for(int i=1;i<estampas+1;i++)
38          {
39              if(nomeinteresan[i]==0 && yatengo[i]==0)
40              {
41                  cout<<" "<<i;
42              }
43          }
44          cout<<endl;
45          for(int i=1;i<estampas+1;i++)
46          {
47              if(nomeinteresan[i]==1 && yatengo[i]==1)
48              {
49                  cont2++;
50              }
51          }
52          cout<<cont2;
53          for(int i=1;i<estampas+1;i++)
54          {
55              if(nomeinteresan[i]==1 && yatengo[i]==1)
56              {
57                  cout<<" "<<i;
58              }
59          }
60          cout<<endl;
61      }
62      return 0;
63  }
```

—D - Dominó*

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆
Temas: Búsquedas
Complejidad: $O(n \log n)$

Mauricio tiene N piezas de domino en fila. Cada pieza se divide en dos partes iguales en tamaño, - la superior y la inferior -. Cada una de las partes contiene un número del 1 al 6.
A Mauricio le encantan los números pares, así que quiere que la suma de los números de las mitades superiores y la suma de los números de las mitades inferiores sea par.

Para lograr eso, Mauricio puede rotar las piezas de dominó 180 grados. Después de una rotación las mitades cambian de lugares. Esta acción toma 1 segundo. Ayuda a Mauricio a averiguar el tiempo mínimo que debe gastar girando piezas de dominó para que se cumpla lo que él quiere.

Entrada

La primera línea contiene un entero $T(1 < T < 50)$, el número de casos de prueba. Para cada uno de los siguientes T casos, hay una línea que contiene un entero $N(1 \leq N \leq 100)$, el número de piezas de dominó que tiene Mau. Las siguientes N líneas contienen dos enteros $X_i, Y_i, (1 \leq X_i, Y_i \leq 6)$ cada una, separados por un espacio.
El número X_i es el que esta escrito inicialmente en la mitad superior del i-ésimo dominó, mientras que Y_i es el que está escrito en la mitad inferior.

Entrada

Para cada caso imprime sólo una línea, esta deberá decir "Caso T: " seguido por el mínimo de segundos requeridos para realizar la tarea. T es el caso actual empezando a numerarlos desde 1. Si resulta imposible lograr tal tarea, imprime un '-1' (sin comillas).

*Maximiliano Vera Luna - Grupo de Algoritmia Avanzada y Programación Competitiva

Entrada Ejemplo

2
2
4 2
6 4
1
2 3

Salida Ejemplo

Caso 1: 0
Caso 2: -1

Entendiendo el problema

Dado una línea alineada de dominos, debemos verificar si es posible hacer que la suma de las partes inferiores y la suma de las superiores es par, ésto mediante la rotación de las fichas de dominó.

Solución Obtenemos las sumas de ambas partes, podemos dividir estas sumas en tres casos:

1. Si las sumas son pares, el problema esta resuelto y la respuesta es 0.
2. Si una suma es par y la otra impar no existe una solución debido a que cada ficha puede ser o con ambas partes pares o ambas impares o una par y la otra impar, pero al rotar cualquiera de estos tres tipos de fichas se sigue sin obtener una solución.
3. Si ambos sumas son impares, basta con encontrar una ficha con una parte par y la otra impar, al rotarla se obtendra una solución, si tal ficha no existe entonces no hay solución.

Código

```
#include <iostream>
#include <sstream>
#include <algorithm>
#include <vector>
#include <map>
#include <set>
#include <queue>
#include <list>
#include <stack>
#include <cmath>
#include <cstdlib>
```

```cpp
#include <cstdio>
#include <cstring>
#include <cctype>
#include <climits>
using namespace std;
int izq[105],der[105];
int n,sumi,sumd;

int main()
{
    int t,cont=0;
    cin>>t;
    while(t--)
    {
        cont++;
        cin>>n;
        cout<<"Caso "<<cont<<": ";
        sumi = sumd = 0;
        bool b = false;
        for(int i=0;i<n;i++)
        {
            cin>>izq[i]>>der[i];
            sumi += izq[i];
            sumd += der[i];
        }
        if((sumi&1) and (sumd&1))
        {
            for(int i=0; i<n; i++)
            {
                if( (izq[i]&1) != (der[i]&1))
                {
                    b = true;
                    break;
                }
            }
            if(b)
                cout<<1<<endl;
            else
                cout<<-1<<endl;
        }
        else if( !(sumi&1) and !(sumd&1) )
        {
            cout<<0<<endl;
        }
        else
        {
            cout<<-1<<endl;
        }
    }
```

```
61      return 0;
62  }
```

—E- Esto es fácil*

—Límite de tiempo: 5 segundos

Dificultad: ⋆⋆
Temas: Búsquedas
Complejidad: $O(n \log n)$

Este problema es fácil, tienes que buscar 2 enteros en una lista, tal que sumados den otro número.

Entrada

La primera línea tendrá un número T que representa el número de casos de prueba.

Las siguientes $3T$ líneas contendrán los casos de prueba, cada caso de prueba tendrá 3 líneas, la primera línea tendrá un número $N(2 \leq N \leq 10000)$ que representa la cantidad de números que tendrá el arreglo, la segunda línea tendrá N números enteros $K_i(1 \leq i \leq N)(0 \leq |K_i| \leq 10000)$ los números sobre los cuáles tienes que buscar la pareja de números, y la tercera línea tendrá un número entero $M(0 \leq |M| \leq 20000)$.

Salida

Se imprimirán T líneas, una por cada caso de prueba, con la palabra "SI" si es que existen 2 números K_i y K_j tales que $K_i + K_j = M$, y "NO" en caso contrario.

Entrada Ejemplo

```
2
3
1 2 3
4
5
6 7 8 9 10
11
```

*Sergio Adrián Lagunas Pinacho - Grupo de Algoritmia Avanzada y Programación Competitiva

Salida Ejemplo

SI
NO

Entendiendo el problema

Dada una lista con n elementos, debemos determinar si existe un par de elementos de la lista cuya suma sea igual a un número M dado.

Solución

Para cada elemento a_j del arreglo A, mientras no hayamos determinado que existe una solucion, debemos verficar si existe a_k tal que $a_j + a_k = M$, eso lo realizamos mediante una busqueda binaria (debemos ordenar los elementos de A); esto debido a la limitante de tiempo del problema, si para cada elemento $a_j \in A$ buscaramos linealmente a su complemento a_k, la complejidad sería cuadrada y obtendríamos como veredicto *Tiempo Limite Excedido*.

Nota: Dado que los elementos de A son relativamente pequeños es posible disminuir la complejidad del problema a $O(n)$, esto marcado en un arreglo en la posición a_i si $a_i \in A$, ahorrandonos la el ordenamiento y la busqueda binaria.

Código

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <algorithm>
#define For(X,A,B) for(X=A; X<B; X++)
#define Maxt 1000
using namespace std;

int arr[1000]={0};
int var;

void busca(int a,int b,int valor)
{
    if(a == b && arr[b]==valor)
        {var = 1; return;}
    else if(a == b && arr[b]!=valor)
        {var = 0; return;}

    int mitad = (a+b)>>1;

    if(valor <= arr[mitad])
        busca(a,mitad,valor);
    else
```

```
24          busca(mitad+1,b,valor);
25  }
26
27  int main(int argc, char const *argv[])
28  {
29      //freopen("Data.in","r",stdin);
30      //freopen("Data.out","w",stdout);
31      int i,j,n,casos,sum,tam,x;
32
33      scanf("%d",&casos);
34      while(casos--)
35      {
36          scanf("%d",&tam);
37
38          for(i=0; i<tam; i++)
39              scanf("%d",&arr[i]);
40
41          scanf("%d",&sum);
42
43          //Cosas
44          var = 0;
45          sort(arr,arr+tam);
46          For(j,0,tam)
47          {
48              x = sum - arr[j];
49              if(x>0)
50                  busca(0,tam-1,x);
51
52              if(var)
53              {
54                  printf("SI\n");
55                  break;
56              }
57          }
58          if(var == 0)
59              printf("NO\n");
60      }
61      return 0;
62  }
```

—F- Falsa Simulación*

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆
Temas: Ad-Hoc, Estructuras de Datos
Complejidad: $O(n+m)$ construcción, $O(1)$ por consulta.

El siguiente problema consta de una simulación muy concreta... ¿o no?. Tendrán una matriz M de tamaño 1234(filas) $\times$ 5678(columnas). Está inicialmente llena con enteros desde el 1, 2, . . . , 1234×5678 en orden ascendente de izquierda a derecha y de arriba a abajo. (La primera fila está llena con los enteros 1, 2, . . . , 5678; la segunda con los enteros 5679, 5680, . . . , 11376, etc) Existen 4 tipos de comandos:

- "R x y" intercambia la x-ésima y y-ésima fila de M.
- "C x y" intercambia la x-ésima y y-ésima columna de M.
- "Q x y" imprime $M(x, y)$ (El entero que está en la posición (x, y) en la matriz M).
- "W z" imprime x e y donde $z = M(x, y)$.

Entrada

Te será dada una lista de comandos válidos, uno por línea. La entrada termina con el fin de archivo.

Salida

Cada que recibas el comando "Q x y" imprime una línea con el valor $M(x, y)$, y cada que recibas el comando "W z" imprime una línea con los valores x e y separados por un espacio, la posición en la que se encuentra el número z.

Entrada Ejemplo

R 1 2
Q 1 1
Q 2 1

*Sphere Online Judge

W 1
W 5679
C 1 2
Q 1 1
Q 2 1
W 1
W 5679
7

Salida Ejemplo

5679
1
2 1
1 1
5680
2
2 2
1 2

Entendiendo el problema

Para el propósito de entender el problema supongamos que se tiene una matriz de n filas por m columnas de la siguiente manera, suponiendo que $n = m = 5$:

$$\begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \\ r_4 \\ r_5 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{pmatrix} \end{array}$$

Si quisieramos efectuar la operación `R 1 2` obtendriamos lo siguiente:

$$\begin{array}{c} \\ r_2 \\ r_1 \\ r_3 \\ r_4 \\ r_5 \end{array} \begin{array}{c} \begin{array}{ccccc} c_1 & c_2 & c_3 & c_4 & c_5 \end{array} \\ \begin{pmatrix} 6 & 7 & 8 & 9 & 10 \\ 1 & 2 & 3 & 4 & 5 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{pmatrix} \end{array}$$

de donde la respuesta para `W 9` sería `1 4`.

Solución

La solución ingenua consiste en mantener una matriz e ir simulando cada uno de las consultas en $O(n)$ u $O(m)$, pero se puede ver en primera instancia que esta solución no es factible para nuestras restricciones ya que ni siquiera pasaría las restricciones de memoria.

Una solución que nos permite hacer consultas y modificaciones a la matriz en $O(1)$, se logra mediante el uso de dos vectores por renglones y columna uno que te relacione el índice original y índice actual, y visceversa.

Sólo queda saber que para obtener los índices de una matriz de la forma original dado el valor de z debemos hacer: $x = \lfloor z/m \rfloor + 1$, $y = (z \mod m) + 1$; y para obtener z conociendo x y y hacemos: $z = (x-1)m + y$.

Código

```
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <vector>
using namespace std;
typedef vector<int> vi;

#define N 1234
#define M 5678

vi rows(N+3, 0);
vi cols(M+3, 0);
vi rr = rows;
vi cc = cols;

void solveR()
{
        int x, y;
        scanf("%d %d", &x, &y);
        swap( rows[x], rows[y] );
        rr[rows[x]] = x;
        rr[rows[y]] = y;
}

void solveC()
{
```

```
27          int x, y;
28          scanf("%d %d", &x, &y);
29          swap( cols[x], cols[y] );
30          cc[cols[x]] = x;
31          cc[cols[y]] = y;
32  }
33
34  void solveQ()
35  {
36          int x, y;
37          scanf("%d %d", &x, &y);
38          printf("%d\n", ((rows[x]-1) * M) + cols[y] );
39  }
40
41  void solveW()
42  {
43          int z;
44          cin >> z;
45          --z;
46          printf("%d %d\n", rr[(z / M)+1], cc[(z%M)+1]);
47  }
48
49  int main()
50  {
51          for (int i = 0; i < M+3; ++i)
52          {
53                  cc[i] = cols[i] = i;
54          }
55          for (int i = 0; i < N+3; ++i)
56          {
57                  rr[i] = rows[i] = i;
58          }
59          char orden;
60          while(cin >> orden)
61          {
62                  switch(orden)
63                  {
64                          case 'R':
65                                  solveR();
66                                  break;
67                          case 'C':
68                                  solveC();
69                                  break;
70                          case 'Q':
71                                  solveQ();
72                                  break;
73                          case 'W':
74                                  solveW();
75                                  break;
```

```
76                  }
77          }
78          return 0;
79  }
```

—G- Geométra Hermann*

—Límite de tiempo: 3 segundos

Dificultad: ⋆
Temas: Matemáticas
Complejidad: $O(1)$

Durante el siglo XIX el matemático Hermann Minkowski investigó acerca de un tipo de geometría no euclidiana, llamada geometría de taxi. En la geometría de taxi la distacia entre dos puntos $T_1(x_1, y_1)$ y $T_2(x_2, y_2)$ es definida como $D(T_1, T_2) = |x_1 - x_2| + |y_1 - y_2|$.
Cualquier otra definición se define igual que en la geometría euclidiana, incluyendo la definición del círculo: Un círculo es el conjunto de todos los puntos en un plano a una distancia dada(radio) de un punto dado(centro del círculo).

Nosotros estamos interesados en la diferencia de las áreas de dos círculos con radio R, uno de los cúales está dado en un espacio normal(euclideano) y el otro en una geometría de taxi.
Dicha tarea ha sido encomendada a tí.

Entrada
Cada línea tendrá un solo número entero $1 \leq R \leq 10000$. Deberás leer hasta el final del archivo.

Salida
Para cada caso deberás imprimir dos líneas, la primera contendrá el area del círculo con radio R en una geometría euclidiana y la segunda línea será el area del círculo con radio R en una geometría de taxi.

Tu salida deberá ser redondeada a 4 lugares decimales.

Además puedes asumir de manera segura de usando valores flotantes de precisión doble y π igual 3.141592653589793 será suficiente.

Entrada Ejemplo
21

*Edgar García Rodríguez - Grupo de Algoritmia Avanzada y Programación Competitiva

9384
887

Salida Ejemplo

1385.4424
882.0000
276646940.0487
176118912.0000
2471707.7105
1573538.0000

Entendiendo el problema

Debes encontrar el área de un círuclo con radio r utilizando una distancia euclideana y un círculo con radio r utilizando una distancia Manhattan.

Solución

El círculo en una geometría con la distancia euclideana es trivial. Para el otro caso hay que observar que un círculo en una geometría con la distancia Manhattan es en realidad un cuadrado rotado de lado $\sqrt{2r^2}$, por lo tanto su área es $2r^2$.

Código

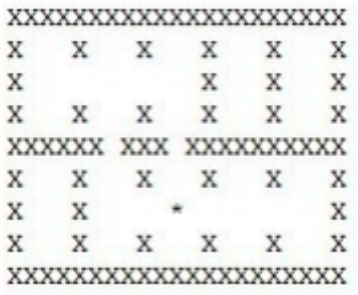
```
#include <stdio.h>
#include <math.h>

int main()
{
    double PI = atan(1.0)*4.0;
    double R;
    while(scanf("%lf", &R)==1)
    {
        printf("%.4lf\n", (double)PI*R*R);
        printf("%.4lf\n", (double)R*R*2.0);
    }
    return 0;
}
```

—I- Investigando Laberinto*

—Límite de tiempo: 3 segundos

Dificultad: ⋆⋆
Temas: Flood fill
Complejidad: $O(nm)$

Un laberinto es representado en una cuadrídula de dos dimensiones como está ilustrado en la figura siguiente. Cada punto de la cuadrícula es representado por



un carácter. Un carácter espacio ( ) representa los lugares en donde puedes caminar.

Las paredes del laberinto son representadas por letras mayúsculas (de A a Z), esto es, lugares dónde no puedes caminar.

Tu tarea es, dado un punto de inicio dentro del laberinto, representado por un asterisco (*), debes de marcar todos los posibles lugares a donde puedes llegar caminando con el carácter gato (#). El asterisco debe ser reemplazado por el car´acter gato.

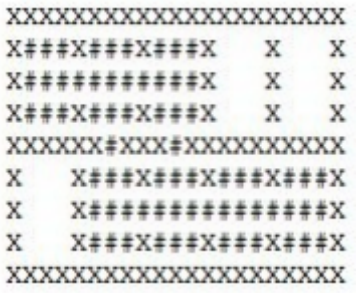
La forma en que puedes caminar es 4 conexidad, esto significa que solo puedes moverte hacia arriba, abajo, izquierda o derecha.

La cuadricula presentada anteriormente quedaría:

Entrada

La primera línea de entrada es T, el número de casos. La primera línea de cada caso son dos enteros, N y M, el número de renglones y el número de columnas de

*Maximiliano Vera Luna - Grupo de Algoritmia Avanzada y Programación Competitiva



la cuadrícula respectivamente.

Las siguientes N lineas contienen M carácteres, las cuales representan al laberinto.

Salida

N líneas de M carácteres cada una, las cuales representa a la cuadrícula marcada en la forma en que se indicó.

Imprime una línea en blanco al final de cada caso.

Entrada Ejemplo

```
2
5 5
XXXXX
X* X
X X
XX XX
XXXXX
5 9
AAAAAAAAA
C* D B
C D B
C D B
EEEEEEEEE
```

Salida Ejemplo

```
XXXXX
X###X
```

X###X
XX#XX
XXXXX
AAAAAAAAA
C###D B
C###D B
C###D B
EEEEEEEEE

Entendiendo el problema

Dado una cuadrícula de $n \times m$, con algunas paredes dentro de ella, y un punto inicial, se te pide marcar todos los lugares dentro de la cuadrícula a los que puedes llegar, sabiendo que no puedes atravezar paredes y que sólo te puedes mover hacia arriba, abajo, izquierda o derecha.

Solución

Este es un problema muy común y su solución es conocida como algoritmo de relleno por difusión (flood fill en inglés), que es básicamente una búsqueda.

Código

```
#include<iostream>
#include<string>
#include<vector>
#include<cstdio>
using namespace std;
vector<string> v;
int dfs(int i,int j)
{
    if(i>0 && j>0 && j<v[i].size() && i <v.size() && v[i][j]!='#' && (v[i][j]==' '
     ↪  || v[i][j]=='*'))
    {
        v[i][j]='#';
        dfs(i,j+1);
        dfs(i+1,j);
        dfs(i-1,j);
        dfs(i,j-1);
    }
    return 0;
}
int main()
{
    string aux;
    int t,a,b;
    cin>>t;
```

```
24      while(t--)
25      {
26          cin>>a>>b;//a=filas, b=columnas
27          getline(cin,aux);
28          for(int i=0;i<a;i++)
29          {
30              getline(cin,aux);
31              v.push_back(aux);
32          }
33          for(int k=0;k<v.size();k++)
34          {
35              for(int l=0;l<v[k].size();l++)
36              {
37                  if(v[k][l]=='*')
38                  {
39                      dfs(k,l);
40                  }
41              }
42          }
43          for(int k=0;k<v.size();k++)
44          {
45              cout<<v[k]<<endl;
46          }
47          cout<<endl;
48          v.clear();
49      }
50      return 0;
51  }
```

—H- Historia de los Relojes*

—Límite de tiempo: 3 segundos

Dificultad: ⋆
Temas: Matemáticas
Complejidad: $O(1)$

El interés medieval en contribuciones mecánicas es bien ilustrado por la invención del reloj mecánico, el más viejo el cual es controlado por contrapesos, y un brazo oscilante con un engranage. Esto fué en 1386.

Los relojes controlador por resortes aparecieron por la mitad del siglo 15, haciendo posible construir mecanismos más compactos, y preparando el camino para el reloj portatil.
Los primeros relojes de péndulo controlador por resortes fueron comúnmente puestos en pequeñas repisas de pared, y después sobre repisas.
Muchos relojes de soporte tenían un cajoncito para guardar la llave del vidrio. Los relojes de soporte más recientes, hechos después de 1660, tenían diseño arquitectónico, con pilares a los lados.

El los siglos 17 - 18 en Francia, los relojes de mesa se convirtieron en un objeto de diseño monumental.

Uno de los primeros relojes atómicos fué un reloj controlado por amoniaco . Este fué hecho en 1949 en el "National Bureau of Standards", Washington, D.C.

La historia de los relojes es fascinante, pero no tiene que ver con éste problema. En éste problema te darán el ángulo entre las manecillas de un reloj en forma de círculo, como el que se muestra en la figura, y tienes que indicar si existe una hora del día tal que el ángulo entre ambas manecillas sea el ingresado por el usuario.

Para éste problema, supondremos que el reloj tiene 60 marcas, una para cada minuto, y que tanto la manecilla de los minutos, tanto como la de las horas sólo pueden estar apuntando a alguna de las marcas.

*Sergio Adrián Lagunas Pinacho - Grupo de Algoritmia Avanzada y Programación Competitiva

Entrada
Está dada por varias líneas, cada una describiendo un caso de entrada. Cada linea contiene un entero A que representa el ángulo entre las manecillas ($0 \leq A \leq 180$).

Salida
Para cada caso de entrada imprime una línea con la palabra "Caso i: " (i es el número de caso), seguido por un espacio, seguido por un caracter. Si existe al menos una hora del día tal que el ángulo mínimo entre las manecillas del reloj sea exactamente A grados, imprime el caracter será "Y". En caso contrario será "N".

Entrada Ejemplo
90
65
66
67
128
0
180

Salida Ejemplo
Caso 1: Y
Caso 2: N
Caso 3: Y
Caso 4: N
Caso 5: N
Caso 6: Y
Caso 7: Y

Entendiendo el problema
Dado un ángulo, quieres saber si es posible que las dos manecillas de un reloj con-

vencional, con 60 marcas, pueden crear dicho ángulo.

Solución

Dado que sólo hay 60 marcas en el reloj, los 360 grados estarán divididos en 60 marcas separadas por 6 grados, así que las manecillas podrán formar un ángulo si y sólo si el ángulo es múltiplo de 6.

Código

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <cstdio>
using namespace std;
int main()
{
    int a,cont=0;
    while(cin>>a)
    {
        cont++;
        cout<<"Caso "<<cont<<": ";
        if(a%6)
        {
            cout<<"N\n";
        }
        else
        {
            cout<<"Y\n";
        }
    }
}
```

7 Cuarto concurso de programación

Cuarto concurso Interno de Programación

—A- Annie la Hija de la Oscuridad*

—Límite de tiempo: 3 segundos

Dificultad: ⋆
Temas: Ad-hoc
Complejidad: $O(1)$

League of Legends es un videojuego de género MOBA, desarrollado por Riot Games para Microsoft Windows y OS X.

Los jugadores (invocadores), se agrupan en 2 equipos de campeones (champions, o bien champs), 3 vs 3 o 5 vs 5. A partir de diciembre del 2013 hay 118 campeones disponibles en los servidores normales, pero este número aumenta periódicamente. Cada equipo comienza en lados opuestos de un mapa en un área llamada Base, cerca de lo que se llama Nexo (o Nexus).

El objetivo del juego es destruir el Nexo del equipo rival. Para destruir un nexo, cada equipo debe llegar a la base enemiga eliminando unas torretas que la protegen.

Cada jugador gana niveles al matar súbditos (NPCs que aparecen constantemente y atacan al otro equipo) del equipo contrario y derrotar a monstruos neutrales. Matar monstruos, campeones enemigos y destruir torretas proporciona oro necesario para mejorar al Campeón y facilitar así las batallas.

*Édgar García Rodríguez - Grupo de Algoritmia Avanzada y Programación Competitiva

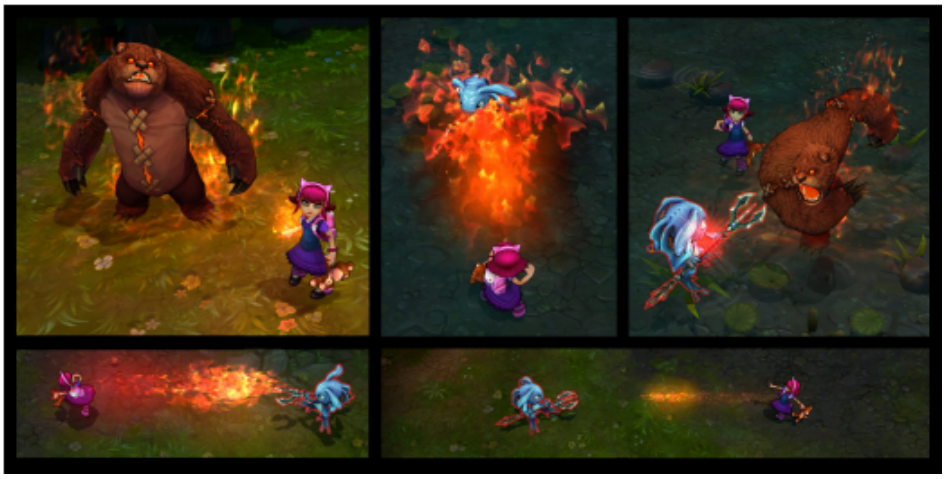


Figura 1: Arriba a la izquierda "Tibbers", arriba al centro "Incinerar", Abajo a la izquierda "Desintegrar"

Uno de estos champs es Annie, tu objetivo es ayudarla a determinar cuando podrá encadenar sus tres hechizos ofensivos a un campeón enemigo. Los hechizos que conoce Annie son los siguientes:

- "Desintegrar" Annie lanza una bola de fuego imbuida de Maná a cualquier campeón en un area de $600u$[1] a su alrededor
- "Incinerar" Annie lanza un abrasador cono de fuego, dañando a todos los enemigos de la zona. Este cono tiene una apertura de 35° y además tiene un alcance de $500u$ (imagine una rebanada de pizza circular, con centro en Annie).

Puedes asumir de manera segura que no hay obstáculos entre Annie y su objetivo; además puedes despreciar el tiempo entre el lanzamiento de cada hechizo.

Entrada

La primer línea de la entrada será un número $2 \leq T \leq 20$ que será el número de casos. Cada caso comenzará con tres números $0 \leq A_x, A_y \leq 10000$ y $1 \leq Q \leq 50$ donde los primeros dos son las coordenadas de Annie y el tercero el número de consultas para ese caso. Las siguientes Q líneas tendrán dos números $0 \leq E_x, E_y \leq 10000$ que es la posición de el campeón objetivo. Se asegura que todos los números en la entrada son enteros.

Salida

Para cada consulta deberas imprimir una línea que dirá “FULL COMBO” si puedes encadenar los 3 hechizos contra el campeón objetivo al mismo tiempo, si no puede, imprime “OUTPLAYED”.

Entrada Ejemplo

2
0 1 1
490 2
23 4 1
700 900

Salida Ejemplo

FULL COMBO
OUTPLAYED

Sugerencias: $c_2 = a_2 + b_2$

Entendiendo el problema

Dadas las posiciones de Annie y de un campeón en el plano, se nos pide saber si Annie es capaz de encadenar los tres hechizos que conoce al campeón enemigo. El problema nos dice que no habrá obstáculos entre ellos y que el lanzamiento de los hechizos es de manera instantánea.

Solución

Lo que necesitamos para poder encadenar los tres hechizos es que la distancia entre Annie y el campeón sea menor o igual a $500u$ ya que de éste modo los tres ataques pueden alcanzar al campeón.

Código

```
#include <bits/stdc++.h>

using namespace std;

#define FOR(i, a, b) for (int i = int(a); i < int(b); i++)
#define REP(i, a) for (int i = 0; i < int(a); i++)
#define INF 1000000000
typedef pair<int, int> ii;
typedef vector<int> vi;
typedef vector<ii> vii;
```

```cpp
typedef long long ll;
typedef vector<bool> vb;

int casos, q;
ii annie, campeon;

int distancia_squared(ii u, ii v)
{
    return (u.first-v.first)*(u.first-v.first) +
     ↪  (u.second-v.second)*(u.second-v.second);
}

void querea()
{
    scanf("%d %d", &campeon.first, &campeon.second);
    int dis_squ = distancia_squared(annie, campeon);
    if( dis_squ <= 250000 )
        printf("FULL COMBO\n");
    else
        printf("OUTPLAYED\n");

}

void solve()
{
    scanf("%d %d", &annie.first, &annie.second);
    scanf("%d", &q);
    REP(i, q)
        querea();
}

int main()
{
    freopen("Annie.in", "r", stdin);
    freopen("Annie.out", "w", stdout);
    scanf("%d", &casos);
    while(casos--)
        solve();
    return 0;
}
```

—B- Colores*

—Límite de tiempo: 1 segundo

Dificultad: $\star$
Temas: Ad-hoc
Complejidad: $O(1)$

Manuel Nicolás tiene éstos códigos de colores:
0 0 0 = Blanco
0 0 1 = Azul
0 1 0 = Rojo
0 1 1 = Morado
1 0 0 = Amarillo
1 0 1 = Verde
1 1 0 = Naranja
1 1 1 = Negro

Pero su amigo Mir Ist Kalt cambió los códigos, tal que los 1's los volvió 0's, y visceversa.

Tu trabajo es ayudar a Manuel Nicolás a decirle a qué color corresponde cada código cambiado.

Entrada

La primera línea contendrá un entero $N(1 \leq N \leq 20)$, el número de casos de prueba. Las siguientes N líneas tendrán 3 dígitos, di_1, di_2, di_3 separados por un espacio $(0 \leq d_{i1}, d_{i2}, d_{i3} \leq 1)(1 \leq i \leq N)$.

Salida

Se tendrán que imprimir N líneas, una por cada caso de prueba, todas terminando con salto de línea; cada línea tendrá el color correspondiente a su código.

*Sergio Adrián Lagunas Pinacho - Grupo de Algoritmia Avanzada y Programación Competitiva

Entrada Ejemplo

4
0 0 0
1 1 1
0 0 1
0 1 1

Salida Ejemplo

Negro
Blanco
Naranja
Amarillo

Entendiendo el problema

Nos dan un diccionario de colores cuyo código está en binario. En el problema se te da el código alterado (intercambiando unos y ceros) de un color y se te pide imprimir el color correspondiente del código original.

Solución

Directa.

Código

```
#include <iostream>

using namespace std;

typedef vector<int> vi;
typedef vector<pair<int,int> > vii;
typedef vector<vi> vvi;
#define rep(a,b) for(int a=0;a<b;a++)
#define For(a,b,c) for(int a=b;a<c;a++)
int main()
{
    std::ios_base::sync_with_stdio(false);
    int n,a,b,c;
    string v[8] =
     ↪  {"Negro","Naranja","Verde","Amarillo","Morado","Rojo","Azul","Blanco"};
    cin>>n;
    rep(i,n){
        cin>>a>>b>>c;
        cout<<v[4*a + 2*b + c]<<endl;
```

```
19      }
20      return 0;
21  }
```

—C - CandyLand*

—Límite de tiempo: 1 segundo

Dificultad: $\star$
Temas: Sliding window
Complejidad: $O(n)$

Mágicamente has sido transportado a la asombrosa tierra de CandyLand, donde hay tantos dulces de tantos sabores como nunca habías imaginado. Siendo un fanático de los dulces no puedes contener la felicidad y empiezas a probar todos los dulces existentes y asignas a cada dulce un valor de sabrosidad.

Sin embargo no todo puede ser tan perfecto, tu tiempo en CandyLand está a punto de acabar y serás transportado de regreso a tu vida cotidiana, pero no quieres irte con las manos vacías.

Enfrente de ti hay una línea con $1 \leq N \leq 106$ dulces numerados del 1 al N, donde al i-ésimo dulce le has asignado un valor de sabrosidad $-100 \leq S_i \leq 100$. Y convenientemente tienes a tu disposición un brazo robótico capaz de agarrar exactamente $1 \leq K \leq 103$, $K \leq N$ dulces consecutivos de dicha línea, y una computadora.

Debido a que hay algunos dulces que no te gustan tanto, quieres agarrar K dulces tales que la suma de su valor de sabrosidad sea el máximo posible. Como la cantidad de dulces es muy grande necesitas hacer un programa que te diga cuál es el índice del primer dulce de la izquierda que debe agarrar el brazo robótico para lograr tu objetivo.

Entrada

La primer línea contendrá el número $1 \leq T \leq 50$ de casos. Para cada caso habrá dos líneas.

La primer línea de cada caso contendrá dos enteros N y K. La segunda línea de cada caso contendrá N enteros Si, $1 \leq i \leq N$.

Salida

Para cada caso deberás imprimir dos enteros en una línea, el índice del primer dulce

*David Felipe Castillo Velázquez - Grupo de Algoritmia Avanzada y Programación Competitiva

de la izquierda que debe agarrar el brazo robótico para lograr tu objetivo, si hay más de una opción imprime el que tenga el menor índice, y la suma máxima del valor de sabrosidad que puedes conseguir.

Nota
El brazo robótico tiene que agarrar exactamente K dulces.

Entrada Ejemplo
1
10 4
1 -5 5 10 -1 3 -2 -3 9 4

Salida Ejemplo
3 17

Entendiendo el problema
El problema se reduce a: dado un arreglo de n enteros, encontrar el subarreglo de tamaño exactamente k, tal que la suma del subarreglo sea máxima.

Solución
La solución $O(nk)$ es demasiado lenta para este problema. Sea `a[]` el arreglo de enteros. Si declaramos un arreglo `v[i]` $:= \sum_{j=1}^{i}$ `a[j]`, entonces la suma del subarreglo de tamaño k empezando en la posición i es: `v[i+k]-v[i-1]`. Con esta observación tenemos una solución $O(n)$.

Código

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

#define For(i, a, b) for(int i=a; i<b; ++i)
#define INF (1<<30)

using namespace std;

int v[1000005];

int main()
{
```

```
15      int T;
16      scanf("%d", &T);
17
18      while (T--)
19      {
20          int N, K;
21          scanf("%d %d", &N, &K);
22
23          int res = -INF, ind = 1;
24          For(i, 1, N+1)
25          {
26              scanf("%d", &v[i]);
27              v[i] += v[i-1];
28          }
29
30          For(i, K, N+1)
31              if (v[i] - v[i-K] > res)
32                  res = v[i]-v[i-K], ind = i-K+1;
33
34          printf("%d %d\n", ind, res);
35      }
36
37      return 0;
38  }
```

—D - Raíces Digitales*

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆
Temas: Ad-hoc
Complejidad: $O(k)$

Recientemente Manuel Nicolás León descubrió qué es la raáz digital, y decidió compartir su conocimiento contigo.
Digamos que $S(n)$ es la suma de los dígitos de n, por ejemplo, $S(4089) = 4 + 0 + 9 + 8 = 21$, entonces la raíz digital del número n es:

1. $rd(n) = S(n) si S(n) < 10$
2. $rd(n) = rd(S(n)) si S(n) \geq 10$

Por ejemplo, $rd(4098) = rd(21) = rd(3) = 3$.

Manuel le tiene miedo a los números grandes, por eso los números con los que trabajará serán a lo mas 10100.

Para todos esos números, Manuel ha probado que
$rd(n) = S(S(S(S(n))))(n \leq 10^{1}00)$.

Ahora Manuel quiere encontrar números rápidamente dada su raiz digital. El problema es que todavía no ha aprendido a hacer lo que te va a preguntar. Tu tarea es, dados los núumeros k y d, encontrar números exactamente de k dígitos (sin ceros al principio), con su raíz digital igual a d.

Entrada

La primera línea tendrá un número $T(T \leq 500)$ que representa el número de casos de prueba.
Las siguientes T líneas contendrán los casos de prueba, cada caso tendrá 2 números, k y $d(1 \leq k \leq 100; 1 \leq d \leq 9)$.

Salida

Se imprimirán $2T$ líneas, 2 por cada caso de prueba: la primera línea de cada caso de prueba tendrá el número n más grande tal que $rd(n) = d$, y el número de dígitos

*Sergio Adrián Lagunas Pinacho - Grupo de Algoritmia Avanzada y Programación Competitiva

de n sea igual a k; la segunda línea de cada caso de prueba tendrá el número n más chico tal que $rd(n) = d$, y el número de dígitos de n sea igual a k.

Puedes tener la seguridad de que dichos números siempre existen, y son únicos. El primer dígito de cada número impreso no tiene que ser un 0.

Entrada Ejemplo

2
1 3
1 7

Salida Ejemplo

3
3
7
7

Entendiendo el problema

En el problema se define $S(n)$ como la suma de los dígitos de n y

$$rd(n) = \begin{cases} S(n) & \text{si } S(n) < 10 \\ rd(S(n)) & \text{si } S(n) \geq 10 \end{cases}$$

Se nos da $1 \leq k \leq 100$ y $1 \leq d \leq 9$, y se nos pide encontrar el número más pequeño min y el número más grande max tales que $rd(min) = rd(max) = d$ y tanto min como max contengan k dígitos.

Solución

Éstos dos números se construyen de la siguiente manera:

$$min = (1)(\underbrace{0 \ \ldots \ 0}_{k-2})(d-1)$$

$$max = (\underbrace{9 \ \ldots \ 9}_{k-1})(d)$$

Código

```
1  import java.util.*;
2  import java.io.*;
3
4  class D
5  {
```

```
6       public static void main(String[] args)
7       {
8           MyScanner sc = new MyScanner();
9           int T = sc.nextInt();
10
11          while (T-- > 0)
12          {
13              int k = sc.nextInt(), d = sc.nextInt();
14
15              for(int i = 0; i < k-1; i++)
16                  System.out.print(9);
17              System.out.println(d);
18
19              if (k > 1)
20                  System.out.print(1);
21              for(int i = 1; i < k-1; i++)
22                  System.out.print(0);
23              if (k > 1)
24                  System.out.println(d-1);
25              else
26                  System.out.println(d);
27          }
28      }
29  };
```

—E - Buscando Asiento*

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆
Temas: Ad-hoc
Complejidad: $O(1)$

Manuel Nicolás León es un chico muy especial... cuando va a conferencias le gusta que la silla de su izquierda y la silla de su derecha estén desocupadas.

Este año irá a una conferencia a la cuál irán sólo chicos igual de especiales que él (que les gusta que las sillas adyacentes a la suya estén desocupadas). Para el problema tienes que decir si es posible que todos los chicos, incluyendo Manuel, estén a gusto, ésto es, que todos los chicos tengan libres las sillas adyacentes a la suya.

Entrada

La primera línea tendrá un número T que representa el número de casos de prueba.

Las siguientes T líneas contendrán los casos de prueba, cada caso tendrá 2 números, A y $B(1 \leq A \leq B \leq 500)$ indicando el número de chicos, y el número de sillas disponibles respectivamente.

Los chicos escogen las sillas uno por uno, escogiendo una silla que tenga sus 2 sillas adyacentes desocupadas (Si es una de las sillas de la orilla, basta con que la única silla adyacente que tiene est´e desocupada).

Tu tarea es, dadas éstas condiciones, decir si los chicos se sentirán a gusto o no.

Salida

Se imprimirán T líneas, una por cada caso de prueba, con alguna de las siguientes palabras: “Si” si siempre es posible que todos los chicos estén a gusto, “No” si es imposible que todos los chicos estén a gusto, o “Tal vez” si depende de cómo se vayan sentando los chicos.

*Sergio Adrián Lagunas Pinacho - Grupo de Algoritmia Avanzada y Programación Competitiva

Entrada Ejemplo

3
1 3
2 3
3 3

Salida Ejemplo

Si
Tal vez
No

Entendiendo el problema

El problema nos dice que hay A niños que se quieren sentar en B asientos, sin embargo cada uno de los niños quiere sentarse en un asiento tal que sus dos vecinos estén desocupados. Los niños van sentándose uno por uno escogiendo cualquier asiento que cumpla con los requerimientos. Debemos ver si siempre es posible que estén agusto independientemente de cómo se vayan sentado (caso "Si"), si depende de los asientos que escojan (caso "Tal vez"), o sin importar que asientos escojan no es posible que estén agustos (caso "No").

Solución

1. Caso "No". Si $A > \left\lceil \frac{B}{2} \right\rceil$. Pues la mejor forma en que se pueden ir sentando es dejando un espacio de separación, si así no es posible sentarlos, entonces no hay forma de que estén agusto.
2. Caso "Si". Si $A \leq \left\lceil \frac{B}{3} \right\rceil$. La "peor" forma en la que se pueden acomodar los niños es si cada uno se coloca a dos asientos de separación de sus vecinos, ya que si se sientan a un asiento de separación sería la forma más "compacta", y si se sientan a más de dos, entonces un tercer niño puede sentarse en medio de ellos. Ésto se vería de la siguiente forma:

   $$...010\underbrace{010}_{3}010...$$

   Por lo tanto se puede apreciar que cada niño "cubre" tres asientos.
3. Caso "Tal vez". Si $\left\lceil \frac{B}{3} \right\rceil < A \leq \left\lceil \frac{B}{2} \right\rceil$

Código

```java
import java.util.Scanner;

class E
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        int T = sc.nextInt();

        while (T-- > 0)
        {
            int n = sc.nextInt(), s = sc.nextInt();

            int limNo = s % 2 == 0 ? s / 2 : s / 2 + 1;
            int limSi = s % 3 == 0 ? s / 3 : s / 3 + 1;

            if (n > limNo)
                System.out.println("No");
            else if (n <= limSi)
                System.out.println("Si");
            else
                System.out.println("Tal vez");
        }
    }
};
```

—F - Fibonacci*

—Límite de tiempo: 1 segundo

Dificultad: $\star$
Temas: Ad-hoc
Complejidad: $O(1)$

Manuel Nicolás León es un chico muy curioso, tanto que durante una de sus clases se le ocurrió una sucesión de dígitos parecida a la de Fibonacci, la cuál empieza con 2 dígitos, 1 y 1, y para la cual cual(es)quiera dígito(s) que se le vaya a añadir es igual a los dígitos que equivalen a la suma de los valores numéricos de los 2 dígitos anteriores, así el tercer dígito sería $1 + 1 = 2$, el cuarto sería $1 + 2 = 3$, el quinto $2 + 3 = 5$, el sexto $3 + 5 = 8$, y así sucesivamente.

Lo interesante de ésta sucesión empieza del dígito 7 en adelante; dado que $5+8 = 13$, los dígitos 7 y 8 serían el 1 y el 3 respectivamente; el dígito que ocuparía la posición 9 sería $1 + 3 = 4$, y así sucesivamente, dando lugar a algo como ésto:

112358134...

Manuel Nicolás León se pregunta si hay alguna forma de saber cualquier dígito de la sucesi´on con ayuda de un programa, para lo cu´al te ha pedido tu ayuda: tienes que programarlo.

Entrada

La primera línea contendrá un entero $N(1 \leq N \leq 20)$, el número de casos de prueba. Las siguientes N líneas tendrán un número entero positivo Si $(1 \leq Si \leq 109)$ $(1 \leq i \leq N)$ que representa el dígito requerido.

Salida

Se tendrán que imprimir N líneas, una por cada caso de prueba, todas terminando con salto de línea; cada línea tendrá un dígito $D_i(0 \leq D_i \leq 9)$ $(1 \leq i \leq N)$ que representa el dígito que está en la posición Si requerido.

Entrada Ejemplo

*Sergio Adrián Lagunas Pinacho - Grupo de Algoritmia Avanzada y Programación Competitiva

6
1
2
3
7
8
9

Salida Ejemplo

1
1
2
1
3
4

Entendiendo el problema

En este caso se nos da una cadena de dígitos F cuya construcción se parece a la sucesión de Fibonacci. La forma de construirla es sumar los dos últimos dígitos de la cadena y concatenar el resultado. Los primeros dígitos de la cadena son los siguientes:

$$\underbrace{1123581347}_{10}112...$$

Y se nos pide saber cuál es el n-ésimo dígito de la cadena F.

Solución

Los dígitos señalados siempre se van a repetir, debido a que se toman los dos últimos dígitos para construir los siguientes. Por lo tanto sólo debemos guardar los diez primeros dígitos y el n-ésimo dígito de F será $F[n\%10]$.

Código

```
#include <bits/stdc++.h>

using namespace std;
typedef vector<int> vi;
typedef vector<pair<int,int> > vii;
typedef vector<vi> vvi;
#define rep(a,b) for(int a=0;a<b;a++)
#define For(a,b,c) for(int a=b;a<c;a++)
```

```cpp
int main()
{
    std::ios_base::sync_with_stdio(false);
    int n,x,v[10]={1,1,2,3,5,8,1,3,4,7};//1123581347
    cin>>n;
    while(n--)
    {
        cin>>x;
        x--;
        printf("%d\n",v[x%10]);
    }
    return 0;
}
```

—G - Código*

—Límite de tiempo: 1 segundo

Dificultad: $\star$
Temas: Ad-hoc
Complejidad: $O(NM)$

Mau es un chico al que le encanta descifrar c´odigos. Recientemente un amigo suyo le contó acerca de un nuevo código que ha estado desarrollando. Este código consiste en colocar el mensaje a codificar sin espacios en una cuadrícula de $M \times N$, siendo M el número de renglones y N el número de columnas.
Luego se rota la cuadrícula 90 grados. Así, al leer normalmente el mensaje colocado en la cuadrícula, éste será incomprensible.

Un ejemplo se muestra en la siguiente imagen:

Mau puede decodificar este código, pero no lo suficientemente rápido, por eso te ha pedido ayuda a ti, su amigo programador, para implementar un programa que descifre este tipo de códigos.

Tu tarea será, dado el mensaje codificado, imprimir el mensaje original que fue colocado en la cuadrícula.

Entrada

La primera línea contiene un entero $T(1 \leq T \leq 50)$, el número de casos de prueba. Para cada uno de los siguientes T casos siguen 2 líneas.

La primera de ellas contiene dos enteros M y N,$(1 \leq M, N \leq 50)$, que representan el número de renglones y columnas respectivamente en las que esta colocado el mensaje original.

La segunda contiene el mensaje codificado, de tamaño $M \times N$ el cual podrá es-

*Maximiliano Luna Vera - Grupo de Algoritmia Avanzada y Programación Competitiva

tar formado de letras mayúsculas, letras minúsculas y números.

Salida
Para cada caso imprime solo una línea que debe contener el mensaje original, con la distinción de mayúsculas y minúsculas.

Entrada Ejemplo
1
4 5
RSAPTDMRIIAODVRGOEER

Salida Ejemplo
PROGRAMARESDIVERTIDO

Entendiendo el problema
En el problema se nos da una S cadena de tamaño $N \times M$ codificada. La forma en que se codifico la cadena fue metiéndola en una matriz de tamaño $N \times M$ y rotarla 90 grados en sentido del reloj. Después de ésto se agarran los caracteres dentro de la matriz rotada en orden de arriba hacia abajo y de izquierda a derecha. La tarea consiste en decodificar el mensaje.

Solución
Para hacerlo hay que meter la cadena en una matriz de $M \times N$, rotarla 90 grados en sentido contrario al reloj e imprimir los caracteres en el mismo orden. Esto se puede lograr con puro manejo de índices.

Código

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <string>
using namespace std;
int main()
{
    int t,m,n;
    string s;
    cin>>t;
    while(t--)
    {
        cin>>m>>n;
```

```
14          cin>>s;
15          for(int i=m-1;i>=0;i--)
16              for(int j=0;j<n;j++)
17                  cout<<s[i+m*j];
18          cout<<endl;
19      }
20
21      return 0;
22  }
```

8 concurso de programación

XIII Semana de MAC

—D - El Desafío*

—Límite de tiempo: 1 segundo

Dificultad: $\star$
Temas: Matemáticas
Complejidad: $O(1)$

Fer y Dianita son grandes amigas, están la mayor parte del tiempo juntas y siempre se la pasan muy bien. Por alguna extraña razón a Fer le gustan los números naturales que son múltiplos del 3 y a Dianita los que son múltiplos del 5.

Entre las cosas que disfrutan hacer es comer frituras, pero a su amigo Kenny no le agrada, (el piensa que no son para nada saludables), así que un día deliveradamente ellas le propusieron lo siguiente: "Si nos dices cuál es la suma de nuestros números favoritos que son menores a N, hoy no comeremos papitas, más sin embargo si te equivocas, tú nos las tendrás que invitar".

Kenny se preocupa por sus amigas, así que aceptó el desafío y no quiere equivocarse (además también es algo tacaño); la cuestión es que no es muy bueno haciendo cálculos tan rápido. ¿Podrías ayudar a nuestro amigo Kenny diciéndole cuál es la respuesta correcta dado un cierto N?

Entrada

La primera línea de entrada contendrá un entero C, que indica el número de casos de prueba. En las siguientes C lineas aparecerá un entero N.

Salida

Para cada caso de entrada deberás imprimir una sola línea que contenga un entero, la respuesta al problema planteado.

Restricciones fácil $C = 100$
$1 \leq N \leq 50000$
1

*Kenny Yahir Méndez Ramírez - Grupo de Algoritmia Avanzada y Programación Competitiva

Restricciones difícil C = 100000
$1 \leq N \leq 109$

Entrada Ejemplo
3
10
15
100

Salida Ejemplo
23
45
2318

Notas: Se asegura que la respuesta cabe en un entero de 64 bits.

Entendiendo el problema
Dado un entero N se quiere determinar la suma de los múltipos de 3 y 5 que son menores a N. No se deben sumar dos o más veces el mismo número.

Solución
Caso Fácil
Para el caso fácil podemos correr 2 ciclos que verifiquen qué números son divisibles por 3 y por 5 respectivamente, sumarlos, incrementar `i` en 3, 5, y en alguno de los 2 ciclos no contar a aquellos números que son múltiplos de ambos, ya que estaríamos contándolos 2 veces. Esta idea corre en $O(n)$. Lo cual es bueno si n no es "grande".

Caso Difícil
Para el caso dificil, la idea anterior no funcionaría ya que las restricciones obligarían a usar un ciclo demasiado grande, entonces usamos una idea mejor:

Pensemos en la suma de los primeros n múltiplos de 3 como:

$$3 + 6 + 9 + \ldots + 3n$$
$$3(1 + 2 + 3 + \ldots + n)$$

Nos percatamos que el segundo factor de la expresión anterior es la suma de los

primeros n naturales, entonces queda como:

$$3\frac{n(n+1)}{2}$$

De manera similar para el caso de 5:

$$5\frac{n(n+1)}{2}$$

Entonces la suma de las expresiones anteriores es casi lo que queremos, esto porque en cada una de ellas se está contemplando los múltiplos de 5 y 3 al mismo tiempo, así que solo necesitamos eliminar una vez la suma de éstos:

$$15\frac{n(n+1)}{2}$$

Ahora ¿cómo encontramos los primeros n múltiplos menores a N?, fácil, basta con hacer una división entera de $N-1$ entre el divisor deseado. De esta manera nuestro programa se vería de la siguiente forma:

Código

```
#include <iostream>
#include <cstdio>
#define For(x,a,b) for(int x=a; x<b; x++)
using namespace std;

typedef long long ll;

ll mtres(ll n){ll r = n/3; return (r*(r+1)/2)*3;}
ll mcinco(ll n){ll r= n/5; return (r*(r+1)/2)*5;}
ll mquince(ll n){ll r= n/15; return (r*(r+1)/2)*15;}

int main()
{
        int T,N;

        scanf("%d",&T);

        For(j,0,T)
        {
                scanf("%d",&N);

                printf("%lld\n",mtres(N-1)+mcinco(N-1)-mquince(N-1));
        }
        return false;
}
```

—A - Avería*

—Límite de tiempo: 1 segundo

Dificultad: $\star$
Temas: Matemáticas
Complejidad: $O(\log \min(a, b))$

Marta es una chica a la que le encanta conducir, ella es muy hábil al volante y conduce su auto a todas horas y a todos lugares en cualquier tipo de clima. Debido a esto y a la lluvia, el pobre automóvil de Marta ha sufrido una pequeña avería.

Durante estas fuertes lluvias, los limpiaparabrisas de su auto se han descompuesto. Ahora cada uno de los dos limpiaparabrisas realiza su recorrido (ida y vuelta) en tiempos distintos, lo que ocasiona que si ella los detiene cuando alguno de los dos llega al final de su primer recorrido, el otro se encuentre obstruyendo su vista y la distraiga, por lo que podría tener un accidente.

Por lo tanto Marta te ha pedido ayuda a tí. Dado el tiempo en segundos en que cada uno de los limpiaparabrisas de su auto hace su recorrido, calcula el mínimo de segundos que deben pasar para que ambos regresen a su posición de partida. Toma en cuenta que su posición de partida es la misma que tienen los limpiaparabrisas en buenas condiciones cuando están apagados.

Entrada
La primera línea de entrada contiene un entero T, el número de casos de prueba. Las siguientes T líneas contienen dos enteros diferentes A y B, que representan los segundos que tarda cada uno de los limpiaparabrisas en su recorrido.

Salida
Para cada caso de prueba imprime una sola línea con un entero que represente el

*Maximiliano Vera Luna - Grupo de Algoritmia Avanzada y Programación Competitiva

reultado del calculo explicado anteriormente.

Entrada Ejemplo

2
3 5
16 12
1

Salida Ejemplo

15
48

Restricciones
Fácil $1 \leq T \leq 100$ $1 \leq A, B \leq 500$ con A diferente de B.
Difícil $1 \leq T \leq 10000$ $1 \leq A, B \leq 40000$ con A diferente de B.

Entendiendo el problema

Se tienen dos limpiaparabrisas, los cuales realizan un ciclo en a y b segundos, respectivamente. Se dice que los limpiaparabrisas empiezan a funcionar al mismo tiempo, en la misma posición, y se quiere saber en cuánto tiempo volveran en encontrarse en la misma posición por primera vez.

Solución

Lo que buscamos son dos números k, q tal que $ak = bq$, y que ak sea mínimo. La respuesta es ak. Es fácil ver que $ak = bq = lcm(a, b)$ nos minimiza este valor.

Caso Fácil

Para el caso fácil, se puede ir aumentando por si misma una de las variables hasta que la otra la divida exactamente, esto es el primer momento en el que ambos se encuentran en su punto de inicio otra vez.

Caso Difícil

Para el caso difícil, las restricciones no nos permiten hacer el mismo cálculo debido al numero de casos de prueba.

La solución es ver que el Mínimo Común Multiplo de dos números se puede calcular de la siguiente forma:

$$\text{MCM}(a,b) = \frac{ab}{\text{MCD}(a,b)}$$

donde MCD es el máximo común divisor de los dos números.

Ahora, para calcular el Máximo Común Divisor de dos números eficientemente, podemos utilizar el algoritmo de Euclides.

La siguiente rutina basada en ese algoritmo nos devuelve el MCD(a,b) y la calcula en $O(\log n)$ donde n es el máximo de a y b.

Código

```cpp
#include <iostream>
#include <algorithm>
using namespace std;

int gcd(int a, int b) { return b == 0 ? a : gcd(b, a % b); }
int lcm(int a, int b) { return a * (b / gcd(a, b)); }

int main()
{
    int n, a, b;
    cin >> n;
    for(int i = 0; i < n; i++)
    {
        cin >> a >> b;
        cout << lcm(a, b) << endl;
    }
    return 0;
}
```

—C - Conejos*

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆⋆⋆
Temas: Matemáticas
Exponenciación de matrices
Complejidad: $O(\log n)$

Johanna es una brillante matemática amante de los conejos que recientemente consiguió una pareja de bellos conejos a los que nombró Fibonacci y Lucas (se desconoce cuál de los dos conejos es la hembra).

Al cabo de unos meses los pequeños conejos se reprodujeron y Johanna empezó a darse cuenta que el número de parejas de conejos que habían cada mes eran: $1, 1, 2, 3, 5, 8, ...$ y no tardó en deducir que el número de parejas f_n en el mes n estaba dada por la recurrencia $f_n = f_{n-1} + f_{n-2}$.

Johanna tiene dos hermanas menores, Isabella y Karen, quienes también aman a los conejos, y es por eso que Johanna ha decidido regalarles todos sus conejos, sin embargo no quiere que se peleen entre ellas, por lo tanto necesita regalarles exactamente la misma cantidad de parejas de conejos a cada una.

Para poderles regalar la misma cantidad de pares de conejos a Isabella y Karen, Johanna necesita que el número total de parejas de conejos sea par, así que a ella sólo le interesan los meses en lo que esto sucede y a dichos meses los ha nombrado conejo-meses. Así en los primeros 4 conejo-meses el número de parejas de conejos son: 2, 8, 34 y 144.

Johanna necesita tu ayuda para determinar cuántos conejos tendrá en el N-ésimo conejo-mes.
Como este número puede ser muy grande Johanna te pide que imprimas el resultado módulo $109 + 7(1000000007)$.

Entrada

La primera línea contiene un entero t, el número de casos de prueba. Las siguientes t líneas contienen un entero N

*David Felipe Castillo Velázquez - Grupo de Algoritmia Avanzada y Programación Competitiva

Salida
Para cada caso de prueba debes imprimir, en una línea, el número de pares de conejos que tendrá Johanna en el N-ésimo conejo-mes.

Restricciones
Fácil t = 100
$1 \leq N \leq 1000$

Difícil t = 1000
$1 \leq N \leq 1018$

Entrada Ejemplo
5
1
2
5
10
43

Salida Ejemplo
2
8
610
832040
461493940

Notas Recordar que:
$(a + b)\%M = (a\%M) + (b\%M)$ $(a \times b)\%M = ((a\%M) \times (b\%M))\%M$

Entendiendo el problema
El problema consistía en encontrar el n-ésimo fibonacci par.

Solución
Sea $f_n = 4f_{n-1} + f_{n-2}$, con $f_1 = 2$ y $f_2 = 8$. Mostraremos que f_n nos da el n-ésimo fibonacci par. Dado que cada tercer elemento de la sucesión de fibonacci es par,

basta con probar que $fib_n = 4fib_{n-3} + fib_{n-6}$.

$$\begin{aligned} fib(n) &= fib_{n-1} + fib_{n-2} \\ &= fib_{n-2} + 2fib_{n-3} + fib_{n-4} \\ &= 3fib_{n-3} + 2fib_{n-4} \\ &= 3fib_{n-3} + fib_{n-4} + fib_{n-5} + fib_{n-6} \\ &= 4fib_{n-3} + fib_{n-6} \end{aligned}$$

Sea $f_n = 4f_{n-1} + f_{n-2}$, con $f_1 = 2$ y $f_2 = 8$. Mostraremos que f_n nos da el n-ésimo fibonacci par. Dado que cada tercer elemento de la sucesión de fibonacci es par, basta con probar que $fib_n = 4fib_{n-3} + fib_{n-6}$.

$$\begin{aligned} fib(n) &= fib_{n-1} + fib_{n-2} \\ &= fib_{n-2} + 2fib_{n-3} + fib_{n-4} \\ &= 3fib_{n-3} + 2fib_{n-4} \\ &= 3fib_{n-3} + fib_{n-4} + fib_{n-5} + fib_{n-6} \\ &= 4fib_{n-3} + fib_{n-6} \end{aligned}$$

Con esto sólo necesitamos calcular f_n.

Caso Fácil

Para el caso fácil era posible hacerlo en $O(n)$ con un **for**.

Caso Difícil

Sin embargo para las restricciones difíciles ésto ya no era posible. Para resolver el caso difícil necesitamos observar que:

$$\begin{pmatrix} f_3 \\ f_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_2 \\ f_1 \end{pmatrix}$$

Utilizando el mismo razonamiento tenemos:

$$\begin{aligned} \begin{pmatrix} f_4 \\ f_3 \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_3 \\ f_2 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^2 \begin{pmatrix} f_2 \\ f_1 \end{pmatrix} \end{aligned}$$

Generalizando ésta idea se obtiene:

$$\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-2} \begin{pmatrix} f_2 \\ f_1 \end{pmatrix}$$

Ahora sólo necesitamos obtener $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ de forma eficiente. Basándonos en el hecho de que

$$x^n = \begin{cases} (x^{\frac{n}{2}})(x^{\frac{n}{2}}) & \text{si } n \text{ es par} \\ x(x^{\frac{n}{2}})(x^{\frac{n}{2}}) & \text{si } n \text{ es impar} \end{cases}$$

podemos utilizar el siguiente procedimiento para calcularlo en $O(\log(n))$

Código

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
#include <bitset>
#include <cmath>
#include <queue>

#define For(i, a, b) for(int i=(a); i<(b); ++i)
#define INF (1<<30)
#define MP make_pair
#define MOD 1000000007

using namespace std;

typedef pair <int, int> ii;
typedef vector <vector <int> > vvi;

int madd(int a, int b)
{
    return (a + b) % MOD;
}

int mmult(int a, int b)
{
    return (1LL*a*b) % MOD;
}

vvi matmult(vvi A, vvi B)
{
    vvi C(2, vector <int>(2, 0));
    For(i, 0, 2)
```

```
34          For(j, 0, 2)
35              For(k, 0, 2)
36                  C[i][j] = madd( C[i][j], mmult(A[i][k], B[k][j]) );
37
38      return C;
39  }
40
41  vvi matexp(vvi X, long long n)
42  {
43      if (n == 1)
44          return X;
45
46      vvi ans = matexp(X, n >> 1);
47      if (n & 1)
48          return matmult( matmult(ans, ans), X );
49
50      return matmult(ans, ans);
51  }
52
53  int main()
54  {
55      int tt;
56      scanf("%d", &tt);
57
58      while (tt--)
59      {
60          long long n;
61          scanf("%lld", &n);
62
63          if (n == 1)
64              printf("2");
65          else if (n == 2)
66              printf("8");
67          else
68          {
69              vvi X(2, vector <int> (2, 1));
70              X[0][0] = 4;
71              X[1][1] = 0;
72
73              X = matexp(X, n-2);
74
75              printf("%d\n", madd( mmult(X[0][0], 8), mmult(X[0][1], 2) ) );
76          }
77      }
78
79      return 0;
80  }
```

9 5to concurso Interno

de programación

—A - Analizando Exámenes*

—Límite de tiempo: 1 segundo

Dificultad: $\star$
Temas: Ad-hoc
Complejidad: $O(n)$

Acaba de pasar la semana de exámenes en la FES Acatlán y los profesores están preocupados porque sospechan que muchos alumnos hicieron trampa durante las pruebas. Los profesores ya calificaron los exámenes y ahora están buscando exámenes sospechosos. Ayúdalos con su tarea (Aunque hayas hecho trampa en algún examen).

Un profesor desea comparar dos exámenes y contar el número de coincidencias en las respuestas.

Haz un programa que lea los resultados de dos alumnos y cuente el número de respuestas iguales que tienen.

Entrada

La primera línea contiene un entero n, $1 \leq n \leq 1000$, el número de preguntas del exámen.
Las siguientes dos líneas contienen cada una n enteros $a_i \in 0, 1$, los resultados de cada pregunta del examen donde 0 es respuesta equivocada y 1 es respuesta correcta.

Salida

Deberás imprimir un entero k que indica la cantidad de coincidencias entre los dos exámenes.

Entrada Ejemplo

5
0 1 0 0 1

*Og Astorga Díaz - Grupo de Algoritmia Avanzada y Programación Competitiva

0 1 1 0 1

Salida Ejemplo
4

Entendiendo el problema
Dados dos arreglos A y B, de tamaño n, se te pide saber cuántos índices i cumplen que $a[i] = b[i]$.

Solución
Directa.

Código

```
import java.util.*;

class Amain
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        int a[] = new int[1005], b[] = new int[1005];
        int n = sc.nextInt();

        for(int i = 0; i < n; i++)
            a[i] = sc.nextInt();

        for(int i = 0; i < n; i++)
            b[i] = sc.nextInt();

        int ans = 0;
        for(int i = 0; i < n; i++)
            ans += a[i] == b[i] ? 1 : 0;

        System.out.println(ans);
    }
}
```

—B - Buscando maíz*

—Límite de tiempo: 1 segundo

Dificultad: $\star$
Temas: Ad-hoc
Complejidad: $O(N)$

Anselmo es alumno de MAC, como ustedes sabrán, al igual que la mayoría se encuentra en la semana de éxamenes y tareas. Por fortuna cuenta con un buen equipo, por lo que hasta el momento todo ha marchado bien.

Ha concluido casi todos sus trabajos, pero, como sabemos, siempre hay algo que se nos escapa, y para Anselmo no fue la excepción. Hoy mientras platicaba con sus amigos en las jardineras, se percató de que una de sus tareas se entrega hoy alas 15.00 hrs !.

Como dijimos, trataban de llevar los trabajos al día, por lo que parte de esta tarea ya está hecha, y como en todo equipo, a cada integrante le tocó una actividad, lo malo es que, de entre las partes que faltan, está la que le corresponde a Anselmo. Vamos a resumir el objetivo de la tarea a presentar en un enunciado:

Dados los diferentes tipos de pagos (S) que se realizan en las cajas de la FES, y la Matriz de probabilidades de transición de un tipo de pago a otro, queremos saber, cuál es la probabilidad condicional de que la persona $t + k$, $k > 0$, realice un pago $X = b$ si sabe que la persona t hizo un tipo de pago $X = a$, donde $a, b \in 1, ..., S$.

A una parte del equipo le correspondió obtener una muestra de N personas observando que tipo de pago realizaba una persona al llegar a las cajas de la facultad (esto es lo que ya tienen). A a partir de ésta se debe obtener la Matriz de transiciones que se mencionó antes y con ella se podrán aplicar los métodos necesarios para poder responder la pregunta del párrafo anterior.

Bien, pues la creación de la matriz de transiciones es la parte que Anselmo debe de terminar. Como no quiere cometer errores porque sabe que de eso depende la calificación de todo el equipo te ha pedido a ti, (que eres muy listo y muy rápido) que lo ayudes, para que pueda acabar pronto.

*Kenny Yahir Méndez Ramírez - Grupo de Algoritmia Avanzada y Programación Competitiva

Definimos la matriz de transición de la siguente manera:
Donde a_{ij} corresponde a la probabilidad de que la persona $t+1$ hago un pago j

$$\begin{array}{c} \\ 1 \\ 2 \\ \vdots \\ S \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \cdots & S \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1S} \\ a_{21} & a_{22} & \cdots & a_{2S} \\ \vdots & \ddots & & \\ a_{S1} & a_{S2} & \cdots & a_{SS} \end{pmatrix} \end{array}$$

dado que la persona t hizo un pago i.

Entrada
Un entero C, $1 \leq C \leq 100$, que es el número de casos a procesar seguido de una línea en blanco. Por cada caso de entrada, se recibirán dos enteros $5 \leq N \leq 500$, $3 \leq S \leq 15$ y a continuación N observaciones dónde apareceran los tipos de pagos $X = 1, 2, ..., S$.

Salida
Deberán aparecer C matrices de transición una por cada caso de la entrada, donde los elementos deberán estar redondeados a dos lugares decimales. Habrá una linea en blanco después de cada matriz.

Entrada Ejemplo
```
2
8 4
1 2 3 1 1 3 2 4
5 3
1 2 3 3 1
```

Salida Ejemplo
```
0.33 0.33 0.33 0.00
0.00 0.00 0.50 0.50
0.50 0.50 0.00 0.00
0.00 0.00 0.00 0.00
0.00 1.00 0.00
0.00 0.00 1.00
0.50 0.00 0.50
```

Entendiendo el problema
Dada una serie de N observaciones y a lo más S tipos de pago, se quiere encontrar una matriz de transiciones entre estos estados. Éstas son las probabilidades de cada

estado por cada entrada de la matriz. Esta matriz no es simétrica.

Solución

Se recorre la cadena de observaciones, se identifica el tipo de observación o_i y o_{i-1}, y en una matriz se cuentan dichas incidencias, esta incidencia debe ser del tipo $s = o_{i-1} \rightarrow s' = o_i$. Entonces por cada una de estas incidencias podemos hacer `mat[s][s']++`. Al final cada renglón de la matriz se normaliza por el total de observaciones del tipo s.

Código

```
import java.util.*;

class Bmain
{
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                int tt = sc.nextInt();

                while (tt-- > 0)
                {
                        double mat[][] = new double[20][20], tot[] = new
                         ↪  double[20];
                        int arr[] = new int[505];
                        int S, N;

                        N = sc.nextInt();
                        S = sc.nextInt();

                        for(int j = 0; j < N; j++)
                        {
                                arr[j] = sc.nextInt();
                                arr[j]--;
                        }

                        for(int j = 0; j < N-1; j++)
                        {
                                tot[arr[j]]++;
                                mat[arr[j]][arr[j+1]]++;
                        }

                        for(int i = 0; i < S; i++)
                        {
                                for(int j = 0; j < S; j++)
                                {
                                        if(tot[i] > 0)
                                                System.out.printf("%.2f ",
                                                 ↪  mat[i][j] / tot[i]);
```

```
37                                          else
38                                                  System.out.printf("%.2f ",
                                                     ↪  mat[i][j]);
39                                      }
40                                      System.out.println();
41                                  }
42                                  System.out.println();
43                              }
44                      }
45              }
```

—C - Contando nombres*

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆⋆
Temas: Matemáticas
Complejidad: $O(n^2)$

Kenny es un ser polinominal, es decir, se le conoce por varios nombres, e.g, "Kenny", "Kevin", "Kenan". Queremos creer que a Kenny no le molesta tener varios nombres, pero al parecer a su novia sí; "¡Se llama Kenny!", grita cada vez que alguien lo llama con otro nombre.

Para intentar reducir su enojo, decidimos utilizar nombres que contengan las mismas letras que "Kenny", i.e, anagramas. Después de un tiempo nos dimos cuenta que ahora teníamos nombres de sobra para Kenny, y nos dió curiosidad saber exáctamente cuántos. ¿Y por qué detenernos sólo con "Kenny"?

Entrada

La primer línea contendrá t, $1 \leq t \leq 100$, el número de casos de prueba. Para cada caso de prueba, en una línea, habrá una cadena S, compuesta por puras letras minúsculas, $1 \leq longitud(S) \leq 200$.

Salida

Para cada caso de prueba tendrás que imprimir un entero $kmod10^9+7(1000000007)$, que indica el número de anagramas distintos que existen para la cadena S.

Entrada Ejemplo

3
kenny
algoritmia
aaaa

Salida Ejemplo

60
907200
1

*David Felipe Castillo Velázquez - Grupo de Algoritmia Avanzada y Programación Competitiva

Entendiendo el problema

Dada una cadena de tamaño n, compuesta por letras minúsculas, se te pide saber cuántas cadenas diferentes puedes formar usando los caracteres de la cadena.

Solución

Sabemos que podemos acomodar los n caracteres de la cadena en $n!$ formas, creando este número de cadenas. Sin embargo existen algunas cadenas que se repiten, en especial, si a_i es la cantidad que se repite el i-ésimo tipo de caracter de la cadena, entonces se repiten $a_1!a_2!...a_k!$ cadenas. Por lo tanto la respuesta es $\frac{n!}{a_1!a_2!...a_k!} = \binom{n}{a_1}\binom{n-a_1}{a_2}...\binom{n-a_1-...-a_{k-1}}{a_k}$

Código

```
#include <bits/stdc++.h>

#define For(i, a, b) for(int i=a; i<b; ++i)
#define INF (1<<30)
#define MP make_pair
#define MOD 1000000007

using namespace std;

typedef pair <int, int> ii;
typedef long long ll;

int C[205][205], v[30];

int main()
{
    //std::ios_base::sync_with_stdio(false);

    For(i, 0, 201)
        For(j, 0, i+1)
            C[i][j] = i == j or j == 0 ? 1 : (C[i-1][j] + C[i-1][j-1]) % MOD;

    int tt;
    scanf("%d", &tt);

    while (tt--)
    {
        memset(v, 0, sizeof v);

        char s[205];
```

```
31          scanf("%s", s);
32
33          int n = strlen(s);
34          For(i, 0, n)
35              v[s[i]-'a']++;
36
37          int ans = 1;
38          For(i, 0, 'z'-'a'+1)
39          {
40              if (v[i] > 0)
41                  ans = (ans*C[n][v[i]]) % MOD;
42              n -= v[i];
43          }
44
45          printf("%d\n", ans);
46      }
47
48      return 0;
49  }
```

—D - Dados*

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆
Temas: Matemáticas
Complejidad: $O(1)$

Seguramente has visto la serie "The Big Bang Theory", y seguramente has visto el episodio donde Sheldon se basa en unos dados para tomar todas sus decisiones. Cada vez que Sheldon tenía que tomar una decisión lanzaba los dados y le asignaba a cada opción un número, dependiendo de la suma de los dados, elegía la opción asociada.

Nosotros queremos adoptar la estrategia de Sheldon, sin embargo hay momentos en que definitivamente tenemos preferencia por una opción, por ejemplo, cuando decidimos entre ¡hacer la tarea¿ o ¡programar¿, preferimos que salga la segunda.

De acuerdo a la cantidad de dados que usamos para decidir y el número de caras que tienen los dados, queremos que nos ayudes a saber qué número asociarle a nuestra opción preferida para tener la mayor probabilidad de elegirla.

Entrada

La primer línea contendrá el número de casos de prueba t, $1 \leq t \leq 100$. Para cada caso de prueba, en una línea, habrá dos enteros n y m, $1 \leq n, m \leq 50$, el número de dados que usamos y el número de caras de los dados, respectivamente. Las caras de los dados están marcadas del 1 al m.

Salida

Para cada caso de prueba tendrás que imprimir la suma de los dados que tiene la mayor probabilidad de salir, si existen varias soluciones, imprime la menor de ellas.

Entrada Ejemplo

```
3
1 6
2 6
50 50
```

*David Felipe Castillo Velázquez - Grupo de Algoritmia Avanzada y Programación Competitiva

Salida Ejemplo

1
7
1275

Entendiendo el problema

Dados n dados, todos con m caras, quieres saber cuál es la suma de las caras de los dados con mayor probabilidad de salir al lanzar todos los dados.

Solución

Sea a y b el mínimo y máximo, respecticamente, valor posible para la suma. Entonces es fácil ver que la respuesta es $\frac{a+b}{2}$.

Código

```
#include <bits/stdc++.h>

#define For(i, a, b) for(int i=(a); i<(b); ++i)
#define INF 1000000000
#define MP make_pair

using namespace std;

typedef pair <int, int> ii;
typedef long long ll;

double p[55][2505];

int main()
{
    //std::ios_base::sync_with_stdio(false);

    int tt;
    scanf("%d", &tt);

    while (tt--)
    {
        int n, m;
        scanf("%d %d", &n, &m);
        printf("%d\n", n == 1 ? 1 : (n*m - n)/2 + n);
    }

    return 0;
}
```

—E - Encriptando Mensajes*

—Límite de tiempo: 1 segundo

Dificultad: $\star$
Temas: Implementación
Complejidad: $O(n)$

David es un chico singular, muy discreto, no le gustan las imprudencias. Por tanto es de esperarse que sus actividades las realice de manera muy reservada. Tanto así que en estos últimos dias se le ha ocurrido la idea de encriptar sus mensajes que manda por Face a sus amigos. Una manera muy fácil de hacerlo, y muy conocida además, es utilizar la operación binaria XOR, esta es, dados dos enteros x y k donde al ´ultimo lo usamos como clave, obtenemos un entero $y = xXORk$ que puede ser usado para devolvernos el valor de x ejecutando la operación $yXORk$. Sabiendo esto, nos percatamos que con reconocer el valor de la clave k podemos enviar cualquier mensaje entre todos los que conozcan tal valor.

Bien, como tú formas parte del circulo social de David deberáss crear tu propio programa que encripte los mensajes que quieras enviarle, proporcion´andote la clave que David ha elegido para ti.

Entrada
La primera línea contendrá un entero C que son los casos a procesar. En los siguientes C bloques aparecer´´4 un un entero $1 \leq k \leq 31$ en la primera linea y en la segunda linea el mensaje a encriptar. La linea no usar más 1000 caracteres.

Salida
Serán C lineas, en cada una el mensaje encriptado para cada clave correspondiente.

Entrada Ejemplo
2
5
Hola, que tal estas?
31

Salida Ejemplo
Mjid)%tp‘%qdi%‘vqdv:

*Kenny Yahir Méndez Ramírez - Grupo de Algoritmia Avanzada y Programación Competitiva

Pfz3?zs?ji ?zlk ?— vp?izm

Entendiendo el problema

Dada una cadena s y un entero k, se te pide que encriptes s haciendo x XOR k para cada caracter x de s.

Solución

Directa.

Código

```
#include <cstdio>
#include <iostream>
#include <string>
#define For(x,a,b) for(int x=(a); x<(b); x++)
using namespace std;

int main()
{
    int C,k;
    string cad;

    scanf("%d",&C);

    For(c,0,C)
    {
        cin >> k;
        getline(cin,cad);
        getline(cin,cad);

        For(i,0,cad.size())
        {
            printf("%c ",(cad[i]^k));
        }
        printf("\n");
    }
    return 0;
}
```

—F - Focos*

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆
Temas: Matemáticas
Complejidad: $O(1)$

El interruptor de la luz del cubo de Algoritmia se descompuso, pero afortunadamente, en el CEDETEC, hay un interruptor universal que cambia de estado, de forma un poco peculiar, los focos de todos los cubos. Después de un rato jugando con el interruptor descubrimos que en la i-ésima vez que accionamos el interruptor, los focos de los cubos cuya numeración es divisible por i, cambian de estado; si están apagados se prenden y si están prendidos se apagan. Por ejemplo, si es la segunda vez que accionamos el interruptor, los focos de los cubos $2, 4, 6, 8, ...$ cambian de estado. El número del cubo de algoritmia es n y suponemos que después de n veces que accionemos el interruptor, el cubo va a tener luz. Ayúdanos a comprobar si ´esto es cierto. El foco del cubo est´a inicialmente apagado.

Entrada

La primer línea contendrá el número de casos de prueba $t, 1 \leq t \leq 100$. Para cada caso de prueba, en una línea, habrpa un entero $n, 1 \leq n \leq 1018$, descrito anteriormente.

Salida

Para cada caso de prueba tendrás que imprimir la cadena “les hace falta estudiar” si es que el foco del cubo de algoritmia termina apagado, o “son unos genios” en caso contrario.

Entrada Ejemplo

3
7
1
16
S

Salida Ejemplo

les hace falta estudiar

*David Felipe Castillo Velázquez - Grupo de Algoritmia Avanzada y Programación Competitiva

son unos genios
son unos genios

Entendiendo el problema

Se te da un arreglo de focos y un interruptor que, al accionarlo la i-ésima vez, cambia el estado de los focos que son múltiplos de i. Después de accionar n veces el interruptor, quieres saber el estado del n-ésimo foco, el cuál está inicialmente apagado.

Solución

Si el número de divisores de n es par, el foco estará apagado, sino estará prendido. Un entero n tiene un número par de divisores si y sólo si es un cuadrado perfecto.

Código

```cpp
#include <bits/stdc++.h>

#define For(i, a, b) for(int i=a; i<b; ++i)
#define INF (1<<30)
#define MP make_pair

using namespace std;

typedef pair <int, int> ii;
typedef long long ll;

int main()
{
    //std::ios_base::sync_with_stdio(false);

    int tt;
    scanf("%d", &tt);

    while (tt--)
    {
        int n;
        scanf("%d", &n);

        int r = sqrt(n);
        printf("%s\n", r*r == n ? "son unos genios" : "les hace falta estudiar");
    }

    return 0;
}
```

—G - Grupos*

—Límite de tiempo: 1 segundo

Dificultad: $\star$
Temas: Ad-hoc
Complejidad: $O(nm)$

La coordinación de MAC necesita asignarle los grupos a los alumnos de nuevo ingreso, sin embargo no quieren asignarlos completamente de forma aleatoria. Ellos quieren evitar que en un grupo haya dos o más personas con exactamente la misma personalidad, pues de ser así, esos alumnos podrían causar muchos problemas.

Se te ha pedido que ayudes a verificar si es posible realizar una asignación de los alumnos que cumpla los requerimientos de la coordinación, dados el número de alumnos, la personalidad de cada uno de ellos y el número de alumnos que debe tener cada grupo.

La personalidad de un alumno estará representada por un entero a_i. Todos los grupos deben tener la misma cantidad de alumnos.

Entrada

La primera línea contiene un entero $t, 1 \leq t \leq 100$, el número de casos de prueba. Para cada caso de prueba habrá dos líneas. La primera contendrá dos enteros n y m, $1 \leq n, m \leq 100$, el número de grupos y el número de alumnos que debe tener cada grupo, respectivamente. La segunda línea tendá nm enteros a_i ,$0 \leq ai \leq 105$, separados por espacio, que representan la personalidad del alumno i.

Salida

Para cada caso de prueba debes imprimir, en una línea, la cadena “si”, si es posible asignar los alumnos de tal forma que no existan dos o más alumnos con la misma personalidad en el mismo grupo. De no existir una asignación, imprime la cadena “no”.

Entrada Ejemplo

3
4 4
5 5 2 3 3 4 8 7 2 5 1 5 2 4 3 2

*David Felipe Castillo Velázquez - Grupo de Algoritmia Avanzada y Programación Competitiva

1 1
5
2 3
1 1 1 1 1

Salida Ejemplo

si
si
no

Entendiendo el problema

Dados n grupos, donde deben haber m alumnos, nm alumnos y la personalidad a_i del i-ésimo alumno, quieres saber si es posible asignar los alumnos a los grupos de tal forma que no haya un grupo con más de un alumno de la misma personalidad.

Solución

Si existen $k > n$ alumnos con la misma personalidad, es imposible crear los grupos, en caso contrario es siempre posible.

Código

```
#include <bits/stdc++.h>

#define For(i, a, b) for(int i=a; i<b; ++i)
#define INF (1<<30)
#define MP make_pair

using namespace std;

typedef pair <int, int> ii;
typedef long long ll;

int a[100005];

int main()
{
    //std::ios_base::sync_with_stdio(false);

    int tt;
    scanf("%d", &tt);

    while (tt--)
    {
        memset(a, 0, sizeof a);
```

```
24
25          int n, m;
26          scanf("%d %d", &n, &m);
27
28          bool ok = true;
29          For(i, 0, n*m)
30          {
31              int x;
32              scanf("%d", &x);
33              a[x]++;
34              if (a[x] > n)
35                  ok = false;
36          }
37
38          printf("%s\n", ok ? "si" : "no");
39      }
40
41      return 0;
42  }
```

10 concurso de programación

XIV Semana de MAC

—A - Armando Elecciones*

—Límite de tiempo: 5 segundos

Dificultad: ⋆⋆⋆
Temas: Teoría de Gráficas
Busquedas
Complejidad: $O(n + r)$

En Maclandia se llevarán a cabo elecciones presidenciales muy pronto, pero dado que conllevan un costo muy elevado, su Instituto Electoral te ha encargado que les digas si es posible determinar quién ganará las elecciones presidenciales, ya que si esto es posible, les ahorrarías mucho dinero.

En Maclandia ocurren muchas situaciones que no ocurren en los demás paises de su planeta Namek, por ejemplo: debido a ciertas influencias (dinero, dinero, dinero, nepotismo, etc.), el voto de algunas personas vale más que el de otras; además, si una persona vota por cierto candidato todos sus amigos votarán por el mismo candidato; aquellas personas que no tengan amigos con un voto definido, el día de las elecciones votarán por algún candidato según su sano juicio.

Las personas en Maclandia tienen agregados a todos sus amigos en su red social, y si una persona es amiga de otra, entonces esta otra también es amiga de esa persona. Es bien sabido que los candidatos votarán por ellos mismos.

Sin embargo en este país existen dos genios: uno malvado y uno bueno. Si alguien es amigo del genio malvado entonces el candidato por el que votaría perderá las elecciones, sin importar el valor de los votos que haya obtenido.

En caso contrario si alguien es amigo del genio bueno, el candidato por el que votaría ganará las elecciones. Si los genios intentan hacer que el mismo candidato gane y pierda las elecciones entonces se anularán sus poderes y no tendrán poder sobre las elecciones. Si algún genio no tiene amigos, entonces este no podrá ejercer

*Silverio Flores Moroni - Grupo de Algoritmia Avanzada y Programación Competitiva

su poder para influir en las elecciones. Los genios sólo pueden ser amigos (directa o indirectamente) de personas que ya tienen predeterminado su voto. Ninguna persona votará por dos candidatos, puesto que ninguna persona será amiga indirecta de dos candidatos al mismo tiempo. Un genio es considerado como una persona en sus relaciones de amistad.

Entrada

La primera línea será un entero $1 \leq T \leq 100$, indicando la cantidad de casos de prueba. Cada caso de prueba se compondrá de varias lineas: la primera tendrá tres números n, c y r $(2 \leq n \leq c \leq 500, 0 \leq r \leq (n + 2 - c)2)$ indicando el número de votantes, el número de candidatos y el número de relaciones de amistad respectivamente.

Los votantes son identificados con los números 1 hasta n inclusive. La siguiente línea tendrá n números enteros $(1 \leq a_j \leq 10000)$, el j-ésimo número indicará la cantidad de puntos que vale el voto de la j-ésima persona. En la siguiente línea vendrán c números, los cuales indican el número de cada candidato. r líneas vendrán después, cada una con dos números $-1 \leq a, b \leq n$, indicando que las personas a y b son amigas. La persona 0 es el genio bueno, y la persona -1 es el genio malvado. Los casos de prueba están separados por una línea en blanco.

Salida

Para cada caso, indica en una línea distinta: el número del candidato que ganará las elecciones, si es posible determinar quien las ganara; si no es posible determinar esto, imprime “Es mas facil salir de la friendzone que saber quien ganara” sin las comillas.

Entrada Ejemplo

```
2
4 3 3
1 10 30 1
1 2 3
1 4
2 0
3 -1
5 3 2
1 50 10 2 20
1 2 3
1 4
2 -1
```

Salida Ejemplo

2
Es mas facil salir de la friendzone que saber quien ganara

Explicación de la entrada ejemplo

En el segundo caso, la persona 2 pierde las elecciones automaticamente porque es amiga del genio malvado, la persona 1 puede juntar 3 puntos resultado de la suma de sus puntos y los de sus amistades, la persona 3 puede juntar 10 puntos. Sin embargo no es posible determinar quien ganará la elección, a pesar de que la persona 3 tiene más puntos, porque la persona 5, que no es amiga directa/indirecta de algún candidato, cuando vote puede hacerlo por alguno de los dos (1 ó 3) inclinando la balanza a favor de cualquiera.

Entendiendo el problema

Para cada caso de prueba se debe construir una gráfica con las relaciones de amistad como aristas y como vértices las n personas. Cada candidato representa una componente de la gráfica.

Solución

Para cada candidato se realiza una búsqueda en amplitud y por cada vértice que puede alcanzar se suma la cantidad de puntos que ese vértice tiene. Se considera que el genio malo aporta $-\infty$ puntos y el bueno ∞. Al final se suman los puntos de aquellas personas que no pertecen a ninguna componente representada por algun candidato; si esta suma más los puntos del segundo candidato más alto (en puntos) es mayor o igual a los puntos del candidato más alto entonces no es posible determinar quién ganará las eleciones, en caso contrario las gana el candidato con más puntos.

Código

```
#include <iostream>
#include <queue>
#include <algorithm>
using namespace std;

queue <int> cola;
int matriz[505][505], votos[505], personas[505];
pair <int, int> candidatos[505];
int infinito=100000000;
int n;
```

```
11
12  int buscar()
13  {
14      int suma =0, a, i;
15      bool ganaste=false, perdiste=false;
16      while (!cola.empty() )
17      {
18          a=cola.front();
19          cola.pop();
20          if ( personas[a] ) continue;
21
22          if (a==0)   ganaste=true;
23          else if (a==n+1) perdiste=true;
24          else suma += votos[a];
25          personas[a]=1;
26          for (i=-1; ++i<=n+1; )
27              if (matriz[a][i]==1)
28                  cola.push(i);
29      }
30      if (ganaste && !perdiste) suma= infinito;
31      if (perdiste && !ganaste) suma =-infinito;
32      return suma;
33  }
34
35  int main()
36  {
37      int t, c, r, a, b, i, j;
38      cin>>t;
39      while (t--)
40      {
41          cin>>n>>c>>r;
42
43          //limpiar
44          while (!cola.empty() ) cola.pop();
45          for (i=-1; ++i<=n+1; )
46              for (j=-1; ++j<=n+1; )
47                  matriz[i][j]=0;
48          for (i=-1; ++i<=n+1; )
49              personas[i]=0;
50          for (i=-1; ++i<c; )
51              candidatos[i].first=0;
52          votos[0]=votos[n+1]=0;
53
54          for (i=0; ++i<=n; )
55              cin>>votos[i];
56
57          for (i=-1; ++i<c; )
58              cin>>candidatos[i].second;
59
```

```
60          while (r--)
61          {
62              cin>>a>>b;
63              if (a==-1) a=n+1;
64              if (b==-1) b=n+1;
65              matriz[a][b]=1;
66              matriz[b][a]=1;
67          }
68
69          for (i=-1; ++i<c; )
70          {
71              cola.push( candidatos[i].second );
72              candidatos[i].first = buscar();
73          }
74
75          sort(candidatos, candidatos+c);
76
77          if (candidatos[c-1].first==infinito  || candidatos[c-2].first==-infinito
             ↪  )
78              cout<<candidatos[c-1].second<<endl;
79          else
80          {
81              //contar los no amigos
82              int otrasuma=0;
83              for (i=0; ++i<=n; )
84                  if ( personas[i]==0 )
85                      otrasuma+=votos[i];
86              if ( (candidatos[c-2].first + otrasuma) >= candidatos[c-1].first )
87                  cout<<"Es mas facil salir de la friendzone que saber quien
                     ↪  ganara\n";
88              else
89                  cout<<candidatos[c-1].second<<"'\n";
90          }
91      }
92      return 0;
93  }
```

—B - Buscando amigos*

—Límite de tiempo: 3 segundos

Dificultad: ⋆⋆⋆
Temas: Matemáticas
Teoría de Números
Complejidad: $O(n)$

Después de una intensa jornada de actividades en la semana de MAC, algunos alumnos han encontrado a quien consideran "el amor de su vida". Sin embargo muchos estudiantes de MAC se consideran a sí mismos matemáticos puros y no le hablan a cualquier persona. Los alumnos que se inscribieron a la semana de MAC recibieron un id (numero de identificación) para poder entrar a las conferencias. Los matemáticos puros sólo le hablarán a la persona cuyo id a sea amigo de su id b. Un número a es amigo de b si la suma de los divisores de a (exceptuando sólo a a) es igual a b, nota que si a es amigo de b, b no necesariamente es amigo de a. Algunos alumnos de MAC han recurrido a ti debido a tus grandes conocimientos en estas áreas del saber para saber si "el amor de su vida" les hablará. Por causas fortuitas todos estos amores son matemáticos puros. Tu tarea es determinar si los alumnos tienen oportunidad de hablar con quienes consideran el amor de su vida.

Entrada

La primer lpinea de la entrada contendrá un entero $T \leq 200$, el número de casos de prueba, a continuación se darán T renglones, cada uno con un caso de prueba. Cada renglón tendrá dos números a, b, tales que $1 \leq a, b \leq 10000$

Salida

Para cada caso de prueba imprime "Quiza y sin quiza", sin las comillas si a es amigo de b; imprime "Llora desconsoladamente" en caso contrario.

Entrada Ejemplo

```
3
6 5
3 10
1184 1210
```

*Silverio Flores Moroni - Grupo de Algoritmia Avanzada y Programación Competitiva

Salida Ejemplo

Llora desconsoladamente
Llora desconsoladamente
Quiza y sin quiza

Entendiendo el problema

Debemos saber si la suma de los divisores propios de un número a son iguales a un número b.

Solución

En este concuso se permitió la solucion "naive", la cual consisitia en verificar todos los números menores a a y sumar aquellos que lo dividen, después verificar si esa suma es igual a b. Sin embargo existe una mejor solucion:

Sean $p_1, p_2, \ldots, p_k$ los factores primos de a. Podemos decir que:

$$a = p_1^{f_1} \times p_2^{f_2} \times \ldots \times p_k^{f_k}$$

La suma de los divisores de a, incluyendo a a es:

$$S = \frac{p_1^{f_1+1} - 1}{p_1 - 1} \times \frac{p_2^{f_2+1} - 1}{p_2 - 1} \times \ldots \times \frac{p_k^{f_k+1} - 1}{p_k - 1}$$

Entonces basta con verificar si $S - a = b$. A pesar de que esta solución se ve mas compleja, computacionalmente es mejor y más rápida.

Código

```
#include <iostream>
using namespace std;
int factores[1000][2], i;

void descomponer(int a)
{
    int j =1;
    while (a!=1)
    {
        ++j;
        if (a%j==0)
        {
            ++i;
            factores[i][0]=j;
            while (a%j==0)
            {
                ++factores[i][1];
                a /=j ;
```

```
19              }
20          }
21      }
22  }
23
24  int geometrica(int j)
25  {
26      int k, res=1;
27      for (k=-1; ++k<=factores[j][1]; )
28          res *=factores[j][0];
29      --res;
30      res /=(factores[j][0]-1);
31      return res;
32  }
33
34  int main()
35  {
36      int t, a, b, j,m, res;
37      cin>>t;
38      while (t--)
39      {
40          cin>>a>>b;
41          i=-1;
42
43          for (j=-1; ++j<1000; factores[j][1]=0);  //limpiar
44
45          descomponer(a);
46
47          res=1;
48          for (j=-1; ++j<=i;)
49          {
50              res *=geometrica(j);
51          }
52          res -=a;
53          if (res!=b)
54              cout<<"Llora desconsoladamente\n";
55          else cout<<"Quiza y sin quiza\n";
56      }
57  }
```

—C - Caballo sin mecate*

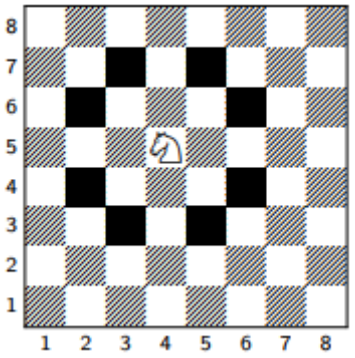
—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆
Temas: Teoría de Grafos
Busquedas
Complejidad: $O(n+m)$

Ultimamente los integrantes del Grupo de Algoritmia se han obsesionado con el milenario juego de ajedrez, pero tienen un inconviente, puesto que Moro siempre les gana, han decidido inventar un nuevo juego al que llamaron "Caballo sin Mecate".

El juego consiste en un tablero de ajedrez de tamaño $n \times m$ en el que interactúan dos jugadores. Dada una configuración de piezas tienes que llegar en la menor cantidad de pasos con el caballo de un punto a otro del tablero; gana el jugador que pueda dar la menor cantidad de movimientos. Además, se estipula que en el tablero sólo hay un caballo y múltiples piezas de otros tipos de las cuales no puede ocupar su lugar, pero sí saltarlas.

El caballo no se desplaza en línea recta, sino que tiene un movimiento característico llamado salto. El salto del caballo se parece a una L, y se compone de un desplazamiento de dos casillas en dirección horizontal o vertical, y otra casilla más en ángulo recto (en el diagrama siguiente, los cuadros marcados son los posibles lugares hacia donde puede desplazarse el caballo).



Tu tarea es ayudarnos a encontrar el número mínimo de movimientos que le tomará al caballo llegar a cierto punto, dada una configuración, para así poderle ganar a Moro.

*Édgar García Rodríguez - Grupo de Algoritmia Avanzada y Programación Competitiva

Entrada

La primer línea contendra un solo número $T(1 \leq T \leq 100)$ que representa la cantidad de casos de prueba, que se seguirá de T casos de prueba. La primer línea de cada caso contendrá dos números enteros n y $m(1 \leq n, m \leq 100)$. Seguirán n lineas, una línea por cada fila del tablero, donde cada línea tendrá m caracteres c_{ij} , que representan el estado del *escaque*[1] de la i-ésima fila y la j-ésima columna (numerados de abajo hacia arriba y de izquierda a derecha como se muestra en el diagrama anterior). Donde c_{ij} puede ser una casilla vacía (*), un peón (p) , una torre (t), un alfil (b), un rey (k), una reina (q) o un único caballo (c) para todo el tablero. La última línea de cada caso tendrá dos enteros x, y $(1 \leq x \leq n, 1 \leq y \leq m)$ que representan la fila y columna a las que quieres llegar.

Salida

Por cada caso deberás imprimir una sola línea que contendrá un entero que es la cantidad mínima de movimientos del caballo para llegar a su destino o -1 en caso de que sea imposible llegar a esa casilla.

Entrada Ejemplo

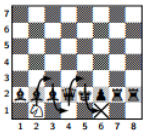
```
2
7 8
********
********
********
********
********
bbbqkptt
*c******
1 6
2 2
**
c*
2 2
```

Salida Ejemplo

```
4
-1
```

Notas

En el primer caso se puede llegar de la siguiente manera:



En el segundo caso no hay manera de llegar a la casilla (2, 2).

Entendiendo el problema

Debemos determinar el mínimo número de movimientos requeridos para que el caballo llegue a cierta pocisión del tablero, si es que ésto es posible.

Solución

El grafo es implícito y es el tablero de ajedrez, y decimos que un vértice (escaque) es vecino de otro si se puede llegar de uno a otro por medio de un movimiento de caballo, los vértices que tienen una pieza de ajedrez, que no es el caballo, en ellos no tienen vecinos. Para resolver el problema realizamos una búsqueda en amplitud desde la posición inicial del caballo contando los movimientos requeridos para llegar a cierto lugar, si en la búsqueda no se encuentra la posción deseada decimos que es imposible.

Código

```
1  import java.util.*;
2
3  class Cmain
4  {
5      static class pair
6      {
7          public int x, y;
8          public pair(int _x, int _y)
9          {
10             x = _x;
11             y = _y;
12         }
13     }
14
15     static int dX[] = {-2, -1, 1, 2, 2, 1, -1, -2}, dY[] = {1, 2, 2, 1, -1, -2,
       ↪  -2, -1};
16
17     public static void main(String[] args)
18     {
19         Scanner sc = new Scanner(System.in);
20         int tt = sc.nextInt();
21
22         while (tt-- > 0)
23         {
```

```
24              int n = sc.nextInt(), m = sc.nextInt();
25              String board[] = new String[n];
26
27              sc.nextLine();
28              for (int i = 0; i < n; ++i)
29                  board[i] = sc.nextLine();
30
31              int xf = sc.nextInt(), yf = sc.nextInt();
32              xf = n-xf;
33              --yf;
34
35              int dist[][] = new int[n][m];
36              Queue<pair> cola = new LinkedList<pair>();
37
38              for (int i = 0; i < n; ++i)
39                  for (int j = 0; j < m; ++j)
40                      if (board[i].charAt(j) == 'c')
41                      {
42                          cola.add(new pair(i, j));
43                          dist[i][j] = 1;
44                      }
45
46              while (!cola.isEmpty())
47              {
48                  pair act = cola.remove();
49
50                  if (act.x == xf && act.y == yf)
51                      break;
52
53                  for (int k = 0; k < 8; ++k)
54                  {
55                      int nx = act.x+dX[k], ny = act.y+dY[k];
56
57                      if (nx < 0 || nx >= n || ny < 0 || ny >= m ||
                        ↪  board[nx].charAt(ny) != '*' || dist[nx][ny] != 0)
58                          continue;
59
60                      cola.add(new pair(nx, ny));
61                      dist[nx][ny] = dist[act.x][act.y] + 1;
62                  }
63              }
64
65              System.out.println(dist[xf][yf]-1);
66          }
67      }
68  }
```

—D - Doncella Jean*

—Límite de tiempo: 1 segundo

Dificultad: -⋆
Temas: Ad Hoc
Complejidad: $O(n)$

En un lejano reino, existe una Doncella llamada Jean, es una chica muy bella pero también muy peculiar. Con lo anterior nos referimos a que tiene costumbres poco casuales. Para resumir, ella cuenta con muchos pretendientes en los reinos vecinos, incluyendo el suyo. Algunos de ellos son príncipes, duques y algunos otros con buen estatus social. No obstante hay un plebeyo que está enamorado de ella, para su fortuna (o no) es un "friend" de nuestra querida doncella, y como se imaginarán ambos tienen una amistad por Facebook.

Nuestro desdichado plebeyo (llamado Anselmo) últimamente ha tenido charlas muy frecuentes con la Doncella, él siente que varios de los mensajes de ella son indirectas pero no está muy seguro. Para serciorarse de ello ha seleccionado N mensajes y los ha marcado como "Le gusto" y "No le gusto". El piensa que si la proporción de "Le gusto" es significativamente mayor a la de "No le gusto" entonces decidirá si está en la friendzone o no. Para no complicarse define que si la proporción de "No le gusto" es $\geq 0,2$ entonces se encuentra en la friendzone.

Ayuda a Anselmo a determinar si se encuentra en la friendzone o no dados los N mensajes de las últimas conversaciones.

Entrada

La primer línea es un número C que denota la cantidad de casos de entrada. En los siguientes C bloques aparecerán un número $2 \leq N \leq 10000$ y N mensajes. Los mensajes serán "si" y "no" para denotar "Le gusto" y "No le gusto" respectivamente.

Salida

Para cada caso deberás imprimir como respuesta "friendzone" cuando $r \geq 0.2$ donde r denota la proporción de "No le gusto", en caso contrario la carita feliz ":)".

*Kenny Yahir Méndez Ramírez - Grupo de Algoritmia Avanzada y Programación Competitiva

Entrada Ejemplo

3
4
si
si
no
si
5
si
si
si
si
si
8
si
no
si
si
si
si
si
si

Salida Ejemplo

friendzone
:)
:)

Entendiendo el problema

En este caso, la solución es indirecta, se debe de contar cuantas veces aparece la cadena ya sea "si" o "no" y determinar su proporción.

Solución

En el caso en el que la proporción de "no" sea ≥ 0.2 se imprime `friendzone`, en caso contrario `:)`.

Código

```
1 #include <iostream>
2 #include <cstdio>
```

```cpp
#include <cstring>
#include <algorithm>
#include <cmath>
using namespace std;

#define For(x,a,b) for(unsigned int x = (a); x < (unsigned int)(b); x++)

typedef long long ll;


int main()
{
	int C, n, x;
	double r, tol = 0.2;
	string cad;

	scanf("%d",&C);

	while(C--)
	{
		scanf("%d", &n);
		x = 0;
		For(i,0,n){
			cin >> cad;
			x += cad == "no";
		}


		r = (double)x / (double)n;

		if(r < tol)
			printf(":)\n");
		else
			printf("friendzone\n");
	}

	return 0;
}
```

—E - Explanada*

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆⋆⋆
Temas: Matemáticas, Geometría
Complejidad: $O(n)$

Uno de tus amigos ha decido visitarte en la escuela, sin embargo él nunca ha venido a la FES y no sabe cómo encontrarte, por lo tanto has decidido ayudarle para que llegue a la explanada de MAC. La explanada la podemos ver como un polígono *convexo*[2] en el plano, y tú sabes muy bien las coordenadas de todos sus vértices, pues fue lo primero que te enseñaron en PIMAC. Para poder darle indicaciones a tu amigo necesitas saber si ya ha llegado a la explanada, así que le has pedido que te mande sus coordenadas. Ahora sólo necesitas hacer un programa que te diga si tu amigo está

Entrada

La primera línea contendrá un entero $1 \leq t \leq 100$, el número de casos de prueba. La primera línea de cada caso de prueba será un entero $3 \leq n \leq 100$ que denota el número de vértices del polígono que representa a la explanada.

En las siguientes n líneas de cada caso de prueba habrá dos enteros $0 \leq x_i, y_i \leq 100$ representando las coordenadas del i-ésimo vértice del polígono. Los puntos estarán dados en orden de tal forma que forme un polígono convexo (puede ir en el sentido de las manecillas del reloj o en sentido contrario a las manecillas del reloj). La última línea de cada caso de prueba serán dos enteros $0 \leq x_a, y_a \leq 100$, las coordenadas de tu amigo. El punto (x_a, y_a) no va a estar nunca sobre el perímetro del polígono.

Salida

Para cada caso de prueba deberás imprimir "si" (sin comillas) si tu amigo está dentro de la explanada, o "no" en caso contrario

Entrada Ejemplo

```
2
5
1 3
2 5
```

*David Felipe Castillo Velázquez - Grupo de Algoritmia Avanzada y Programación Competitiva

4 5
5 3
2 1
4 2
5
1 3
2 5
4 5
5 3
2 1
3 3

Salida Ejemplo

no
si

Entendiendo el problema

Dado un polígono convexo de n vértices ($p_0, p_1, \ldots, p_{n-1}$ en el orden de las manecillas del reloj o inverso), debemos determinar si un punto a se encuentra dentro del polígono.

Solución

Este un problema bien conocido de geometría computacional que se puede resolver con el Counter Clockwise Test. Sean $\overrightarrow{ap_i}$ y $\overrightarrow{ap_{i+1}}$ dos vectores obtenidos de estos 3 puntos. EL producto cruz de $\overrightarrow{ap_i} \times \overrightarrow{ap_{i+1}}$ en otro vector que es perpendicular a ambos. Si la magnitud del vector resultante es positiva/cero/negativa, entonces tenemos que a se encuentra a la derecha/es colineal/izquierda, respectivamente. Entonces si se cumple que para todo i el vector resultante $\overrightarrow{ap_i} \times \overrightarrow{ap_{(i+1) \mod n}}$ tiene el mismo signo, quiere decir, que el punto siempre estuvo del mismo lado de las aristas y por lo tanto se encuentra dentro del polígono convexo.

Código

```
1 #include <bits/stdc++.h>
2 #define For(i, a, b) for(int i=(a); i<(b); ++i)
3 #define INF 1000000000
4 #define MP make_pair
5 #define x first
6 #define y second
7
```

```
8  using namespace std;
9
10 typedef long long ll;
11 typedef pair <int, int> ii;
12
13 ii p[105];
14 bool ccw(ii A, ii B, ii C)
15 {
16         return (B.x-A.x)*(C.y-A.y) - (B.y-A.y)*(C.x-A.x) >= 0;
17 }
18
19 int main()
20 {
21         int tt;
22         scanf("%d", &tt);
23         while (tt--)
24         {
25                 int n;
26                 scanf("%d", &n);
27                 For(i, 0, n)
28                         scanf("%d %d", &p[i].x, &p[i].y);
29                 ii f;
30                 scanf("%d %d", &f.x, &f.y);
31                 bool sgn = ccw(f, p[0], p[1]), ok = true;
32                 For(i, 1, n)
33                         if (ccw(f, p[i], p[(i+1)%n]) != sgn)
34                         {
35                                 ok = false;
36                                 break;
37                         }
38                 printf("%s\n", ok ? "si" : "no");
39         }
40
41         return 0;
42 }
```

—F - Fiesta*

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆⋆
Temas: Matemáticas
Probabilidad
Complejidad: $O(n * m)$

Después del intenso concurso de programación de la XIV Semana de MAC tu equipo piensa dar una gran fiesta en tu casa para celebrar su aplastante triunfo sobre los demás equipos.

El problema es que para entonces tu cerebro ya no podrá trabajar muy bien, puesto que lo exprimiste al máximo en el concurso, y necesitas saber cuál es el valor esperado de la cantidad de personas que asistiran a tu fiesta, por lo que desde ahora harás un programa que te ayude en esta tarea.

Lamentablemente es probable que no todas las persona asistan a tu fiesta, porque algunas no resolvieron muchos problemas y no se sienten de ánimo.

Se han invitado n personas a tu fiesta y cada persona tiene una probabilidad $p_i(i = 1, 2, ..., n)$ de asistir, que está dada en función de su desempeño, es decir,

$$p_i = \sum_{j=1}^{m} \frac{r_{ij}}{m}$$

$$r_{ij} = \begin{cases} \frac{1}{a_{ij}} & \text{si} \quad a_{ij} \neq 0 \\ 0 & \text{si} \quad a_{ij} = 0 \end{cases}$$

donde a_{ij} es igual a la cantidad de intentos que le tomó a la persona i resolver el problema j, aij $= 0$ en caso de que la i-ésima persona no haya resuelto el j-ésimo problema, $(j = 1, 2, ..., m)$, y m es la cantidad de problemas en total .

*Édgar García Rodríguez - Grupo de Algoritmia Avanzada y Programación Competitiva

Entrada

La primer línea contendra un solo número $T(1 \leq T \leq 100)$ que representa la cantidad de casos de prueba.

La primer línea de cada caso contendrá dos números enteros n y $m(1 \leq n \leq 100, 1 \leq m \leq 11)$.

Seguirán n lineas, una línea por cada persona, donde la i-ésima línea tendrá m enteros $a_{ij}(0 \leq a_{ij} \leq 100)$ separados por un espacio.

Salida

Por cada caso deberás imprimir usa sola línea que contendrá el valor esperado de personas en tu fiesta redondeado a 2 decimales.

Entrada Ejemplo

```
2
5 3
1 0 1
1 1 1
0 19 2
11 0 2
3 0 7
4 7
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
1 1 1 1 1 1 1
```

Salida Ejemplo

```
2.21
4.00
```

Notas

En el primer caso la primer persona pudo resolver el primer y tercer problema en un intento, mientras que la segunda resolvió todos, por lo que sus probabilidades de ir son 0,66 y 1 respectivamente.

En el segundo caso todos resolvieron todos los problemas en un solo intento, por lo

que se espera que todos asistan.

Entendiendo el problema

Debemos encontrar el valor esperado del número de personas que asistirán a la fiesta, el cual está dado por la siguiente ecuación:

$$E(X) = \sum_{x=0}^{n} x f(x)$$

donde $f(x)$ es la probabilidad de que asistan x personas a la fiesta.

Solución

Obtner $f(x)$ es algo complicado, ya que para poder obtener esto debemos encontrar todos los subconjuntos A de tamaño x del conjunto $[n] = \{1, 2, \ldots, n\}$. Después considerando a $f(x) = 1$, decimos que para cada subconjunto A, si $i \in A$ entonces $f_A(x) = f_A(x) * p_i$ o en caso contrario $f_A(x) = f_A(x) * (1 - p_i)$. Y $f(x) = \sum f_A(x)$ Esta tarea puede sonar bastante complicada, aún así hemos probado que:

$$\begin{aligned} E(X) &= \sum_{i=1}^{n} p_i \\ &= \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{r_{ij}}{m} \\ &= \frac{1}{m} \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{1}{a_{ij}} \end{aligned}$$

Código

```cpp
#include <bits/stdc++.h>
using namespace std;
//look at my code my code is amazing
#define FOR(i, a, b) for (int i = int(a); i < int(b); i++)
#define FOREACH(it, a) for (typeof(a.begin()) it = (a).begin(); it != (a).end(); it++)
#define ROF(i, a, b) for (int i = int(a); i >= int(b); i--)
#define REP(i, a) for (int i = 0; i < int(a); i++)
#define INF 1000000000
#define INFLL 1000000000000000000LL
#define ALL(x) x.begin(), x.end()
#define MP(a, b) make_pair((a), (b))
#define X first
#define Y second
```

```
14 #define EPS 1e-9
15 #define DEBUG(x)   cerr << #x << ": " << x << " "
16 #define DEBUGLN(x) cerr << #x << ": " << x << " \n"
17 typedef pair<int, int> ii;
18 typedef vector<int> vi;
19 typedef long long ll;
20 typedef vector<bool> vb;
21 //ios_base::sync_with_stdio(0);//fast entrada/salida ;-)
22
23 void solve()
24 {
25     int n, m;
26     scanf("%d %d", &n, &m);
27     double ev = 0.0, p = 0.0;
28     REP(i, n)
29     {
30         p = 0.0;
31         REP(j, m)
32         {
33             int a;
34             scanf("%d", &a);
35             p += (a ? (1.0/a) : 0 );
36
37         }
38         ev += p/m;
39     }
40     printf("%.2lf\n", ev);
41 }
42
43 int main()
44 {
45     int T;
46     scanf("%d", &T);
47     REP(i, T)
48         solve();
49     return 0;
50 }
```

—G - Guiando a Edgar*

—Límite de tiempo: 1 segundo

Dificultad: ⋆⋆⋆
Temas: Teoría de Grafos **Complejidad:** $O(n^3)$

Edgar es una persona a la que le gusta optimizar su tiempo, por lo tanto cada que se tiene que desplazar de un lugar a otro dentro de la FES, le gusta irse por el camino que minimice el tiempo que le toma desplazarse. Además a él le gusta siempre pasar por lugares nuevos, porque asegura que eso ayuda a crear nuevas conexiones neuronales, entonces cada que planea la trayectoria para llegar de un lugar a otro, él considera un camino en específico por el que debe pasar obligatoriamente.

En este momento Edgar está demasiado ocupado encargándose del concurso, así que te ha pedido que le ayudes a organizar sus siguientes Q rutas para no tener que perder el tiempo.

Dado que la FES es muy grande, nos vamos a concentrar sólo en los N lugares favoritos de Edgar, numerados del 1 al N, con caminos entre cada uno de los N lugares. Los caminos entre cada par de lugares i, j, son bidireccionales, es decir, Edgar puede recorrer el camino de i a j o de j a i.

Entrada

La primera línea tendra un entero $1 \leq T \leq 100$, el número de casos de prueba.
La primer línea de cada caso de prueba será un entero $2 \leq N \leq 50$, los lugares favoritos en la FES de Edgar, seguido vendrán N líneas, con N enteros cada una. El j-ésimo valor de la i-ésima línea, $1 \leq w_{ij} \leq 1000$, indicará el tiempo que le toma a Edgar recorrer el camino del lugar i al lugar $j(w_{ij} = w_{ji})$. La $N + 2$ línea de cada caso de prueba será un entero $1 \leq Q \leq 100$, el número de rutas que tienes que planear para Edgar, seguido de Q líneas, cada una con cuatro enteros $1 \leq a, b, c, d \leq N$, indicando que Edgar quiere ir del lugar a al lugar b pasando obligatoriamente por el camino entre c y d (en cualquier sentido), a $6 = b, c6 = d$.

Salida

Para cada caso de prueba deberás imprimir Q enteros, representando el tiempo que el toma a Edgar realizar cada una de las rutas en el mismo orden en que fueron

*David Felipe Castillo Velázquez - Grupo de Algoritmia Avanzada y Programación Competitiva

solicitadas.

Entrada Ejemplo

```
1
5
0 9 5 2 1
9 0 5 2 2
5 5 0 2 6
2 2 2 0 7
1 2 6 7 0
2
5 1 5 2
5 1 2 3
```

Salida Ejemplo

```
5
11
```

Explicación de la entrada ejemplo: Para le primera ruta, Edgar puede tomar la siguiente sucesión de caminos: 5-2-5-1, con un tiempo total de 2+2+1=5.

Para la segunda ruta, Edgar puede tomar la siguiente sucesión de caminos: 5-2-3-4-1, con un tiempo total de 2+5+2+2=11.

Entendiendo el problema

Dado un grafo con $1 \leq n \leq 50$ vértices, y una serie de consultas, donde cada consulta se compone por dos vértices s, t y una arista $\{u, v\}$. Se te pide encontrar la distancia más corta de s a t pasando por la arista $\{u, v\}$.

Solución

Sea $d(a, b)$ la distancia mínima entre los vértices a y b. La solución es

$$\min\big(d(s,u) + w(u,v) + d(v,t), d(s,v) + w(u,v) + d(u,t)\big)$$

donde $w(u, v)$ es el peso de la arista $\{u, v\}$. Como vamos a procesar varias consultas, es necesario obtener todas las distancias mínimas, lo que podemos hacer con Floyd Warshall.

Código

```cpp
#include <bits/stdc++.h>

#define For(i, a, b) for(int i=(a); i<(b); ++i)
#define INF 1000000000
#define MP make_pair

using namespace std;

typedef long long ll;
typedef pair <int, int> ii;

int dist[55][55], AdjMat[55][55];

int main()
{
    //std::ios_base::sync_with_stdio(false);

    int tt;
    scanf("%d", &tt);

    while (tt--)
    {
        int n;
        scanf("%d", &n);

        For(i, 0, n)
            For(j, 0, n)
                scanf("%d", &AdjMat[i][j]);

        For(i, 0, n)
            For(j, 0, n)
                dist[i][j] = AdjMat[i][j];

        For(k, 0, n)
            For(i, 0, n)
                For(j, 0, n)
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);

        int Q;
        scanf("%d", &Q);
        while (Q--)
        {
            int a, b, c, d;
            scanf("%d %d %d %d", &a, &b, &c, &d);
            --a, --b, --c, --d;

            printf("%d\n", min(dist[a][c]+AdjMat[c][d]+dist[d][b],
             ↪  dist[a][d]+AdjMat[d][c]+dist[c][b]));
        }
```

```
49      }
50
51      return 0;
52  }
```

—H - Hurgando en el CEDETEC*

—Límite de tiempo: 1 segundo

Dificultad: $\star\star\star$
Temas: Matemáticas
Teoría de Números
Complejidad: $O(logn)$

Por fortuna el CEDETEC acaba de recibir presupuesto ilimitado y ha aumentado el número de cubículos desde $-\infty$ a ∞. El grupo de Algoritmia ha aprovechado la expansión del CEDETEC y se mudó de cubículo.

Lamentablemente Kenny no se enteró de la mudanza y ahora no sabe en qué cubículo quedó el grupo de Algoritmia, pero como es muy importante que lo encuentre decidió ir buscando de cubículo a cubículo, sin embargo resulta que Kenny quiere ir preguntando de una manera un poco peculiar: puede empezar a buscar en cualquier cubículo, una vez que haya buscado en el cubículo n sólo puede buscar en el cubículo $n \pm a$ o $n \pm b$.

Tu tarea consiste en saber si Kenny podrá algún día encontrar el cubículo de Algoritmia, sin importar a qué cubículo se mudó, conociendo los valores a y b que escogió Kenny.

Entrada

La primera línea será un entero $1 \leq T \leq 100$, indicando los números de casos de prueba. Para cada caso de prueba habrá una única línea con dos enteros $1 \leq a, b \leq 108$, explicados anteriormente.

Salida

Para cada caso de prueba debes imprimir "si" (sin las comillas) si Kenny podrá encontrar el cubículo de Algoritmia o "no" (sin las comillas), en caso contrario.

Entrada Ejemplo

```
2
3 4
2 4
```

*David Felipe Rodríguez Velázques, Moroni Silverio Flores - Grupo de Algoritmia Avanzada y Programación Competitiva

Salida Ejemplo

si
no

Entendiendo el problema

Para poder determinar si Kenny podrá algún día llegar al cubículo del CEDETEC o no, tenemos que saber si la combinación lineal $ax + by = d$ tiene solciones enteras para todo número. Suena complicado, aun así veremos que es una sencilla ecuación diofántica.

Solución

La ecuación $ax + by = d$ tiene soluciones enteras *si y solo si* el $\text{mcd}(a, b)$ divide a d, y como d puede ser cualquier número, el único número que divide a todos es el 1, entonces hay solución si $\text{mcd}(a, b) = 1$, o en otras palabras si a y b son *primos relativos*. Utilizamos entonces el algoritmo de Euclides podemos obtener la solución.

Código

```
#include <bits/stdc++.h>

#define For(i, a, b) for(int i=(a); i<(b); ++i)
#define INF 1000000000
#define MP make_pair

using namespace std;

typedef long long ll;
typedef pair <int, int> ii;

int gcd(int a, int b){ return b ? gcd(b, a % b) : a; };

int main()
{
    //std::ios_base::sync_with_stdio(false);

    int tt;
    scanf("%d", &tt);

    while (tt--)
    {
        int a, b;
        scanf("%d %d", &a, &b);
        printf("%s\n", gcd(a, b) == 1 ? "si" : "no");
    }

```

```
28      return 0;
29  }
```

References