Finding a negative cycle in the graph

Table of Contents

You are given a directed weighted graph G with N vertices and M edges. Find any cycle of negative weight in it, if such a cycle exists.

In another formulation of the problem you have to find all pairs of vertices which have a path of arbitrarily small weight between them.

It is convenient to use different algorithms to solve these two variations of the problem, so we'll discuss both of them here.

Using Bellman-Ford algorithm

Bellman-Ford algorithm allows you to check whether there exists a cycle of negative weight in the graph, and

if it does, find one of these cycles.

The details of the algorithm are described in the article on the Bellman-Ford algorithm. Here we'll describe only its application to this problem.

Do N iterations of Bellman-Ford algorithm. If there were no changes on the last iteration, there is no cycle of negative weight in the graph. Otherwise take a vertex the distance to which has changed, and go from it via its ancestors until a cycle is found. This cycle will be the desired cycle of negative weight.

Implementation

```
struct Edge {
    int a, b, cost;
};

int n, m;
vector<Edge> edges;
const int INF = 1000000000;

void solve()
{
    vector<int> d(n);
    vector<int> p(n, -1);
    int x;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (Edge e : edges) {
            if (d[e.a] + e.cost < d[e.b]) {
```

```
                d[e.b] = d[e.a] + e.cost;
                p[e.b] = e.a;
                x = e.b;
            }
        }
    }

    if (x == -1) {
        cout << "No negative cycle found.";
    } else {
        for (int i = 0; i < n; ++i)
            x = p[x];

        vector<int> cycle;
        for (int v = x;; v = p[v]) {
            cycle.push_back(v);
            if (v == x && cycle.size() > 1)
                break;
        }
        reverse(cycle.begin(), cycle.end());

        cout << "Negative cycle: ";
        for (int v : cycle)
            cout << v << ' ';
        cout << endl;
    }
}
```

Using Floyd-Warshall algorithm

The Floyd-Warshall algorithm allows to solve the second variation of the problem - finding all pairs of vertices

(i, j) which don't have a shortest path between them (i.e. a path of arbitrarily small weight exists).

Again, the details can be found in the Floyd-Warshall article, and here we describe only its application.

Run Floyd-Warshall algorithm on the graph. Initially $d[v][v] = 0$ for each v. But after running the algorithm $d[v][v]$ will be smaller than 0 if there exists a negative length path from v to v. We can use this to also find all pairs of vertices that don't have a shortest path between them. We iterate over all pairs of vertices (i, j) and for each pair we check whether they have a shortest path between them. To do this try all possibilities for an intermediate vertex t. (i, j) doesn't have a shortest path, if one of the intermediate vertices t has $d[t][t] < 0$ (i.e. t is part of a cycle of negative weight), t can be reached from i and j can be reached from t. Then the path from i to j can have arbitrarily small weight. We will denote this with `-INF`.

Implementation

```
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j) {
        for (int t = 0; t < n; ++t) {
            if (d[i][t] < INF && d[t][t] < 0 &
                d[i][j] = - INF;
        }
    }
}
```

Practice Problems

- UVA: Wormholes
- SPOJ: Alice in Amsterdam, I mean Wonderland
- SPOJ: Johnsons Algorithm