**Group Problem Approach**

Dijkstra’s Algorithm being a greedy algorithm, need to fetch element with minimum distance from source; from a set of elements that have not been processed already. Once a vertex is fetched from this set, it would update other vertices connected to it. While implementing, we need to fetch vertex with minimum distance with maximum efficiency from a set of unprocessed elements, thus we decided to use min heap. But since, a heap runs percolateDown operation as soon as an element is deleted, we modified this heap to run percolateDown when an element inside a heap is updated. This heap stores DijkstraHeapNode as its element. We created a separate class DijkstraHeapNode, which stores previous vertex and distance information from source.

percolateDown operation takes in an index as a input, so we need to store this index into DijkstraHeapNode. Every DijkstraHeapNode is associated with a vertex and also points to the vertex it is associated with. Previous Vertex in this node points to the vertex through which it is at a minimum distance from source. Every time this distance is updated, DijkstraHeapNode would notify any listener who registers itself to distance updated operation, which in this case is heap.

We also created three input files that simulate different situations in a graph.

* The first file simulates finding the shortest distance in a graph using dijkstra’s algorithm.
* The second file simulates the situation in which a given city is approachable from another city via multiple parallel routes but with different edge weights (this can be considered as simulating a traffic situation). The algorithm evaluates both routes and picks the one with lower edge weight (or lower traffic and hence faster).
* The third input file applies dijkstra’s algorithm to a graph that consist of cycles.

**Individual work**

Following were the tasks performed:

* Implementation of the DijkstraHeapNode class which consists of distance, heap index and previous node.
* Implementation of compareTo function in DijkstraHeapNode.
* Modification of the heap code to insert DijkstraHeapNode object, and update the heap index inside DijkstraHeapNode every time its location inside heap array is modified.
* Mechanism to notify heap when a distance from source is updated inside DijkstraHeapNode.
* Implementation of actual Dijkstra’s algorithm once above mentioned tasks were achieved and verification of result.

**Difficulties encountered**

* It was difficult to modify the heap code to suit the implementation of Dijkstra’s algorithm.
* It was difficult to notify the heap such that when distances are updated, it should make a call to percolate up.

**Learning outcome**

This exercise helped in learning the following:

* Understood Dijkstra algorithm and its implementation.
* Understood complicated data structures (heap) and how to modify them as needed.
* How to coordinate and work in a group.
* Better task management and time division between tasks.
* How to use subversion system (SVN).
* Gained familiarity with Java UI components.