Lab07-Network Flow

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2019.

∗ If there is any problem, please contact TA Mingran Peng.

∗ Name:田雪飞 Student ID: 515030910347 Email:13487426939@qq,com

1. **Solution.** The following is the process of solution.
The **way(1)** to solve this problem: at first, we can find the maximum distance from one vertex s to other vertexes by SPFA. then traversal all the vertexes to find maximum distance in the network.

• Find maximum distance in all vertexes.

Algorithm 1: Maximum distance.

Input: All the vertexes of G(V,E)
Output: The maximum distance in the network.

$maximum \leftarrow 0$;
for ***each*** $u \in$ ***V*** **do**
    $maximum \leftarrow SPFA(u)$;
return $maximum$;

• Find the maximum distance from s(start vertex) to other vertex.

Algorithm 2: SPFA(s) algorithm(maximum).

Input: Computer s(start), and an undirected Graph $G = (V, E)$ represent the relation of computer(connected(delay i) and unconnected(assume delay 0)). vertex s,t ϵ **V**.
Output: The maximum time needed to send message other computers by vertex s.

$max(s) \leftarrow 0$; **for** ***each*** $u \in$ ***V*** **do**
    $DIST(u) \leftarrow 0$;
    $in\_queue[i] \leftarrow false$;
$Q.PUSH(s)$;(Using a queue Q to do SPFA.)
$in\_queue[s] \leftarrow true$;(s in queue)
while *Q is not empty* **do**
    $u \leftarrow Q.POP()$;
    $in\_queue[u] \leftarrow false$;(u out of Q)
    for ***each*** *(u,v)* ϵ ***E*** **do**
        if $DIST[v] < DIST[u] + t_i(u, v)$ **then**
            $DIST[v] \leftarrow DIST[u] + t_i(u, v)$;
            if $DIST[v] > max(s)$ **then**
                $max \leftarrow DIST[v]$;
            if *v is not in queue* **then**
                $Q.PUSH(v)$;
                $in\_queue[v] \leftarrow true$;(v in queue)
return *max(s)*

Time complexity:
Best case: if all the vertexes are pushed into once by using SPFA,then SFPA time complexity is o(V+E), the time time complexity of finding the maximum distance in network is o(V(V+E)).
Worst case: we assume: there are n vertexes$v_0, v_1, v_2, v_3 \ldots v_n$,at first we push v_0 into the queue, when we pop v_0, in worst case, we should update the $v_1, v_2, \ldots v_n$,when we pop v_1, we

should update $v_2, v_3 \ldots v_n$,and so on. in this case,SPFA time complexity is becoming o(VE). So the time time complexity of finding the maximum distance in network is $o(EV^2)$.

The **way(2)** to solve this problem: we can use Floyd-Warshall Algorithm.

Algorithm 3: Floyd-Warshall Algorithm

Input: An undirected Graph $G = (V, E)$ represent the relation of computer(connected(delay i) and unconnected(assume delay 0)). vertex s,t ϵ **V**

Output: The maximum distance in the network.

```
for each v ϵ V do
    for each u ϵ V do
        if (v,u) ϵ E then
            EDGE[v][u] ← (−ti(v, u));(use negative to find minimal number)
        else
            EDGE[v][u] ← 0;
min ← 0;
for  k ← 1  to n do
    for i ← 1 to n do
        for j ← 1 to n do
            if EDGE[i][j] > EDGE[i][k] + EDGE[k][j] then
                EDGE[i][j] ← EDGE[i][k] + EDGE[k][j];
for i ← 1 to n do
    for j ← 1 to n do
        if EDGE[i][j] < min then
            min ← EDGE[i][j];
maximum ← (−min);
return maximum;
```

Time complexity: Obviously, Floyd-Warshall Algorithm time complexity is $o(n^3)$.

□

2. **Solution.** The following is the process of solution.
ab. In this case,because there exist the cost which is negative, i will use SPFA to solve this problem. and in this problem,we also should know whether there exist a negative circle. So i use an array to count the times of all vertexes are pushed into queue.If there exist a vertex which is pushed into queue more than n times, there must exist a circle in G(V,E).

Algorithm 4: SPFA(s) algorithm.

Input: Computer s(start) and e(end), and a directed Graph $G = (V, E)$ represent the cost $w_i(u, v)$ from one vertex to another vertex. vertex s,e ϵ **V**.
Output: The minimum cost from s to e.

```
for each u ϵ V do
    DIST(u) ← ∞;
    in_queue[u] ← false;
    COUNT[u] = 0;(count the times of vertex is pushed into the queue)
Q.PUSH(s);(Using a queue Q to do SPFA.)
COUNT[s] ← COUNT[s] + 1;
in_queue[s] ← true;(s in queue)
while Q is not empty do
    u ← Q.POP();
    in_queue[u] ← false;(u out of Q)
    for each (u,v) ϵ E do
        if DIST[v] > DIST[u] + t_i(u, v) then
            DIST[v] ← DIST[u] + t_i(u, v);
            if v is not in queue then
                Q.PUSH(v);
                in_queue[v] ← true;(v in queue)
                COUNT[s] ← COUNT[s] + 1;
                if COUNT[s] ≥ n then
                    return There exist a negative circle.
return DIST[e];
```

Time complexity:
Best case: if all the vertexes are pushed into once by using SPFA,time complexity is o(V+E).
Worst case: we assume: there are n vertexes $v_0, v_1, v_2, v_3 \dots v_n$,at first we push v_0 into the queue, when we pop v_0, in worst case, we should update the $v_1, v_2, \dots v_n$,when we pop v_1, we should update $v_2, v_3 \dots v_n$,and so on. in this case, SPFA time complexity is becoming o(VE).

We also can solve this problem by using Bellman-Ford Algorithm.The input and output are same as SPFA.

Algorithm 5: Bellman-Ford Algorithm.

for *each* $v \in V$ **do**
  $DIST[v] \leftarrow +\infty$
  ;
$DIST[s] \leftarrow 0$
for $i \leftarrow 1$ *to* n **do**
  for *each* $e(u,v) \in E$ **do**
    if $DIST[v] > DIST[u] + t_i(u,v)$ **then**
      $DIST[v] = DIST[u] + t_i(u,v)$;
for *each* $v \in V$ **do**
  if $DIST[v] > DIST[u] + t_i(u,v)$ **then**
    return *False*;
return $DIST[e]$

Time complexity of Bellman-Ford is $o(VE)$.

□

3. (Bonus)

In this problem, we can treat slots,lessons,and professors as nodes, construct edges according to professors' preference, then the question is obvious becoming a maximum problem, so,we can use the Max-Flow to solve this problem.

So, we use blue, red, green nodes to represent slots, professors, lessons, and the flow of two nodes is 1; then we can use Max-Flow strategy to solve this problem.

Then we can use Fort-Fulkerson Algorithm. If there exists augmenting path in the residual graph G_f, then we update residual graph G_f, repeat until there have no augmenting path.

To update the residual graph, we can use DFS to find a path from s to t, meanwhile we can remember this path so that we can update G_f.

The following graph(example) is to show the relation of professors ,slots and lessons.

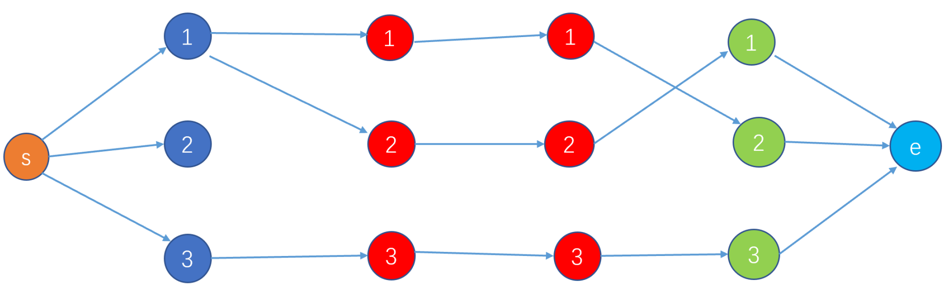

Figure 1: The flow of two nodes is 1

Remark: You need to include your .pdf and .tex files in your uploaded .rar or .zip file.