Ethan Nguyen
ethantn04@csu.fullerton.edu
CPSC 335 Algorithm Engineering
Project 3 - Algorithm 2

Project Report - Algorithm 2

Pseudocode

Function findCheapestRoute(int n, vector routes, int src, int dst, int k)

- a graph needs to be built as an adjacency list
- the graph needs to have n empty lists

```
function findCheapestRoute(int n, vector routes, int src, int dst, int k):
        for each route in routes:
                from route[0], to route[1], price route[2]
                add to and price to from
```

A min-heap should be used to track the cheapest route.

```
pq.push (0, src, 0)
```

An array should be made to track the minimum cost to each of the nodes. Once these elements have been made, the priority queue should be processed.

```
while pq not empty:
        (cost, node, stop) = pq.top

        (if node == dst) return cost //destination with the least total cost

        (if stop > k) continue // if exceeded the max number of stops

        for each (neighbor, price) in graph: //checking neighbors of current node
                new = cost + price
                 if new < visited[neighbor] or stop < k:
                        visit[neighbor] = new
                        pq.push(new , neighbor, stop + 1)

return -1
```

Sample

```
vector<vector<int>> routes1 =
   {
      {0, 1, 100},
      {1, 2, 100},
```

```
        {0, 2, 500}

    };
```

Results :
Test Case 1 Output: 200
Test Case 2 Output: 500
Test Case 3 Output: -1

Mathematical Analysis & Efficiency Class

Let n be the num of cities which can count for the number of nodes.
Let e be the num of routes which can count for the number of edges.
Let k be the number of maximum stops that are allowed.
A graph must be created and for each edge, there is a directed edge added to the adjacency list which is of O(e) time.

When a visited array is initialized, it will track the least cost so far per node which takes O(n) time.
For the min-heap, when each of the node's neighbors are pushed into the queue, a node could be pushed into the queue k + 1 times. Each heap operation should take O(log n) time.

Considering the total cost of these operations, the time complexity should be:
O((k + 1) x E x log n) or O(kE log(n))