CSC3100 A4 REPORT

XUE, Zhongkai 122090636

Problem 1

This problem mainly challenges us to build a DFS basically from scratch. Hence I have implemented a DFS class to satisfy the needs to traverse with DFS algorithm.

The graph is utilized with adjacent list in the size (v + 1). A boolean array is applied to indicate the status of visited or not; an array is used to store the output DFS order; v is used for the number of vertices.

To run the dfs with respect to a single vertice, first indicate the visited status, and record it in the dfs order. After that, alll neighbours of this vertice, if not previously visted, will be executed recursively.

Given this sub-module to run dfs, we could construct the adjacent list with the given parent list. After dfs is executed based on this adjacent list, we could finally compare with the given order for yes or no.

The time complexity of this approach is O(N) where N is the number of nodes. This efficiency is achieved as each node is visited exactly once during the DFS traversal.

Problem 2

This problem mainly challenges us to build a MST and apply it on an application problem. To implement this Prim's Algorithm, basically I import the heapq from the Python standard libra and modify it for my use.

The algorithm is devided into preparation, randomly select a vertice to begin and add edges with minimal costs. In the randomAdd, I add vertices connected with this random one at a non-zero cost, and push it into the heap. Then, I ask for the vertice with minimal cost from the heap, add it by the cost and continue with new exploration.

This problem could also be solved with Kruskal's algorithm, both at the cost of O(NlogN), though the heap operation may add a bit complexity to that.