Last update: July 17, 2022 Translated From: e-maxx.ru

Solving assignment problem using min-cost-flow

The **assignment problem** has two equivalent statements:

- Given a square matrix $A[1..N, 1..N]$, you need to select N elements in it so that exactly one element is selected in each row and column, and the sum of the values of these elements is the smallest.
- There are N orders and N machines. The cost of manufacturing on each machine is known for each order. Only one order can be performed on each machine. It is required to assign all orders to the machines so that the total cost is minimized.

Here we will consider the solution of the problem based on the algorithm for finding the minimum cost flow (min-cost-flow), solving the assignment problem in $\mathcal{O}(N^3)$.

Description

Let's build a bipartite network: there is a source S, a drain T, in the first part there are N vertices (corresponding to rows of the matrix, or orders), in the second there are also N vertices (corresponding to the columns of the matrix, or machines). Between each vertex i of the first set and each vertex j of the second set, we draw an edge with bandwidth 1 and cost A_{ij}. From the source S we draw edges to all vertices i of the first set with bandwidth 1 and cost 0. We draw an edge with bandwidth 1 and cost 0 from each vertex of the second set j to the drain T.

We find in the resulting network the maximum flow of the minimum cost. Obviously, the value of the flow will be N. Further, for each vertex i of the first segment there is exactly one vertex j of the second segment, such that the flow F_{ij} = 1. Finally, this is a one-to-one correspondence between the vertices of the first segment and the vertices of the second part, which is the solution to the problem (since the found flow has a minimal cost, then the sum of the costs of the selected edges will be the lowest possible, which is the optimality criterion).

The complexity of this solution of the assignment problem depends on the algorithm by which the search for the maximum flow of the minimum cost is performed. The complexity will be $\mathcal{O}(N^3)$ using Dijkstra or $\mathcal{O}(N^4)$ using Bellman-Ford. This is due to the fact that the flow is of size $O(N)$ and each iteration of Dijkstra algorithm can be performed in $O(N^2)$, while it is $O(N^3)$ for Bellman-Ford.

Implementation

The implementation given here is long, it can probably be significantly reduced. It uses the SPFA algorithm for finding shortest paths.

```
const int INF = 1000 * 1000 * 1000;

vector<int> assignment(vector<vector<int>> a) {
    int n = a.size();
    int m = n * 2 + 2;
    vector<vector<int>> f(m, vector<int>(m));
    int s = m - 2, t = m - 1;
    int cost = 0;
    while (true) {
        vector<int> dist(m, INF);
        vector<int> p(m);
        vector<bool> inq(m, false);
        queue<int> q;
        dist[s] = 0;
        p[s] = -1;
        q.push(s);
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            inq[v] = false;
            if (v == s) {
                for (int i = 0; i < n; ++i) {
                    if (f[s][i] == 0) {
                        dist[i] = 0;
                        p[i] = s;
                        inq[i] = true;
                        q.push(i);
                    }
                }
            } else {
                if (v < n) {
                    for (int j = n; j < n + n; ++j) {
                        if (f[v][j] < 1 && dist[j] > dist[v] + a[v][j - n]) {
                            dist[j] = dist[v] + a[v][j - n];
                            p[j] = v;
                            if (!inq[j]) {
                                q.push(j);
                                inq[j] = true;
                            }
                        }
                    }
                } else {
                    for (int j = 0; j < n; ++j) {
                        if (f[v][j] < 0 && dist[j] > dist[v] - a[j][v - n]) {
                            dist[j] = dist[v] - a[j][v - n];
                            p[j] = v;
                            if (!inq[j]) {
                                q.push(j);
                                inq[j] = true;
                            }
                        }
                    }
                }
            }
        }
```

```
        int curcost = INF;
        for (int i = n; i < n + n; ++i) {
            if (f[i][t] == 0 && dist[i] < curcost) {
                curcost = dist[i];
                p[t] = i;
            }
        }
        if (curcost == INF)
            break;
        cost += curcost;
        for (int cur = t; cur != -1; cur = p[cur]) {
            int prev = p[cur];
            if (prev != -1)
                f[cur][prev] = -(f[prev][cur] = 1);
        }
    }

    vector<int> answer(n);
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (f[i][j + n] == 1)
                answer[i] = j;
        }
    }
    return answer;
}
```