**ITCS 6114: Algorithms and Data Structures**

**Shortest Paths in a Network**

**Program Environment:**

The program has been developed in C++. Netbeans IDE 8.2 was used as the environment to build the program. The GCC version used for development was GCC 5.4.0

**Program Design:**

The program has been implemented majorly using STL containers such as maps, vectors and pairs. Separate classes have been with well-defined functions and data members

**Tasks Performed:**

1. Building initial graph
2. Updating graph to reflect changes
3. Finding shortest path between any two vertices
4. Printing the graph
5. Finding reachable sets of vertices

I will briefly explain each task performed.

**Building the graph:**

A graph was formulating using an input file provided by the user (Ex: network.txt) in the command line. This file typically has a pair of strings and a floating-point value denoting edge weights. I designed vertex, edge and graph classes and placed its relevant members within. The graph was structured using an **addedge** function which gets the source and destination vertices and constructs its adjacency list.

**Updating graph to reflect changes:**

To perform this operation, six primary functions were implemented. **addedge**, **deleteedge**, **vertexUp**, **vertexDown**, **edgeUp** and **edgeDown.** These functions perform their respective operations and would not be considered while obtaining the shortest path or reachable vertices if their status is **DOWN**

**Finding shortest path between any two vertices**

Djikstra’s algorithm is implemented as a priority queue using a binary min-heap. A Min-heap class was created and operations such as Build-heap, min-heapify and decrease key operations were handled. Initially all the vertex distances were set to “INFINITY” and their parent vertices were set to NULL. A map was used to store vertices and their information. Once the vertices were initialized, they were added to the priority queue and built using a build-heap operation. For the core implementation of Djikstra’s algorithm, I Iterated through all the vertices and their adjacency lists and simultaneously updated the priority queue. Every time a value popped out of the priority queue, the decrease key operation was performed. The statements in the inner loop are executed O(V+E) times. The decrease key operation takes O(logV) time. The overall complexity is O(E+V)\*O(LogV) 🡪 **O(ElogV).** The paths are then printed by recursively calling the destination vertex’s parent vertex taking O(n) at most.

**Printing the graph:**

The vertices and their information are stored in a map of string and a pointer to a vertex. The information for edges are also stored in a map containing string and a pointer to an edge. Since maps are organized as red-black trees, each vertex insertion takes O(logV) time and each edge insertion takes O(logE) time. So, the insertion and sorting time together for V vertices would be O(VlogV), iterating through V vertices and sorting time together for E edges would be O(V\*ElogE). Printing all vertices and their edges after sorting would take O(VlogV + V\*ElogE)

**Reachable set of vertices:**

The algorithm used to print reachable set of vertices takes O(V\*(V+E)). For each vertex, a depth-first search algorithm is employed. This would take O(V+E) for each vertex. Since reachable vertices should be printed out for every vertex, every vertex must be iterated once. Sorting using a map for each set of vertices would take O(VlogV) time, hence, sorting through all set of vertices would take O(V2logV). Hence the total running time of the algorithm to determine reachable vertices would be **O((V\*(V+E))+ V2logV)**

**Command Used:**

Command Prompt: 1) g++ shortestpath.cpp

2) a.exe Input.txt < queries.txt >Output.txt