Caminhos mínimos

Algoritmo de Dijkstra: problemas resolvidos

Prof. Edson Alves - UnB/FGA

2019

Sumário

UVA 1112 – Mice and Maze

Problema

A set of laboratory mice is being trained to escape a maze. The maze is made up of cells, and each cell is connected to some other cells. However, there are obstacles in the passage between cells and therefore there is a time penalty to overcome the passage Also, some passages allow mice to go one-way, but not the other way round.

Suppose that all mice are now trained and, when placed in an arbitrary cell in the maze, take a path that leads them to the exit cell in minimum time.

Problema

We are going to conduct the following experiment: a mouse is placed in each cell of the maze and a count-down timer is started. When the timer stops we count the number of mice out of the maze.

Write a program that, given a description of the maze and the time limit, predicts the number of mice that will exit the maze. Assume that there are no bottlenecks is the maze, i.e. that all cells have room for an arbitrary number of mice.

Entrada e saída

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The maze cells are numbered $1, 2, \ldots, N$, where N is the total number of cells. You can assume that $N \leq 100$.

The first three input lines contain N, the number of cells in the maze, E, the number of the exit cell, and the starting value T for the count-down timer (in some arbitrary time unit).

Entrada e saída

The fourth line contains the number M of connections in the maze, and is followed by M lines, each specifying a connection with three integer numbers: two cell numbers a and b (in the range $1, \ldots, N$) and the number of time units it takes to travel from a to b.

Notice that each connection is one-way, i.e., the mice can't travel from b to a unless there is another line specifying that passage. Notice also that the time required to travel in each direction might be different.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

The output consists of a single line with the number of mice that reached the exit cell E in at most T time units.

Exemplo de entradas e saídas

Sample Input

```
1

4
2
1
8
1 2 1
1 3 1
2 1 1
2 4 1
3 1 1
3 4 1
4 2 1
4 3 1
```

Sample Output

```
3
```

Solução com complexidade $O(T(V + E \log E))$

- O problema consiste em determinar o caminho mínimo de todos os vértices do grafo até a saída E
- O custo de cada caminho mínimmo pode ser obtido através do algoritmo de Dijkstra
- Contudo, ao invés de executar o algoritmo $N - 1$ vezes, é mais eficiente inverter todas as arestas do grafo, e computar todos os caminhos a partir de E
- Deste modo, é necessário executar o algoritmo de Dijkstra uma única vez, de modo que a solução tem complexidade $O(T(V + E \log E))$, onde T é o número de casos de teste
- Observe que um rato é posto também na saída, de modo que a resposta sempre será maior ou igual a um

Solução com complexidade $O(T(V + E \log E))$

```cpp
#include <bits/stdc++.h>

using namespace std;
using ll = long long;
using ii = pair<ll, ll>;

const ll MAX { 1010 }, oo { 1000000000000000010LL };
ll dist[MAX];
vector<ii> adj[MAX];
bitset<MAX> processed;

void dijkstra(int s, int N)
{
    for (int i = 1; i <= N; ++i)
        dist[i] = oo;

    dist[s] = 0;
    processed.reset();

    priority_queue<ii, vector<ii>, greater<ii>> pq;
    pq.push(ii(0, s));
```

Solução com complexidade $O(T(V + E\log E))$

```cpp
    while (not pq.empty()) {
        auto p = pq.top();
        pq.pop();

        auto d = p.first, u = p.second;

        if (processed[u])
            continue;

        processed[u] = true;

        for (const auto& q : adj[u]) {
            auto v = q.first, w = q.second;

            if (dist[v] > d + w) {
                dist[v] = d + w;
                pq.push(ii(dist[v], v));
            }
        }
    }
}
```

Solução com complexidade $O(T(V + E \log E))$

```

int solve(int N, int E, int T)
{
    dijkstra(E, N);

    auto ans = 0;

    for (int i = 1; i <= N; ++i)
        ans += (dist[i] <= T ? 1 : 0);

   return ans;
}

int main()
{
    ios::sync_with_stdio(false);

    int C;
    cin >> C;

```

Solução com complexidade $O(T(V + E \log E))$

```
    for (int test = 0; test < C; ++test)
    {
        int N, E, T;
        cin >> N >> E >> T;

        for (int i = 0; i < MAX; ++i)
            adj[i].clear();

        int M;
        cin >> M;

        while (M--)
        {
            int u, v, w;
            cin >> u >> v >> w;

            adj[v].push_back(ii(u, w));
        }

        auto ans = solve(N, E, T);

```

Solução com complexidade $O(T(V + E \log E))$

```
        if (test)
            cout << '\n';

        cout << ans << '\n';
    }

    return 0;
}
```

Codeforces Alpha Round #20 (Codeforces format) – Problem C: Dijkstra?

Problema

You are given a weighted undirected graph. The vertices are enumerated from 1 to n. Your task is to find the shortest path between the vertex 1 and the vertex n.

Entrada e saída

Input

The first line contains two integers n and m $(2 \leq n \leq 10^5, 0 \leq m \leq 10^5)$, where n is the number of vertices and m is the number of edges. Following m lines contain one edge each in form a_i, b_i and w_i $(1 \leq a_i, b_i \leq n, 1 \leq w_i \leq 10^6)$, where a_i, b_i are edge endpoints and w_i is the length of the edge.

It is possible that the graph has loops and multiple edges between pair of vertices.

Output

Write the only integer -1 in case of no path. Write the shortest path in opposite case. If there are many solutions, print any of them.

Exemplo de entradas e saídas

Sample Input

```
5 6
1 2 2
2 5 5
2 3 4
1 4 1
4 3 3
3 5 1

5 6
1 2 2
2 5 5
2 3 4
1 4 1
4 3 3
3 5 1
```

Sample Output

```
1 4 3 5




1 4 3 5
```

Solução com complexidade $O(N + M \log M)$

- O nome do problema efetivamente aponta a solução
- O algoritmo de Dijkstra pode ser utilizado tanto para computar o caminho mínimo quanto para reconstruir este caminho
- É preciso atentar ao fato de que a distância mínima pode exceder a capacidade de um inteiro, sendo necessário o uso de variáveis do tipo `long long`
- Também é preciso ter cuidado com o tratamento dos casos onde não há caminho de 1 a N
- Como o número de vértices é grande, o algoritmo de Bellman-Ford leva ao TLE

Solução AC com complexidade $O(N \log N)$

```cpp
#include <bits/stdc++.h>

using namespace std;
using ll = long long;
using ii = pair<ll, ll>;

const ll MAX { 100010 }, oo { 1000000000000000010LL };
ll dist[MAX], pred[MAX];
vector<ii> adj[MAX];
bitset<MAX> processed;

void dijkstra(int s, int N)
{
    for (int i = 1; i <= N; ++i) {
        dist[i] = oo;
        pred[i] = -1;
    }

    dist[s] = 0;
    pred[s] = s;
    processed.reset();
```

Solução AC com complexidade $O(N \log N)$

```cpp
    priority_queue<ii, vector<ii>, greater<ii>> pq;
    pq.push(ii(0, s));

    while (not pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();

        if (processed[u])
            continue;

        processed[u] = true;

        for (const auto& [v, w] : adj[u]) {
            if (dist[v] > d + w) {
                dist[v] = d + w;
                pq.push(ii(dist[v], v));
                pred[v] = u;
            }
        }
    }
}
```

Solução AC com complexidade $O(N \log N)$

```cpp
vector<int> solve(int N)
{
    dijkstra(1, N);

    vector<int> path;
    auto p = N;

    while (p != 1) {
        path.push_back(p);
        p = pred[p];

        if (p < 0)
            return vector<int> {};
    }

    path.push_back(1);
    reverse(path.begin(), path.end());

    return path;
}

```

Solução AC com complexidade $O(N \log N)$

```cpp
int main()
{
    ios::sync_with_stdio(false);

    int N, M;
    cin >> N >> M;

    while (M--)
    {
        int u, v, w;
        cin >> u >> v >> w;

        adj[u].push_back(ii(v, w));
        adj[v].push_back(ii(u, w));
    }

    auto ans = solve(N);

```

Solução AC com complexidade $O(N \log N)$

```
    if (ans.empty())
        cout << -1 << '\n';
    else
        for (size_t i = 0; i < ans.size(); ++i)
            cout << ans[i] << (i + 1 == ans.size() ? "\n" : " ");

    return 0;
}
```

Referências

1. UVA 1112 – Mice and Maze
2. Codeforces Alpha Round #20 (Codeforces format) – Problem C: Dijkstra?