Lab09-Network Flow

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

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

∗ Name:Hanzhang Yang   Student ID:518030910022   Email: linqinluli@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. For it has just changed one edge, the b_{th} row of matrix D can be updated by $w^{'}_{ab}$ and $D_{i,b}$. And for the rest element in D, it can be updated by the b_{th} row. Because for all path, it can visit b or not. The shortest path weight $d[i][j] = min\{d[i][b] + d[b][j], d[i][j]\}$

Here is the pseudo-code.

```
Algorithm 1: updated shortest path matrix
Input: G(V,E), W = (w_ij)_{n×n}, D = (d_ij)_{n×n}, w'_ab
Output: updated D = (d_ij)_{n×n}

for j = 0 to n do
    d[j][b] = min{w'[a][b] + d[j][a], d[j][b]};
for i = 0 to n do
    for j = 0 to n do
        d[i][j] = min{d[i][b] + d[b][j], d[i][j]};
```

And it's clear that the time complexity is $O(n^2)$. □

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.)

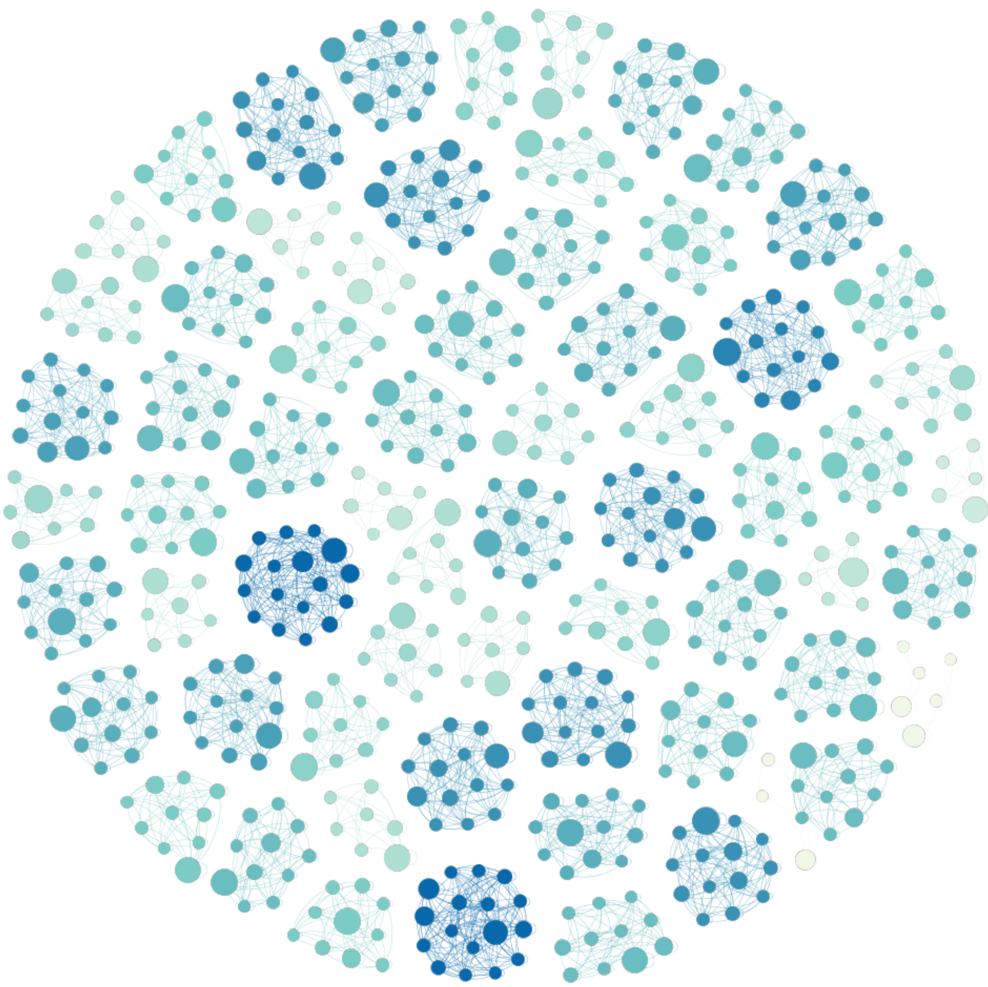

图 1: SCC(node size represents pagerank; color represents degrees)

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. Now we prove it by induction. Assume $f(x)$ is the minimized cost flow when flow is equal to x.

And f is a minimized cost flow which is equal to that the residual graph has no negative circle. Or the cost can be smaller by visiting these negative circles.

Basics.

It's clear that $f(0) = 0$.

Induction proof.

Assume $f(k)$ is the minimized cost flow. And $f(k+1)$ is the flow induced by our algorithm. Thus $f(k+1) - f(k)$ is a shortest path from s to t.

Now we assume there is another flow $f'(k+1)$ with lower cost. There must exist negative circles in $f'(i+1) - f(i)$, which causes a contradiction.

Therefore $f'(k+1)$ is the minimized cost flow. And the algorithm is correct.

$\square$