Algorithms PA3 report

B07502028 吳宗翰

1. Data Structures Used in PA3.

(1) `edge.h`: a class representing an edge
`class Edge`: Attributes `i, j, w` represents an edge (i, j) with weight w.

(2) `djset.h`: a disjoint set (with path compression, union by rank) for maxSTByKruskal
`vector<int> p`: Specify parents of all vertices
`vector<int> r`: Specify ranks of all vertices

(3) `graph.h`: the graph data structure
`vector<Edge> edgesList`: store all the (sorted) edges
`vector<Edge> edgesNotInMaxST`: store all the (sorted) edges that are not in the MaxST generated by maxSTByKruskal
`vector<vector<int>> adjListMaxST`: adjacency list representation of the MaxST generated by maxSTByKruskal and addMoreEdges procedure

(4) `main.cpp`:
`vector<Edge> edgesList`: store all the edges read from the input file

2. Findings.

(1) Initially, I used countingSort to sort all the edges, but later on I found out that the built-in sort produces better cost. The reason why different sorters produce different results is that different sorters may sort the edges of the same weight in different orders. However, inserting edges of the same weight in the addMoreEdges procedure in `graph.h` in different orders may poduces different results (since inserting one edge may prevent another edge from being inserted).

(2) Initially, I used the Dfs procedure implemented in the textbook to check existence of any cycle on a directed graph. However, when an edge (i, j) is inserted in the addMoreEdges procedure in `graph.h`, the cycle must include (i, j) if any cycle exists. Therefore, I modify the Dfs procedure so that it starts from vertex j and return true (meaning a cycle exists) if vertex i is visited.