陳維翰（B08901129）

Algorithms

Professor **Chang**

28 December 2022

Report for Programming Assignment 3

First of all, let’s just say I have a bit more time to research due to most class already ended. I have always used fin and fout my entire life, never really familiarize myself with fscanf and fprintf. But it was to my surprise that fin and fout are significant slower than fscanf and fprintf. Next, let’s really go into this assignment. So mainly there are three main cases for this assignment. The first case being that the entire graph is unweighted and undirected with all edges equaling to one. The second case being that it’s an undirected weighted graph that could include zero/negative edge weights. These are a bit harder, but still manageable. Last but not least, the directed and weighted graphs are the hardest of them all. So, for these more difficult optimization problems, I’m thinking I should either develop efficient heuristics or use greedy/local search method to solve them.

Let’s kick everything off by starting with the data structure for storing the edges. At first, I thought of story them in a 2D vector, but it’s impossible to compress the entire runtime down to the requirement of 1 minute. Thus, it was after a while I came across this article saying that using self-defined structure by using “struct” will greatly reduce the runtime by a ton. Now let’s move on to undirected weighted/unweighted graphs. To minimize remove cost, we used the Kruskal algorithm by letting it choose the largest weight edge and connect it to the graph. This is due to the fact that we discover that undirected graph can contain at most v minus one edges without cycle. I did implement the disjoint set method that we talked about in class. This is achieved by having two arrays recording its rank and each other’s parent. In addition, there are functions like find, union, createSet, etc.

Now, moving to the extremely hard directed graphs problems. I try to follow the hint given by TA that save as many positive weights and try to delete as many negative weights as possible. Thus, we should sort the edges from biggest weights to smallest weights. We add the biggest one to the adjacency list while using depth first search to check whether it creates a cycle or not. Now moving on to the case when we have edges with negative weights. In these cases, we need to use Union function to solve it. Due to the fact that the depth first search is only use for checking cycles, only the recoding color part of algorithm is needed. Lastly, it was given that these cases have smaller data size so we can run this thing a couple times to see which shuffled version creates the best output.