Homework 2 – Algorithm

Part 1 Directed Graph

Introduction

Description

1. Requirement

2. Input / Output
   - **Input file format**

     input_example_1:

     ```
     4 5
     0 1 1
     0 2 1
     1 2 1
     3 0 1
     3 1 1
     ```

     input_example_2:

     ```
     4 8
     0 1 1
     0 2 1
     0 3 1
     1 0 1
     1 2 1
     1 3 1
     2 0 1
     2 1 1
     ```

     The first line of input files contains two integers representing the number of vertices(n) and edges (m) in the graph.

     The following m lines contain three integers representing the start vertex(s), the target vertex(t) and weight (w). The weight of edges in Part 1 input files are guaranteed to be one.

     In each line, all the integers will be separated by space.

- **Output**
  If the graph is acyclic, output files should contain exactly one line that represents the topological order of the graph.
  You only need to print the first topological order obtained by DFS.

  output_example_1:
  ```
  3 0 1 2
  ```

  If there are cycles in the graph, you need to print CG computed from G. The format is the same as the input files.
  We define the value of SCC as the smallest vertex in SCC. The index of nodes in CG is determined by the sorted value of the SCCs list and the weight of edges in the CG is determined by the sum of edge weights in G from one node to the other. See the example below for details.

  In the input_example_2, there have 2 SCC in the graph:

  ```
  {0,1,2} // C1
  {3}     // C2
  ```

  We can get the sorted list of the value of SCCs:

  ```
  0, 3
  ```

  Use its index as the node in CG:

  ```
  C1: 0
  C2: 1
  ```

  Calculate the weight of edges in the CG:

  ```
  for v1 in CG(0):
        for v2 in CG(1):
              w_CG(0, 1) += w_G(v1, v2)
  ```

  output_example_2:
  ```
  2 1
  0 1 2
  ```

Constraint

- You are required to implement it in C++.
- 0 < n <= 1000
- 0 < m <= 3000
- 0 <= s, t < n
- w == 1.

Part 2 Shortest Path

Introduction

In this part you need to implement 2 different single source shortest path algorithms and try to do comparison between them.

Description

1. You need to implement 2 different shortest path algorithms.
    - Dijkstra's Algorithm
    - Bellman-Ford Algorithm
        - You have to detect if there is any negative loop in the graph when implementing this algorithm.

2. Requirement

    You need to find the shortest path from the first node to the last. For example, if there is a graph containing 9 nodes, you need to find the shortest path from node 0 to node 8.

3. Input / Output
    - **Input**

      The first line of input files contains two integers representing the number of nodes(n) and edges(m) in the graph.

      The following m lines represent the edge information. Each line contains three integers: start node(a), end node(b), and the cost(c).

      *For Dijkstra's Algorithm, we define the cost of edges as abs(c).
      *The path from the start node to the end node is guaranteed to exist in every input graph.

      In each line, all the integers will be separated by space.

    - **Input file format**

      input_example_3:

      ```
      5 5
      0 1 1
      1 2 2
      2 3 -3
      3 1 -5
      3 4 4
      ```

    - **Output**

The output should contain two lines.
First line contains the cost calculated by Dijkstra's Algorithm.
Second line contains the result of Bellman-Ford Algorithm:
If there is no negative loop in the graph:
The output should contain one integer that represents the cost of the shortest path.
If there is a negative loop in the graph, please output the string "Negative loop detected!"

- **Output file format**
  output_example_3:

```
10
Negative loop detected!
```

4. Discussion
   - Describe how you detect the negative loop in the BellmanFord Algorithm.
   - If you need to print out the path of the shortest path, describe how it can be done?
   - Compare the time complexity of the two algorithms.

Constraint

- You are required to implement it in C++.
- You are free to use the C++ standard library.
- 0 < m <= 1000
- 0 < n <= 3000
- 0 <= a, b < m
- -10000 <= c <= 10000.

Grading

Correctness (80%)

- strongly connected component (20%)
- Topological Sort (20%)
- Dijkstra's Algorithm (20%)
- Bellman-Ford Algorithm (20%)

Paper Report (20%)

Paper Report Guideline

The paper report should be detailed, clear, and well-organized. You have to describe how you implemented the algorithm and show the result. If you encountered any challenges during the implementation, it is encouraged to describe it in the Discussion as well.

Please write the paper report with the following bullets. You must write down two parts of it. Each part must have implementation details and discussion.

- implementation Details
  - Steps
  - Etc.

- Discussion
  - Time complexity
  - Your discover
  - Which is better algorithm in which condition
  - Challenges you encountered
  - Etc.

Submission

1. A zip file named <HW2_studentID.zip> includes the following :
    - A folder named <HW2_studentID> includes the following :
        - Part1.cpp
        - Part2.cpp
        - Part1.h
        - Part2.h

        *This means that you can't modify the contents of "main.cpp", "makefile" and "SolverBase.h".
    - A paper report named <report_studentID.pdf>

2. Uploaded onto the new E3 platform before 06 / 06 (Mon) 11:55 pm.
3. The late penalty will be 10% per day. The last submission is allowed before 06 / 10 (Fri) 11:55 pm.
4. If there is any wrong format. You will get a 5% penalty each.
5. Plagiarism is forbidden. You will get 0 points if we find it.

Reference

http://alrightchiu.github.io/SecondRound/mu-lu-yan-suan-fa-yu-zi-liao-jie-gou.html
https://web.ntnu.edu.tw/~algo/DirectedAcyclicGraph.html
https://www.geeksforgeeks.org/topological-sorting/
https://web.ntnu.edu.tw/~algo/Path.html