Lab07-Network Flow

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

∗ If there is any problem, please contact TA Mingran Peng.
∗ Name: Yitian Ma Student ID: 517021910467 Email: mkochab@sjtu.edu.cn

1. Remember the network problems in last lab?

   Consider there is a network consists n computers. For some pairs of computers, a wire i exists in the pair, which means these two computers can communicate with delay t_i.

   The distance of two computers are defined as their communication delay. Design an algorithm to compute the maximum distance in the network.

   You need to provide the pseudo code and analyze the time complexity.

   Solution. For simplicity, if a wire exists in the pair of computer u and computer v, the delay can be denoted as $t_{(u,v)}$, the value of which is exactly t_i. Let's use d_{uv} to denote the distance between computer u and v. Obviously if $u = v$, we have $d_{uv} = t_{(u,v)} = 0$.

   Here is the pseudo code for the algorithm.

Algorithm 1: Maximum distance in the network

Input: Graph $G = (V, E)$; delay $t_{(u,v)}$ for edge (u, v)
Output: DIST_{max}, the maximum distance in the network

```
for u ← 1 to |V| do
    for v ← 1 to |V| do
        if no wire between u and v then
            d_uv ← ∞;
        else
            d_uv ← t_(u,v);
for k ← 1 to |V| do
    for u ← 1 to |V| do
        for v ← 1 to |V| do
            if d_uv > d_uk + d_kv then
                d_uv ← d_uk + d_kv;
DIST_max ← ∞;
for u ← 1 to |V| do
    for v ← 1 to |V| do
        if DIST_max < d_uv then
            DIST_max ← d_uv;
return DIST_max
```

Time Complexity: In the initialization part the time complexity is $O(|V|^2)$. The updating of the distance of each pair takes $O(|V|^3)$ time. The search for the maximum distance costs $O(|V|^2)$ time. The total time complexity is clearly $O(|V|^3)$. □

2. Suppose you are traveling through a country defined as a directed graph $G = (V, E)$. You start from vertex s and want to go to e. For each $i \in E$, there is a w_i regarding the cost for traveling via i. Some times $w_i < 0$ which means you can earn money by traveling.(Do not ask why.) Here you need to design an algorithm satisfying the following demands:

   (a) Find the minimum cost from s to e. The problem guarantee that there is a path from s to e.

   (b) Indicate whether there is a circle in G, that by traveling through this circle you can earn money continually.

   You need to provide the pseudo code and analyze the time complexity.

   (a) **Solution.** We use $w_{(u,v)}$ to denote the weight of the edge from vertex u to vertex v. Here is the pseudo code for the algorithm.

Algorithm 2: Minimum cost between two vertices

Input: Directed graph $G = (V, E)$; start vertex s; destination vertex e
Output: *cost*, the minimum cost from s to e

```
1 for each u ∈ V do
2     C(u) ← ∞;
3 C(s) ← 0;
4 for i ← 1 to |V| − 1 do
5     for each vertex v ∈ V do
6         if C(v) has been updated in previous iteration then
7             for each edge(u, v) ∈ E do
8                 C(u) ← min{C(u), C(v) + w_(u,v)};
9 return cost ← C(e)
```

   Time Complexity: The outer loop runs for $|V| - 1$ times. Each loop contains at most $|E|$ updates. The total time complexity is $O(|E| \cdot |V|)$. □

   (b) **Solution.** Here is the pseudo code for the algorithm.

Algorithm 3: Detect negative cycle

Input: Directed graph $G = (V, E)$; start vertex s
Output: $flag$, whether there is a circle in G, that by traveling through this circle you can earn money continually

```
1  for each u ∈ V do
2      C(u) ← ∞;
3  C(s) ← 0;
4  for i ← 1 to |V| − 1 do
5      for each vertex v ∈ V do
6          if C(v) has been updated in previous iteration then
7              for each edge(u, v) ∈ E do
8                  C(u) ← min{C(u), C(v) + w_(u,v)};
9  flag ← false;
10 for each vertex v ∈ V do
11     if C(v) has been updated in previous iteration then
12         for each edge(u, v) ∈ E do
13             if C(u) > C(v) + w_(u,v) then
14                 flag ← true;
15 return flag
```

Time Complexity: The total time complexity is the same as the first question, which is $O(|E| \cdot |V|)$. □

3. (Bonus)Suppose you are a staff of SJTU who is in charge of arranging lessons. Suppose you have n time slots, n lessons and n professors. Clearly, you should assign exactly one time slot and one lesson to every professor. A lesson or a time slot should be assigned to exactly one professor.

   For each professor, he will prefer some certain time slots among these n time slots, and prefer to taught some certain lessons among these n lessons.

   A professor will be satisfied iff you arrange him both his preferred lesson and preferred time slot. Your goal is to satisfy as many professors as you can. Design an algorithm to output how many professors can you satisfy at most.

   Notice that this problem is really hard, even draft idea is welcomed.

   (Hint: Treat time slots, lessons, professors as nodes. Construct edges according to professors' preference. Use network flow algorithm to solve it.)

   Solution. Let's denote the professors with the node p_i, $(i = 1, 2, 3, \cdots, n)$. Similarly, each time slot is mark as t_i while each lesson is referred to as l_i. Also, we have a starting node (source) s and an ending node (sink) e. In our directed graph, we will include the set of professors twice.

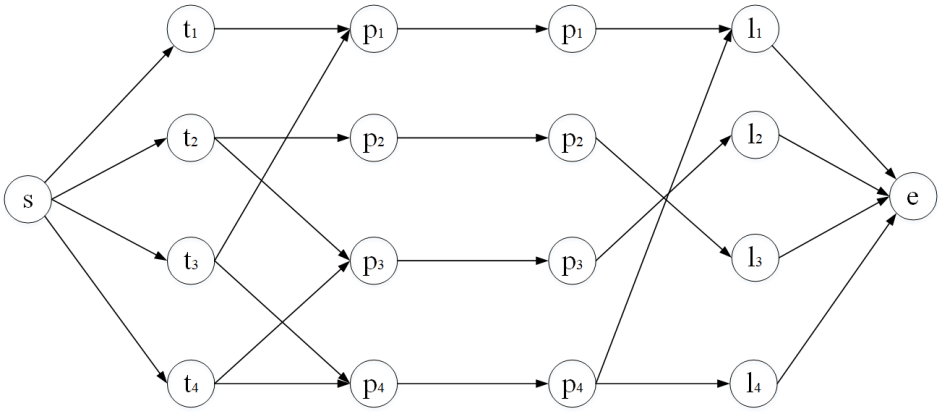
   For edges, we have:

   - There is an edge from s to t_i for $i = 1, 2, 3, \cdots, n$.
   - There is an edge from t_i to p_j iff. the j^{th} professor prefers the i^{th} time slot.

- There is an edge from p_i to the next layer of p_i for $i = 1, 2, 3, \cdots, n$.
- There is an edge from p_i to l_j iff. the i^{th} professor prefers the j^{th} lesson.
- There is an edge from l_i to e for $i = 1, 2, 3, \cdots, n$.
- There are no other edges except those stated above.

Suppose that each edge has a capacity of 1.

Here is an example graph illustrating the above rules.



The path from s to e will surely have four nodes in between, one standing for a time slot, two for the same professor and one for a lesson. By observation, we ought to find the value of the max flow from s to e with the Ford-Fulkerson algorithm, which is exactly the number of professors that we can satisfy at most.

Algorithm 4: AUGMENT(f, c, P)

```
δ ← bottleneck capacity of augmenting path P;
for each e ∈ P do
    if e ∈ E then
        f(e) ← f(e) + δ;
    else
        f(e^R) ← f(e^R) − δ;
return f
```

Algorithm 5: Ford-Fulkerson

Input: Directed graph $G = (V, E)$, c, s, e
Output: $v(f)$, the value of the max flow

```
for each e ∈ E do
    f(e) ← 0;
G_f ← residual graph;
while there exists augmenting path P do
    f ← AGUMENT(f, c, P);
    update G_f;
return v(f)
```

□

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