Grafos
Professor Wladimir

Pedagio

Fonte: https://br.spoj.com/problems/PEDAGIO/

Juquinha ganhou uma viagem para a Coreia do Sul com a família. Seu pai, o Sr. Juca, quer visitar outras cidades dirigindo, mas com limite de gastos com pedágios. Como cada estrada entre cidades exige o pagamento de um pedágio fixo por sentido, o objetivo é listar todas as cidades diferentes da atual que podem ser alcançadas com no máximo P pedágios pagos, considerando somente a ida.

Estratégia de solução

- Modelar as cidades como vértices de um grafo não direcionado.
- Modelar cada estrada como uma aresta com custo 1 (um pedágio).
- Realizar busca em largura (BFS) a partir da cidade L, limitando a profundidade da busca em P e atualizando a distância de cada vértice até L.
- Percorre os vértices seguindo a ordem e checando se a distância P (exceto o nó de origem).

Estrutura de Dados

A estrutura de dados é um vetor de listas de adjacência para representar um grafo não direcionado com N vértices.

Essa estrutura permite acesso rápido aos vizinhos de qualquer vértice — ideal para algoritmos como BFS e DFS com complexidade O(—d(v)—) ou seja proporcional ao grau do vértice v.

Complexidade

- Cada vértice tem uma lista de adjacência, que contém os vizinhos.
- Quando visitamos um vértice, percorremos sua lista de vizinhos uma única vez.
- Ao longo da execução completa do BFS, cada aresta é examinada no máximo duas vezes (uma vez para cada ponta da aresta, em grafos não direcionados).
- $\sum_{v \in V(G)} d(v) = 2|E|$ Lema do aperto de mão https://en.wikipedia.org/wiki/Handshaking_lemma
- Então o custo total de percorrer todas essas listas é O(—E—).
- Complexidade do algoritmo O(—V—+—E—).

Código

```cpp
#include <stdio.h>
#include <iostream>
#include <queue>
using namespace std;
int main() {
    int N , M;
    int L, P;
    int teste = 1;
    while( cin >> N >> M >> L >> P ){
        if(N+M+L+P == 0) break;
        vector <int> adj[N];
        for(int i = 0; i < M; i++){
            int a, b;
            cin >> a >> b;
            a--;
            b--;
            adj[a].push_back(b);
            adj[b].push_back(a);
        }
        vector <int> dist;
        dist.assign(N, -1);
        L--;
        queue<int> q;
        q.push(L);
        dist[L] = 0;
        while( !q.empty() ){
            int u = q.front();
            q.pop();
            if( dist[u] >= P ) break;
            //printf("tirando da fila %d dist %d\n", u, dist[u]);
            for(int v : adj[u]){
                if(dist[v] == -1){
                    dist[v] = dist[u] + 1;
                    if( dist[v] < P){
                        q.push(v);
                    }
                }
            }
        }
        if(teste > 1) printf("\n");
        cout << "Teste " << teste << endl;
        for(int i = 0; i < N; i++){
            if(i != L && dist[i] != -1)
                printf("%d ", i+1);
        }
        printf("\n");
        teste++;
    }
    return 0;
}
```

MESA - Mesa da Sra Montagny!

Fonte: https://br.spoj.com/problems/MESA/

A socialite Sra. Montagny quer organizar jantares em que os convidados fiquem sentados frente a frente com todos os seus amigos. Para isso, ela coleta, de cada convidado, uma lista de amigos que gostariam de ter sentados em frente.

Objetivo do problema

Verificar se é possível organizar os convidados em uma única mesa retangular (com dois lados opostos) de forma que todos os amigos de cada pessoa estejam do lado oposto da mesa (isto é, cada pessoa e seus amigos devem estar em lados opostos).

Modelagem

Isso equivale a verificar se o grafo não direcionado construído pelas relações de amizade é bipartido — ou seja, se é possível dividir os vértices (convidados) em dois conjuntos tais que não haja arestas (amizades) dentro de um mesmo conjunto.

Código

```cpp
#include <vector>
#include <queue>
#include <stdio.h>
using namespace std;
typedef enum{
    CINZA, BRANCO, PRETO
} tipo_cor;
/* Função que testa se um subgrafo é bipartido*/
int bfs(vector <int> adj[], vector <tipo_cor> cor, int s){
        queue <int> q;
        cor[s] = BRANCO;
        q.push(s);
        while( !q.empty() ){
            int u = q.front();
            q.pop();
            for(int v : adj[u]){
                if(cor[v] == CINZA){
                    if( cor[u] == BRANCO) cor[v] = PRETO;
                    else cor[v] = BRANCO;
                    q.push(v);
                }else{
                    if(cor[v] == cor[u]) return 0;
                }
            }
        }
        return 1;
}
int main(){
    int N, M, teste = 1;
    while( scanf("%d %d", &N, &M) > 0){
        vector <int> adj[N];
        for(int i = 0; i < M; i++){
            int a, b;
            scanf("%d %d", &a, &b);
            a--;
            b--;
            adj[a].push_back(b);
            adj[b].push_back(a);
        }
        vector <tipo_cor> cor;
        cor.assign(N, CINZA);
        bool bipartido = true;
        for(int i = 0; i < N; i++){
            if(cor[i] == CINZA){
                if ( bfs(adj, cor, i) == 0){
                    bipartido = false;
                    break;
                }
            }
        }
        if(teste > 1) printf("\n");
        printf("Instancia %d\n", teste++);
        if(bipartido) printf("sim\n");
        else printf("nao\n");
    }
}
```

Exercícios

1. https://br.spoj.com/problems/DENGUE/ (Encontrar o centro de um grafo)
2. https://br.spoj.com/problems/ENERGIA/ (Checar se um grafo é conexo)
3. https://br.spoj.com/problems/DUENDE/