Caminho Mínimo
Professor Wladimir

Problema do Caminho Mínimo

Dado um grafo direcionado com pesos positivos, o objetivo é encontrar o **menor custo** para ir de um vértice origem (`src`) até um vértice destino (`dst`).

Estruturas principais:

1. dist[v]: menor distância conhecida de src até v.
2. pai[v]: pai (ou antecessor) do vértice v no caminho mínimo.
3. pq: fila de prioridade que guarda os pares (distância, vértice).

Etapas principais:

1. Inicializa distâncias como ”infinito”(1e9) e pais como -1.
2. Define dist[src] = 0 e insere src na fila de prioridade.
3. Enquanto a fila não estiver vazia:
   - Remove o vértice com menor distância atual.
   - Relaxa as arestas: para cada vizinho v, se $dist[v] > dist[u] + w$, atualiza dist[v] e pai[v].
4. Quando o destino for alcançado ou a fila esvaziar, termina.

```cpp
vector <int> dist(n+1, 1e9);
vector <int> pai(n+1, -1);
priority_queue <ii, vector<ii>, greater<ii>> pq;
dist[src] = 0;
pq.push({0, src});
while( !pq.empty() ){
    auto [currDist, node] = pq.top();
    pq.pop();
    if(node == dst) break;
    if(dist[node] < currDist) continue;
    for(auto & [v, w] : g[node] ){
        if( dist[v] > currDist + w){
            dist[v] = currDist + w;
            pai[v]  = node;
            pq.push({dist[v], v});
        }
    }
}

if( dist[dst] == 1e9) printf("-1\n");
else {
    print_path(pai, src, dst);
    printf("\n");
    printf("%d\n", dist[dst]);
}
```

Árvore de Caminho Mínimo

Durante a execução do algoritmo, é mantido um vetor `pai` que indica, para cada vértice, qual foi o antecessor responsável por alcançar esse vértice com menor custo. Isso permite reconstruir o caminho completo de `src` até `dst` e forma uma **árvore de caminhos mínimos** enraizada em `src`.

Função de Impressão do Caminho

A função abaixo imprime o caminho mínimo de forma recursiva:

```
void print_path(vector<int> &pai, int src, int dst){
    if(dst == src){
        printf("%d", src);
    }else{
        print_path(pai, src, pai[dst]);
        printf("->%d", dst);
    }
}
```

Resumo

- O algoritmo de Dijkstra resolve o problema do caminho mínimo com pesos não negativos.
- Utiliza fila de prioridade para otimizar a seleção do próximo vértice.
- Constrói uma árvore de caminhos mínimos a partir do vértice origem.
- Permite reconstruir e imprimir o caminho mínimo completo até qualquer vértice alcançável.
- No pior caso, cada um dos n vértices é removido da fila de prioridade uma vez, com custo $\mathcal{O}(\log n)$ por remoção, totalizando $\mathcal{O}(n \log n)$. Além disso, cada uma das m arestas pode ser relaxada no máximo uma vez, também com custo $\mathcal{O}(\log n)$ devido à inserção na fila, resultando em $\mathcal{O}(m \log n)$ no total.
- Assim, a complexidade total do algoritmo é $\mathcal{O}((n + m) \log n)$

Engarrafamento

Antônio enfrenta engarrafamentos diários para ir ao trabalho e descobre que a secretaria de tráfego fornece um mapa da cidade com os tempos médios de travessia de cada rua durante o horário de pico. Usando esses dados, ele calcula o caminho mais rápido de sua casa até o trabalho. Percebendo o valor dessas informações, ele decide criar (ou delegar a criação de) um programa para indicar o menor tempo de trajeto entre dois pontos da cidade.

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,int> ii;
int main(){
    int n, m;
    while( scanf("%d %d", &n, &m) ){
        if(n+m==0) break;
        vector <ii> g[n+1];
        for(int i = 1; i <= m; i++){
            int a, b, w;
            scanf("%d %d %d", &a, &b, &w);
            g[a].push_back({b,w});
        }
        int src, dst;
        scanf("%d %d", &src, &dst);
        vector <int> dist(n+1, 1e9);
        priority_queue <ii, vector<ii>, greater<ii>> pq;
        dist[src] = 0;
        pq.push({0, src});
        while( !pq.empty() ){
            auto [currDist, node] = pq.top();
            pq.pop();
            //printf("visitando %d dist %d\n", node, currDist);
            if(node == dst) break;
            if(dist[node] < currDist) continue;
            for(auto & [v, w] : g[node] ){
                if( dist[v] > currDist + w){
                    dist[v] = currDist + w;
                    //printf("vizinho %d dist %d\n", v, dist[v]);
                    pq.push({dist[v], v});
                }
            }
        }
        if( dist[dst] == 1e9) printf("-1\n");
        else printf("%d\n", dist[dst]);
    }
}
```

Exercícios

1. https://olimpiada.ic.unicamp.br/pratique/p2/2009/f1/pontes/
2. https://br.spoj.com/problems/QUASEMEN/
3. https://www.beecrowd.com.br/repository/UOJ_1391.html
4. https://www.beecrowd.com.br/repository/UOJ_2768.html
5. https://br.spoj.com/problems/DESVIO/
6. https://br.spoj.com/problems/MINIMO/
7. https://leetcode.com/problems/path-with-maximum-probability/description/?envType=problem-list-v2&envId=shortest-path
8. https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/practice-problems/algorithm/routes-48c6192a/