Lawrance
B09901116
EE3

PA2 Report

All were written by myself without copying, but there are references used(mostly for part 3 directed).

Overall

First I thought to use Maximum Spanning Tree and deleted the remaining edges that were not used. For this, between Kruskal and Prim, I decided to use Kruskal because it seems a lot easier to implement. After looking through some videos of MST, I started to code.

Data Structure Analysis:

cycleBreaking.cpp and cycleBreaking.h

First, I created four classes, which are Vertex, Edge, Graph, and Disjoint Set. The class Graph contains functions that take in vertex and edge from the other two classes as object-oriented programming. Class Edge initializes to establish pointers toward things that belong to each object declared by the class, which is starting point u, target point v, and weight between u and v, w. Vertex is initialized by setting its parent, rank, and color to 0, used for future DFS and MST. Graph initialized to get input of the total number of edges, vertices, and the type (directed or undirected).

When sending in a list of vertices connected to form edges with weight, we create a function in class Graph named “addEdge”, which creates an Edge object carrying u, v, w, and it is pushed into a vector and the object, corresponding to the directed and undirected graph. Since we would be considering a directed graph, we must use the function of erasing edges from a graph. The function “delEdge” is easily achieved by giving it the edge, then using erase function to erase the vector and also minus 1 in edges. For DFS/DFS visit function, I use pseudocode from the textbook as a sample, which returns true or false depending on if a node traverses around back to itself and we check all nodes.

Next, for Kruskal, we need to sort weight, find set, and union/link. For sorting, for undirected graphs, we can just use merge sort function from PA1(so I just copied and pasted), but change it such that it is in descending order instead of ascending order. It is also used in the reference I referenced from, and I believe the reason for using merge sort is because its time complexity is $\Theta(n\log(n))$ in the worst, best, and average cases. Next, for disjoint set operations(find, union, link), just reference pseudocode from the textbook. For the Kruskal operation, the same pseudocode from the textbook, but this time we add a vector that stores the edge that is removed in MaxSpanningTree since those are the ones we want to remove and output.

main.cpp

Lawrance
B09901116
EE3

- read input and send the number of vertices, edges, and type to initialize Graph class object. Next, add edge to initialize Edge. Lastly, we call sort and Kruskal to get the result of “undirected”. For directed, inspired by my reference, we removed all edges from the removed edge that we got from Kruskal Algorithm. Sort those edges and if the removed edge weight is greater than zero, add them into the graph G. Push back each edge with a weight greater than 0 into temporarily removed vertex. From that temporarily removed vertex, we are allowed to erase the ith(amount of weight that is not negative) amount of edge from the removed edge and still maintain an MST. This is due to the fact we have negative weighted nodes, such we are allowed to do so.
- Finally, just print the answer.

- Merge Sort runs O(nlgn)
- Kruskal runs O(ElgE)

Total time complexity is related to Kruskal algorithm’s running time since it has a higher time complexity(dominates).

From this problem, I’ve learned a lot of conditions that must be met in the difference between directed and undirected. At the start, I would think that it makes no difference between directed and undirected, which proved that I was wrong from the start. Another thing that I have learned is to combine things we’ve learned from the past such as sorting, DFS, MST and combine it to solve a problem.

References:

- Psuedocodes from the textbook
- Previous PA1(merge-sort part used in Kruskal)
- github/GeeksforGeeks