// Guillermo Arriaga García Algoritmos. Tarea 2. Prob. 4: FICHAS DE DOMINÓ

/\*

**EL PROBLEMA ES EL SIGUIENTE:**

Problem D: Dominos 2

Dominos are lots of fun. Children like to stand the tiles on their side in long lines. When one domino falls, it knocks down the next one, which knocks down the one after that, all the way down the line. However, sometimes a domino fails to knock the next one down. In that case, we have to knock it down by

hand to get the dominos falling again.

Given a set of dominos that are knocked down by hand, your task is to determine

the total number of dominos that fall.

**Input Specification**

The first line of input contains one integer specifying the number of test cases to follow. Each test case begins with a line containing three integers n, m, l no larger than 10 000, followed by m+l additional lines. The first integer n is the number of domino tiles. The domino tiles are numbered from 1 to n. Each of the m lines after the first line contains two integers x and y indicating that if domino number x falls, it will cause domino number y to fall as well. Each of the following l lines contains a single integer z indicating that the domino numbered z is knocked over by hand.

**Output Specification**

For each test case, output a line containing one integer, the total number of dominos that fall over.

**Sample Input**

1

3 2 1

1 2

2 3

2

**Output for Sample Input**

2

\*/

/\*

**SE PIDE** SOLUCIONARLO usando una búsqueda en profundidad (Depth-First) en el grafo que queda al tener los n nodos y las aristas dadas por las m líneas que indican que si cae x entonces cae y.

La búsqueda en profundidad comenzaría en las fichas indicadas por las l líneas siguientes a las m (quienes siguen al caso prueba) y si una ficha al caer tira otra ficha que ya estaba caida entonces ya no es necesario inspeccionar pues ya fueron contadas las caídas.

**ESTRATEGIA DE SOLUCIÓN:**

\* Hacer un arreglo de n+1 elementos. El elemento de la posición cero guardará la cantidad de fichas caídas en cada prueba. Las demás posiciones se inicializarán a 0 indicando que no se han caído.

\* Otro arreglo será de listas, tendrá n+1 elementos y cada elemento (lista) indicará las fichas que caerán si él cae.

\* La búsqueda en profundidad se realizará de la siguiente manera:

Si iniciamos en la casilla 2 la marcamos como caída, contamos que ha caído una más a las que llevábamos, nos movemos a las fichas que tira la 2 (si es que hay alguna). Repetimos este proceso hasta no avanzar. Y hacemos backtracking hasta tirar todas las que el 2 provoca que caigan.

Luego, nos vamos a la siguiente ficha que se cae manualmente. Si no está tirada realizamos el procedimiento anterior, si está tirada nos vamos a la siguiente, si existe, hasta acabar con las fichas a tirar con la mano, entonces terminamos con la prueba e imprimimos las fichas que en total se cayeron.

\*/

#include <iostream>

#include <list>

#include <stack>

using namespace std;

**int main()**

{

int p,n=0,m,l,x,y,k;

int \*A; // A[i]=0 Ficha i arriba

list<int> \*C; // C = conexiones

stack<int> ruta;

list<int>::iterator it;

// Con un arreglo de listas es posible hacer

// búsqueda en profundidad de cualquier grafo.

// Inicialización para generalizar el siguiente bucle

A=new int[n];

C=new list<int>[1];

cin>>p; // Casos prueba

for(int a=0;a<p;a++)

{

**// CAPTURA DE DATOS DE CADA CASO PRUEBA**

delete [] A;

delete [] C;

ruta.empty();

cin>>n>>m>>l; // n fichas, m enlaces, l empujes

A=new int[n+1]; // Creación del grafo/arreglo

C=new list<int>[n+1];

for(int i=0;i<=n;i++)

{ A[i]=0; } // indica que no ha caído, A[0] contará las caídas

for(int b=0;b<m;b++) // Captura de conexiones

{ cin>>x>>y;

C[x].push\_back(y);

}

for(int b=0;b<l;b++) // Captura de las fichas que se van a tirar a mano.

{ cin>>k;

ruta.push(k);

}

**// COMIENZA EL CONTEO DE FICHAS QUE SE CAERÁN EN ESTE CASO PRUEBA**

// Recorrido en profundidad con backtracking

while(!ruta.empty())

{ k=ruta.top();

ruta.pop(); // Saca al último que se metió (profundidad)

// Mete a la ruta a todos los que se caen con él y no han caído,

// si han sido tirados antes.

if(A[k]==0 && C[k].size()!=0)

{

for ( it=C[k].begin(); it!=C[k].end(); it++)

{ if(A[\*it]==0)

{ ruta.push(\*it); }

}

}

if(A[k]==0) // Si está parado lo tira.

{ A[0]++;

A[k]=1;

}

}

**// IMPRESION DE LAS FICHAS TIRADAS EN ESTE CASO PRUEBA**

cout<<A[0]<<"\n\n";

}

delete [] A;

delete [] C;

ruta.empty();

// system("PAUSE"); // Opcional

return 0;

}