Opracowanie: j/
Problem J: Routing

HISTORIA:
• wersja 1.0: 2007, Tomasz Idziaszek

dokument systemu SINOL 1.3.1

1 Problem analysis

Let us rephase the problem in graph theory language. We are given a **directed graph** $G = (V, E)$, which has n nodes. We have two special nodes in G: the start node s and the destination node t. We want to find two **directed paths** in this graph: one from s to t and one in another direction. In other words we want paths $v_0 \to v_1 \to \cdots \to v_{k-1} \to v_k$ and $\bar{v}_0 \to \bar{v}_1 \to \cdots \to \bar{v}_{l-1} \to \bar{v}_l$ and $v_0 = \bar{v}_l = s$, $v_k = \bar{v}_0 = t$. We are interested in such paths that visit minimal number of nodes, that is we want to minimize the size of the set $\{v_0, \ldots, v_k, \bar{v}_0, \ldots, \bar{v}_l\}$.

2 Solution

Firstly, we introduce a **distance function** $\delta\colon V \times V \to \mathbb{N}$. For every $u, v \in V$ we have $\delta(u, v) = \infty$ if there is no directed path in G from u to v. Otherwise $\delta(u, v) = k$ if such paths exist and the shortest one has length k (it means that it has k edges).

For a proposition P the Iverson bracket $[P]$ is defined as follows: $[P] = 1$ if P is true, $[P] = 0$ otherwise.

We define another directed graph $G_2 = (V \times V, E_2)$ and **weight function** $w\colon E_2 \to \mathbb{N}$. They are constructed as follows:

- for every edge $(u, v) \in E$ and every node $v_0 \in V$ we add to E_2 an **edge of the first kind** $e = ((u, v_0), (v, v_0))$ and put $w(e) = [v \neq v_0]$;
- for every edge $(v, u) \in E$ (notice that u and v are swithed) and every node $v_0 \in V$ we add to E_2 an **edge of the second kind** $e = ((v_0, u), (v_0, v))$ and put $w(e) = [v \neq v_0]$;
- for every pair of different nodes $u, v \in V$ if $\delta(u, v) \neq \infty$ we add **edge of the third kind** $e = ((u, v), (v, u))$ to E_2 and put $w(e) = \delta(u, v) - 1$.

Our solution is the length of the shortest path in G_2 from (s, s) to (t, t) with respect to weight function w.

3 Proof of correctness

Let's say that we have the two paths (one from s to t and other from t to s) that are the solution for our problem. Let $a_1, a_2, \ldots$ be the set of vertices that these two paths has in common, and in the order of apperance in the first path.

If a is empty (paths don't have any common vertice except the beginning and the end) then it's trivial to prove that our solution is equal to the shortest path from (s, s) to (t, t) in graf G_2. We simply use egdes of the first kind to travel from (s, s) to (t, s) with the cost equal do the length of the first path (from s to t) and then we travel from (t, s) to (t, t) using edges of the second type.

Now we have two possibilities: either a_1 is the last vertice from the set in the second path, or there exists such k that a_k is placed after a_1 in the second path, and a_{k+1} is not.

In the first case, we can divide our problem into two separate problems where values of s, t will be accordingly (s, a_1) and a_1, t. There is a solution for the first part that has no common points, so our thesis is correct there. And for the second part we use our proof again (checking if there is any common point etc.)

In the second case we'll have to use the edges of the third kind. First it's easy to see that edges $(a_1, a_2), (a_2, a_3), \ldots (a_{k-1}, a_k)$ belong to both paths. Moreover on the second path there are no vertices from a set between the vertice a_{k+1} (it may be equal to t) and a_1. So again we can divide our problem into two subproblems where s, t are equal s, a_{k+1} and a_{k+1}, t.

Now we have to prove that the shortest path in G_2 between (s, s) and (a_{k+1}, a_{k+1}) is a correct solution for the first subproblem. The shortest path in G_2 will be equal to shortest path from (s, s) to (a_1, s) using edges of the first kind, then to (a_1, a_k) using edges of the second kind. Then we will use the edge of the third kind to go from (a_1, a_k) to (a_k, a_1) and then edges of the first kind to (a_{k+1}, a_1) and edges of the second kind to finally reach (a_{k+1}, a_{k+1}).

The edges of the first and second kind correspond to moving along the first and second path. Edge of the third kind is moving along both paths at the same time along the same edges. So if there exist a path from (a, b) to (c, d) of cost x it means that there are two paths, one from a to c and second from d to b such that the total number of vertices they use is equal to x.

So we have proven that for each path in G_2 there are corresponding paths in G and that the length shortest path in G_2 is not greater than the correct solution for G. Having all these proofs, we have just proven that the length of the shortest path in G_2 is equal to the solution in G.

4 Algorithm

After reading graph G from the input we construct it in memory together with its transposition G^T. Then we calculate function δ for every pair of nodes from G. We can do it using **Floyd-Warshall algorithm** in time $O(n^3)$.

Then we run **Dijkstra algorithm** on graph G_2 and weight function w. Note, than we will construct graph G_2 on the fly. Every time we have popped node (u_1, u_2) from priority queue we iterate over adjacency list of u_1 to update edges of the first kind, then we iterate over adjacency list of u_2 in transpose graph G^T to update edges of the second kind and finally we check whether $u_1 \neq u_2$ and then we update an edge of the third kind.

Graph G_2 has $O(n^2)$ nodes. Every node is an origin for $O(n)$ edges of the first and second kind and for at most one edge of the third kind. Therefore we have $O(n^3)$ edges in G_2. If we implement priority queue in Dijkstra algorithm using a heap, total time complexity will be $O(n^3 \log n)$.

This algorithm was implemented in C++ and in Java (programs `j.cpp` and `j.java`).

5 Wrong solutions

Implementing priority queue in Dijkstra algorithm using an array gives an $O(n^4)$ algorithm, which is too slow.

A common mistake is to implement the algorithm without edges of the third kind.

Also algorithms which fix a shortest path in graph G and then find the second path, are incorrect.

6 Tests

First few tests are the correctness tests.

Test no 2 is a simple loop.

Test 3 checks if program uses the edges of the third kind

Test 4 also checks the edges of the third type and eliminates the algorithms that try to find two shortest paths.

Test 5 is a big test with almost full graph of size 50.