Lab09-Network Flow

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

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

∗ Name:Zehao Wang Student ID:518021910976 Email: davidwang200099@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. When w_{ab} changes, for any different vertexes u and v, the weight of path $u \to a \to b \to v$ changes. If this path has smaller weight than the original shortest path from u to v, the smallest weight should be updated. Otherwise it should not.

   Both the outer loop and inner loop iterate n times, and the update has $O(1)$ complexity. Therefore this algorithm has a complexity of $O(n^2)$.

   □

Algorithm 1: ShortestPaths(G,W,D,a,b,w)

Input: Graph $G = (V, E)$, weight matrix W, shortest path matrix D, a, b, new weight of (a, b): w.

Output: Updated shortest path matrix D

```
1 D[a][b] ← w;
2 for i = 1 to |V| do
3     for j = 1 to |V| do
4         if i ≠ j then
5             D[i][j] = min{D[i][j], D[i][a] + w + D[b][j]};
6 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) Code has been put into the .zip file as SCC.cpp.

   (b) There are 666 SCCs in total.

   □

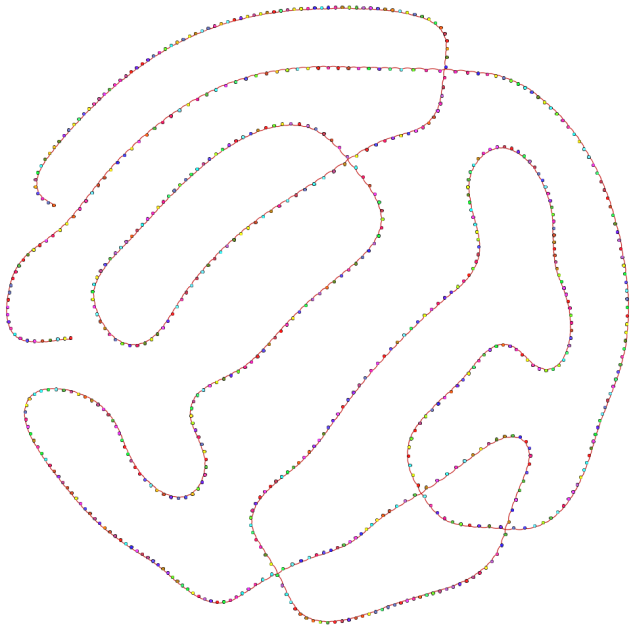

Figure 1: Visualization of SCC. Each vertex stands for one strongly-connected component.

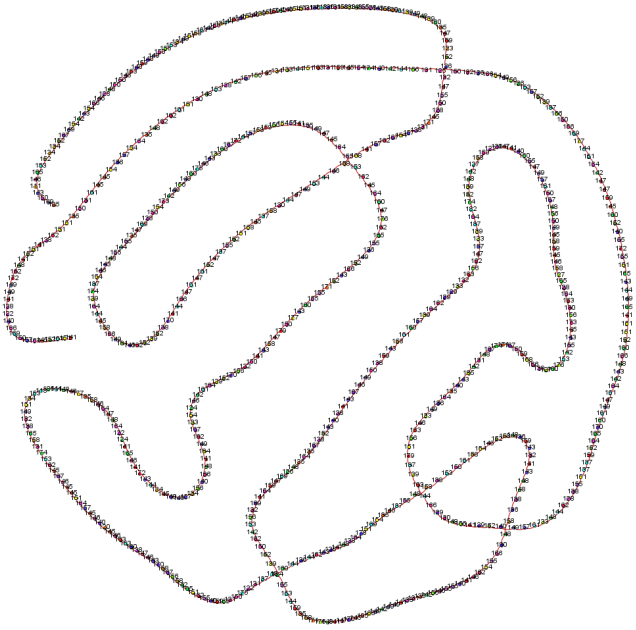

Figure 2: Visualization of SCC. Numbers of vertexes in each SCC is marked as labels.

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 4 2 30 2 4 3 20 3 2 3 20 1 2 1 30 9 1 3 40 5	50 280

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.

Solution. (a) Assume that two flows f and f' are equal, but f' costs less than f.

Then consider the flow $f' - f$.

The flows into all vertexes except s and t equal the flows out of them. Therefore there is a cycle whose total cost is negative. Some flow can go around this cycle and pay less to go to the target.

Therefore, as long as there is a negative cycle in a flow f, f must not be the minimal cost flow.

Assume that $f(i)$ denotes the cost found by this greedy approach when total flow from s is i and $f(i)$ is indeed the minimal cost.

$f(0)$ is just the original graph. Obviously for a graph without negative cycle, $f(0) = 0$.

When the total flow is $i + 1$, by the same greedy approach, we can also find $f(i + 1)$.

Assume that $f(i + 1)$ is not the minimal cost and $f'(i + 1)$ is the minimal cost.

Then consider the flow $f'(i + 1) - f(i)$.

By definition, $f(i+1)-f(i)$ is a flow through the least cost path. However, $f'(i+1)-f(i)$ cost even less than $f(i + 1) - f(i)$. That must be because there is a cycle with negative cycle in the graph, which means that $f(i)$ is not the minimal cost, which is contradictory to the former assumption. Therefore $f(i + 1)$ must be the minimal cost.

Therefore this greedy approach must be correct.

(b) For the test case, the maximal flow is 14098 and the minimal cost is 5290116.

□