Soluciones OIFem 2020

Apoyo Grafos

Build a Graph

Teoría

- Qué es un grafo y terminología básica

Solución

El mínimo número de nodos en una componente conexa donde se cumple que todos tengan un grado de 0 o 1 y todos los nodos sean nodos de corte es 6. Por lo tanto, se necesitan 12 nodos para que se cumpla esta característica y sean dos o más componentes conexas.

Código

C++

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

int main() {
    int n;
    cin >> n;
    if (n < 12)
        cout << "No";
    else
        cout << "Yes";
    return 0;
}
```

Treasures in a map (1)

Teoría

- DFS

Solución

Consideraremos que cada cuadrado del tablero es un nodo y que este está conectado a través de aristas con los nodos inmediatamente pegados a él verticalmente y horizontalmente. Esto se llama un grafo implícito, o sea, una estructura que se puede simplificar si actuamos sobre ella como si fuera un grafo.

Iniciamos el DFS en la casilla que nos dice el usuario, visitando sus vecinos, los vecinos de estos vecinos... La diferencia entre este DFS y un DFS normal es que el índice del nodo tiene dos coordenadas: su fila y su columna. Por lo demás, el DFS es igual que el de una dimensión. Marcamos en un vector de booleanos las casillas ya visitadas e incrementamos un contador cada vez que vemos un tesoro. Cuando llegamos a una casilla marcada con una 'X', retornamos, ya que no podemos viajar a través de esa casilla.

Una vez termine el DFS de ejecutarse, imprimiremos yes si hay algún tesoro encontrado y no en caso contrario.

Código

C++

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

int filas, columnas, tesoros;
vector<vector<bool>> visitado;
vector<string> grid;

void dfs(int f, int c) {
        if (f < 0 || f == filas || c < 0 || c == columnas)
            return; // esto significa que la coordenada está fuera del tablero
        if (visitado[f][c]) return; // si volvemos a visitar nodos en los que ya
        ↪  hemos estado entraremos en un bucle infinito
        visitado[f][c] = true;
        if (grid[f][c] == 't')
            tesoros++; // incrementamos el contador del número de tesoros que
            ↪  hemos visto
        if (grid[f][c] == 'X') return;
        // nos movemos hacia arriba
        dfs(f-1, c);
        // nos movemos hacia abajo
        dfs(f+1, c);
        // nos movemos hacia la derecha
        dfs(f, c+1);
        // nos movemos hacia la izquierda
        dfs(f, c-1);
}
```

```cpp
int main() {
        cin >> filas >> columnas;
        grid = vector<string>(filas);
        visitado.assign(filas, vector<bool>(columnas, false));
        for (int i = 0; i < filas; i++)
                cin >> grid[i];
        int initF, initC;
        cin >> initF >> initC;
        tesoros = 0;
        dfs(initF-1, initC-1);
        if (tesoros) cout << "yes\n";
        else cout << "no\n";
        return 0;
}
```

Treasures in a map (2)

Teoría

- BFS

Solución

Para este ejercicio, nos aprovecharemos de que BFS opera en orden de capas, es decir, que visita los nodos en orden de lejanía de la raíz. Por lo tanto, modelamos el tablero como un grafo implícito, como en la pregunta anterior, esta vez usando BFS en vez de DFS. La estrategia de guardar distancias es la misma que usamos en clase para el problema de Erdös, es decir, la distancia de un nodo de la raíz es el mínimo de las distancias de la raíz de sus vecinos más uno, ya que hay que recorrer un cuadrado (una arista) más. Guardaremos las distancias en un vector y, al encontrarnos una X, no recorreremos esa casilla.

Como sabemos que el primer tesoro que nos encontremos será el más cercano a la raíz por el uso de BFS, retornaremos su distancia. Si no hay ningún tesoro retornaremos 10^9, ya que no habrá ningún tesoro a tal distancia.

Código

C++

```cpp
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

int filas, columnas;
vector<string> grid;

int minimaDistancia(int initF, int initC) {
        queue<pair<int, int>> cola;
        vector<vector<int>> distancia(filas, vector<int>(columnas, 1e9));
        distancia[initF][initC] = 0;
        cola.push(make_pair(initF, initC));
        int f, c;
        while(!cola.empty()) {
                f = cola.front().first;
                c = cola.front().second;
                cola.pop();
                if (grid[f][c] == 'X') continue;
                if (grid[f][c] == 't')
                        return distancia[f][c];
                if (f != 0 && distancia[f-1][c] > distancia[f][c]+1) {
                    // nos movemos hacia arriba
                        distancia[f-1][c] = distancia[f][c]+1;
                        cola.push(make_pair(f-1, c));
                }
```

```
                if (f != filas-1 && distancia[f+1][c] > distancia[f][c]+1) {
                    // nos movemos hacia abajo
                        distancia[f+1][c] = distancia[f][c]+1;
                        cola.push(make_pair(f+1, c));
                }
                if (c != 0 && distancia[f][c-1] > distancia[f][c]+1) {
                    // nos movemos hacia la izquierda
                        distancia[f][c-1] = distancia[f][c]+1;
                        cola.push(make_pair(f, c-1));
                }
                if (c != columnas-1 && distancia[f][c+1] > distancia[f][c]+1) {
                    // nos movemos hacia la derecha
                        distancia[f][c+1] = distancia[f][c]+1;
                        cola.push(make_pair(f, c+1));
                }
        }
        return 1e9;
}

int main() {
        int minDist, initF, initC;
        cin >> filas >> columnas;
        grid = vector<string>(filas);
        for (int i = 0; i < filas; i++)
                cin >> grid[i];
        cin >> initF >> initC;
        minDist = minimaDistancia(initF-1, initC-1);
        if (minDist == 1e9) cout << "no treasure can be reached\n";
        else cout << "minimum distance: " << minDist << '\n';
        return 0;
}
```

Monk in the Real Estate

Teoría

- Terminología básica de grafos
- Conjuntos desordenados

Solución

Esta era una pregunta sencilla para ver terminología de grafos pero repasando lo aprendido la semana pasada. Para resolverlo, bastaba con guardar cuántos nodos diferentes compraba el monje, es decir, para cada arista $u - v$, guardar u y v en un conjunto desordenado y, al terminar de leer la entrada, imprimir el tamaño del conjunto. Era importante acordarnos de vaciar el conjunto entre casos.

Código

C++

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

int main(){
        unordered_set<int> conjunto;
        int T, M, u, v;
        cin >> T;
        while(T--) {
                conjunto.clear();
                cin >> M;
                while(M--) {
                        cin >> u >> v;
                        conjunto.insert(u);
                        conjunto.insert(v);
                }
                cout << conjunto.size() << '\n';
        }
}
```