

Programación Competitiva

Problemas Semana 13/05/2020 grafos bfs

Franco Quiroga
Javier Zavala
Marco Herrera

May 16, 2020

1 Tutorial

1.1 Problema 131D Subway

Se nos presenta un esquema de metro con n ($3 \leq n \leq 3000$), siendo n el número de estaciones y trenes en el esquema. Adicionalmente se tiene los pares x_i, y_i ($1 \leq x_i, y_i \leq n$), estos pares indican la presencia de un pasaje entre la estación x_i y la y_i , además sabemos que siempre $x_i \neq y_i$. Se nos menciona que siempre se forma un anillo vial que no contiene una estación más de una vez. El problema como tal consiste en dado un orden para los nodos del esquema con sus respectivas conexiones, obtener la distancia de los nodos al anillo, siendo esta 0 si forman parte de el.

Para la solución que se presentará, se consideró usar el algoritmo dfs (búsqueda en profundidad) como se sugiere en uno de los *tags*, con la intención de buscar ciclos. La función básicamente recibe un nodo, su padre y la profundidad.

Está bien lo que explique entonces o no? :p Primero, explicamos cómo opera DFS (Depth-First Search). El algoritmo empieza con un nodo arbitrario, que llamaremos *raíz*. Desde la raíz tomamos los nodos vecinos en algún orden y a cada nodo: - lo registramos como visitado en alguna estructura de datos; y, - aplicamos el mismo proceso. Cuando se llegue a un nodo sin vecinos (aparte del padre), el algoritmo se "contrae", y se retornan los nodos encontrados en el camino al padre. Esto visitará al menos 1 vez los nodos de cualquier grafo.

Con este algoritmo, podemos encontrar distancias: tomamos el nodo origen como *raíz*, y a cada paso del algoritmo podemos contar un nodo más visitado. Cuando se halle el nodo objetivo, "contraemos" el algoritmo, y la *profundidad* a la que llegó el algoritmo corresponde a la distancia entre los nodos.

Este algoritmo tiene la ventaja de que además encuentra *anillos* en el grafo. Como se garantiza que hay solo 1 anillo, si aplicamos el algoritmo a partir de cualquier nodo, llegaremos a un punto en el que se visita un nodo una 2da vez: el algoritmo llega al anillo en algún punto (o parte en él), y lo recorre entero hasta volver al mismo nodo.

Ya habiendo implementado el algoritmo y habiendo inicializado las distancias de cada nodo al anillo, aplico dfs a todos los nodos del vector.

2 Solución

2.1 Código Solución

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int n;
5 int steps;
6 const int maxn=3e3+3;
7 vector<int> v[maxn];
8 int dist[maxn];
9 int dfs(int node, int par, int depth){
10     dist[node]=depth;
11     for(int i=0;i<v[node].size();i++){
12         int next=v[node][i];
13         if(next==par){
14             continue;
15         }
16         if(dist[next]){
17             return dist[next];
18         }
19         int steps=dfs(next,node,depth+1);
20         if(steps){
21             return steps;
22         }
23     }
24     return 0;
25 }
26
27 int main(){
28     int x,y;
29     cin >> n;
30     for(int i=1;i<=n;i++){
31         cin >> x >> y;
32         v[x].push_back(y);
33         v[y].push_back(x);
34     }
35     for(int i=1;i<=n;i++){
36         for(int j=1;j<=n;j++){
37             dist[j]=0;
38         }
39         cout << dfs(i,-1,1)-1 << "\t";
40     }
41     cout << endl;
42 }
```

2.2 Link Codeforces

<https://codeforces.com/contest/131/submission/79995864>

2.3 Link Git

<https://gitlab.labcomp.cl/mherrera/inf-349>