

Lab09-Network Flow

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2020.

* If there is any problem, please contact TA Shuodian Yu.

* Name: Haotian Xue Student ID: 518021910506 Email: xavihart@sjtu.edu.cn

1. Given a weighted directed graph $G(V, E)$ and its corresponding weight matrix $W = (w_{ij})_{n \times n}$ and shortest path matrix $D = (d_{ij})_{n \times n}$, where w_{ij} is the weight of edge (v_i, v_j) and d_{ij} is the weight of a shortest path from pairwise vertex v_i to v_j . Now, assume the weight of a particular edge (v_a, v_b) is decreased from w_{ab} to w'_{ab} . Design an algorithm to update matrix D with respect to this change, whose time complexity should be no larger than $O(n^2)$. Describe your design first and write down your algorithm in the form of pseudo-code.

Solution. Since the shortest path between any two vertices before the modification are given, then for any two vertices x and y , if the shortest path between this two vertices includes (v_a, v_b) , $d(x, y)$ should be modified. The above situation will happen if and only if $d(x, y) = d(x, v_a) + d(v_b, y)$ or $d(x, y) = d(x, v_b) + d(v_a, y)$. So we can use two loops to tranverse all the (x, y) pairs and try to renew the shortest path. Then time complexity is obvious no larger than $O(n^2)$. The pseudo code is as follows:

□

Input: $G(V, E)$, $D = (d_{ij})_{n \times n}$.

Output: updated D

```
1  $D' = D = (d'_{ij})_{n \times n}$ 
2 for  $x$  in  $V$  do
3   for  $y$  in  $V$  do
4     if  $d'_{xy} = d'_{xv_a} + w_{ab} + d'_{v_b y}$  then
5        $d_{xy} = d'_{xv_a} + w'_{ab} + d'_{v_b y}$ 
6     if  $d'_{xy} = d'_{xv_b} + w_{ab} + d'_{v_a y}$  then
7        $d_{xy} = d'_{xv_b} + w'_{ab} + d'_{v_a y}$ 
8 return  $D$ ;
```

2. Given a directed graph G , whose vertices and edges information are introduced in data file "SCC.in". Please find its number of Strongly Connected Components with respect to the following subquestions.
 - (a) Read the code and explanations of the provided C/C++ source code "SCC.cpp", and try to complete this implementation.
 - (b) Visualize the above selected Strongly Connected Components for this graph G . Use the *Gephi* or other software you preferred to draw the graph. (If you feel that the data provided in "SCC.in" is not beautiful, you can also generate your own data with more vertices and edges than G and draw an additional graph. Notice that results of your visualization will be taken into the consideration of Best Lab.)

Solution. (a) Tarjan algorithm is used to solve the SCC problem, and the code is in `./SCC.cpp` and the result for the test case id 666.

(b) To visualize the 100000 vertices, I do some modifications according to results in (a). To be specific, the Strongly Connected Components are treated as new vertices in the generated graph $G' = (V', E')$ and for any two SCC SCC_1 and SCC_2 , we have:

$$(SCC_1, SCC) \in E' \Leftrightarrow \exists v_1 \in SCC_1, v_2 \in SCC_2 \text{ st. } (v_1, v_2) \in E$$

according to the results in (a), we have $|V'| = 666$, the visualization is generated by Gephi with certain distribution and the color of nodes are determined by their ranks. The figures can be referred to in the last page of this lab report.

□

3. The **Minimum Cost Maximum Flow** problem (MCMF) is an optimization problem to find the cheapest possible way of sending the maximum amount of flow through a flow network. That is, in a flow network $G = (V, E)$ with a source $s \in V$ and a sink $t \in V$, where each edge $(u, v) \in E$ has a capacity $c(u, v) > 0$ and a cost $a(u, v) \geq 0$, find a maximum s - t flow f over all edges ($f(u, v) \geq 0$), such that the total cost of $\sum_{(u,v) \in E} a(u, v) \cdot f(u, v)$ is minimized.

A common greedy approach to solve the MCMF problem can be described as follows: We can modify Ford-Fulkerson algorithm, where each time we choose the least cost path from s to t . To do this correctly, when we add a back-edge to some edge e into the residual graph, we give it a cost of $-a(e)$, representing that we get our money back if we undo the flow on it.

Note that such procedure may create a residual graph with negative-weight edges, which is not suitable for Dijkstra's Algorithm. However, motivated by Johnson's Algorithm, we can reweight the edge cost with vertex labels and convert the weight non-negative again.

Please prove the correctness of such greedy approach and implement this algorithm in C/C++. The file *MCMF.in* is a test case, where the first line contains four graph parameters n, m, s, t , and the rest m lines exhibit the information of m edges. Each line contains four integers: u_i, v_i, c_i, a_i , denoting that there is an edge from u_i to v_i with capacity c_i and cost a_i . (Your source code should be named as *MCMF.cpp* and output the maximum flow and minimum cost of this test case.)

Sample Input:	Sample Output:
4 5 4 3	50 280
4 2 30 2	
4 3 20 3	
2 3 20 1	
2 1 30 9	
1 3 40 5	

Remark: The source code *SCC.cpp*, and the input data *SCC.in* and *MCMF.in* are attached on the course webpage. Please include your .pdf, .tex, .cpp files for uploading with standard file names.

Proof. The code is in *./MCMF.cpp* and with the input in *./MCMF.in* and output in *./MCMF.out*. We call ResGraph as short for the residual graph.

Brief description: In each iteration, we figure out the path with the least cost in the ResGraph and increase the flow in the path by the bottleneck flow. Then we keep doing this until there is no path between s and t in the ResGraph. Then let's proof the correctness of this algorithm for finding the MCMF.

Lemma 1. For flow f , f is the minimal cost flow \Leftrightarrow there exists no cycle with negative cost.

Proof. " \Rightarrow ": Suppose there is a negative-cost cycle in the ResGraph, it is $v_1 - > v_2 - > \dots - > v_k$, and the edges in the cycle is: $e_1 = (f_1, c_1), \dots, e_k = (f_k, c_k)$, where f_i and c_i are the flow and cost respectively. Then we augment the flow along the negative cycle, that will obviously not affect the property of flows. Since for every edge augmented, the in-flow changes with respect to the out-flows.

The most important thing we want to know is that the operation above will destroy the negative cycles in finite steps. Think about e_1, e_2, \dots, e_k , after each iteration of augmentation of cycle, the f_1, f_2, \dots, f_k is decreasing and it is sure that after finite steps, some of them will become zero, which make the negative cycle destroyed.

Also, we can proof the operation will decrease the cost of flow. The key point is: the cost of a flow edge (inversed edge) in a ResGraph is always negative. So for the negative cycle, we have the change of total cost:

$$\Delta \sum_e f(e)c(e) = \min(f_1, \dots, f_k) * \sum_{i=1}^k c_i$$

Since in the negative cycle, we have $\sum_{i=1}^k c_i < 0$, then we can see the total cost is decreasing. So we can proof the lemma by contradiction, there is no negative cycle in the ResGraph of MCMF actually.

" \Leftarrow ": If there exists one flow f' which has a cost fewer than f , we think about $f' - f$, which is made up of cycles and since f' has a fewer cost than f . We can see $f' - f$ must have negative cycles in it, which makes sense in the ResGraph of f as well since the

Lemma 2. *During the algorithm, no negative cycle is generated.*

Proof. If we generate one negative cycle C contains v_1, v_2, \dots, v_k . Since the edges on the new path P has a negative cost, and we have $c < 0$ is the sum of cost of $P \cap C$. so the sum of cost on rest edges must be smaller than $|q|$. So if we change P to $C - P$, the cost will be fewer, which contradicts with the shortest path. □

□

Considering the above two lemmas, we can get that the algorithm is correct. □

□

