Número de Erdős

Es de sobra conocido que no hay mascota más altiva, fría y malévola que el gato. Prueba de ello es que es el único animal (con la salvedad, claro está, del humano) con la saña y la mala idea necesarias para torturar a sus víctimas antes de zampárselas. Tú, pobre ratoncillo, has caído en las garras de este temible felino, pero sus malvados juegos pueden ser tu salvación. El gato te ha encerrado en un laberinto donde controla solo algunas de las salas. Las salas del laberinto están conectadas entre ellas por puertas que solo se abren en una dirección. En las salas que controla el gato, solo encontrarás abierta la puerta que escoja el gato en ese momento. En las demás, siempre podrás escoger el camino. Determina si puedes escapar, o si, mareado de tanto buscar una escapatoria, caerás presa del gato. Las salas se representarán con números indexados sobre 0, y partes de la sala 0 y deseas llegar a la sala 1.

Estrategia de la solución

Este problema es una aplicación directa del algoritmo de BFS. La única cosa delicada es como representar adecuadamente el grafo. En la solución dada, se han representado directamente los nodos con los nombres y se han usado maps para acceder a ellos, pero otra posibilidad consiste en traducir al principio los nombres a enteros y trabajar entonces con la representación habitual.

Solución en Python

```
ncases = int(input())

for case in range(ncases):
    narticles = int(input())

    # El grafo G es un diccionario de listas de strings
    G = {'erdos' : []}
    # erdosNum es un diccionario de ints
    erdosNum = {'erdos' : -1}

    for article in range(narticles):
        nauthors = int(input())
        auth_list = []

        for author in range(nauthors):
                author_name = input()
                erdosNum[author_name] = -1

                for auth in auth_list:
                        # Este snippet intenta añadir y si es el primer elemento creado
                        # en G[auth] inicializa el array
                        try:
                                G[auth].append(author_name)
                        except KeyError:
                                G[auth] = [author_name]
```

```
                        try:
                                G[author_name].append(auth)
                        except KeyError:
                                G[author_name] = [auth]

                auth_list.append(author_name)

    Q = []
    Q.append( ('erdos', 0) )
    erdosNum['erdos'] = 0

    # Iniciar BFS
    while (len(Q) != 0):
        node = Q.pop(0)

        if node[1] != erdosNum[node[0]]:
                break

        for neig in G[node[0]]:
                if erdosNum[neig] == -1:
                        Q.append((neig, node[1]+1))
                        erdosNum[neig] = node[1]+1

    # Ordenar lexicograficamente
    erdosNum = sorted(erdosNum.items(), key=lambda kv: kv[0])

    for node in erdosNum:
        print(node[0], node[1])
```

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

Solución en C++

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

using namespace std;

int main() {
    int z;
    cin >> z;
    while (z--) {
        int n;
        cin >> n;
        map<string, vector<string> > G;

        for (int i = 0; i < n; ++i) {
            int a;
            cin >> a;
            vector<string> p;
            for (int j= 0; j < a; ++j) {
                string s;
```

```
            cin >> s;
            if (!G.count(s)) G[s] = vector<string>();
            for (string x : p){
                G[x].push_back(s);
                G[s].push_back(x);
            }
            p.push_back(s);
        }
    }

    map<string, int> dist;
    if (G.count("erdos")) {
        queue<string> q;
        q.push("erdos");
        dist["erdos"] = 0;
        while (!q.empty()) {
            string x = q.front();
            q.pop();
            for (string v : G[x]) {
                if (!dist.count(v)) {
                    dist[v] = dist[x] + 1;
                    q.push(v);
                }
            }
        }
    }

    for (auto p : G){
        cout << p.first << " " << (dist.count(p.first) ? dist[p.first] : -1) << endl;
    }
  }
}
```

Autor de la solución: Miquel Ortega (Miembro del comité organizador)

Talar un árbol

Un leñador quiere talar un árbol pero solo puede mover trozos de tamaño *m*. Sin embargo, quiere obtener el mínimo número de trozos. Sin embargo, es un leñador un tanto peculiar porque el árbol que quiere talar es un grafo, es decir, quiere separar un grafo conexo sin ciclos en distintas componentes conexas de tamaño menor que *m*. Escribe un programa que dado un árbol encuentre este mínimo número.

Estrategia de la solución

En este problema, la observación clave consistía en ver que se podía conseguir la configuración óptima de un árbol encontrando la configuración óptima de los subárboles y uniendo el nodo a tantos subárboles como fuera posible, es decir, uniéndolo a los que les quedara el trozo de arriba más pequeño (la demostración de esta propiedad se deja como ejercicio). Entonces tenía una implementación natural como dfs sobre el árbol.

Solución en Python

```
G = [[]]
n, m = 0, 0
cnt = 0

def dfs(node, par):
    sz = []
    for v in G[node]:
        if v != par: sz.append(dfs(v, node))

    sz = sorted(sz)
    global cnt # Fuerza a que cnt sea una variable global
    res = 1
    for i in range(len(sz)):
        if res + sz[i] <= m: res += sz[i]
        else: cnt = cnt+1
    return res

ncases = int(input())
for case in range(ncases):
    line1 = input() # linea inutil
    line1 = input().split(" ")
    n, m = int(line1[0]), int(line1[1])

    # El grafo G es una lista de n listas de ints
    G = [[] for i in range(n)]

    for i in range(n-1):
        line2 = input().split(" ")
        u, v = int(line2[0]), int(line2[1])

        G[u].append(v)
        G[v].append(u)
```

```
    cnt = 0
    dfs(0, 0)
    print(cnt+1)
```

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

Solución en C++

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

typedef vector<int> VI;
typedef vector<VI> VVI;
VVI t;
int cnt, m;

int dfs(int x, int par){
    VI sz;
    for (int v : t[x]){
        if (v != par){
            sz.push_back(dfs(v, x));
        }
    }
    sort (sz.begin(), sz.end());
    int res = 1;
    for (int i =0; i < sz.size(); ++i){
        if (res + sz[i] <= m) res += sz[i];
        else cnt++;
    }
    return res;
}

int main() {
    int z;
    cin >> z;
    while (z--) {
        int n;
        cin >> n >> m;
        t = VVI(n);
        cnt = 0;
        for (int i = 0; i < n-1; ++i) {
            int u, v;
            cin >> u >> v;
            t[u].push_back(v);  t[v].push_back(u);
        }
        dfs(0, 0);
        cout << cnt+1 << endl;
    }
}
```

Autor de la solución: Miquel Ortega (Miembro del comité organizador)

El gato y el ratón

Paul Erdős fue un matemático húngaro famoso por su prolífica contribución a las matemáticas. Llegó a trabajar con tantos autores que se bautizó en su honor el número de Erdős, una métrica que mide a qué distancia colaborativa un matemático cualquiera se encuentra del portentoso húngaro. Concretamente, el número de Erdős es el mínimo número de saltos que hay que dar entre autores que han publicado juntos para llegar a Erdős. Si no se puede llegar, diremos que es -1. El problema consiste en encontrar el número de Erdős de los matemáticos sugeridos, dada una lista de todos los artículos publicados. En la entrada del problema se escribirá Erdős como erdos.

Estrategia de la solución

La solución esperada de este problema consistía en hacer un BFS ligeramente modificado desde el nodo 1 (el objetivo) para ver si se podía llegar desde el punto de partida. Se mantiene la invariante que los nodos a los que se llega con el BFS son nodos desde donde se puede llegar a 1. Los nodos controlados por el ratón se comportan de forma habitual, pero los nodos controlados por el gato solo se pueden añadir a la cola cuando ya se han visitado todos las otras opciones que tiene para salir de ese nodo (es decir, cuando el gato ya no puede escoger y está forzado a coger un nodo que puede llegar a 1).

Solución en C++

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

using namespace std;

typedef vector<int> VI;
typedef vector<VI> VVI;

int main() {
    int z;
    cin >> z;
    while (z--) {
        int n, m;
        cin >> n >> m;
        VVI G(n);
        vector<bool> cat(n, false);
        for (int i =0; i < n; ++i) {
            int a;
            cin >> a;
            cat[i] = a;
        }

        VI f(n, 0);
        for (int i = 0; i < m; ++i) {
            int u, v;
            cin >> u >> v;
```

```
            G[v].push_back(u);
            if (cat[u]) ++f[u];
        }

        queue<int> q;
        q.push(1);
        vector<bool> vis(n, false);
        vis[1] = true;

        while (!q.empty() and !vis[0]) {
            int x = q.front();
            q.pop();
            for (int v : G[x]) {
                if (!vis[v]) {
                    --f[v];
                    if (f[v] <= 0) {
                        vis[v] = true;
                        q.push(v);
                    }
                }
            }
        }
        cout << vis[0] << endl;
    }
}
```

Autor de la solución: Miquel Ortega (Miembro del comité organizador)