TAHSIN SHAMSUL
Dijkstra Writeup

1. <u>Explanation of Data Structures</u>

   – To solve this problem, I used several data structures. First, I initialized an adjacency list of each vertex to hold each of its neighbor and its cost to that neighbor from this specific vertex. I used a dictionary to store the adjacency list because I can look up vertices in O(1) complexity and their neighbors in O(N) depending on the numbers of their neighbors. The adjacency list comes in handy for my Graph to represent the [visited, cost, parent] value of each vertex. I also stored this graph in a dictionary with one list having the values mentioned to perform Dijkstra's algortithm and updating whether it was visited, each vertex's optimal cost from the starting node, and the previous node that determines its optimal path. The obvious data structure used here was Dijkstra's algorithm to find and update each sub-optimal path from the starting vertex on its way to the destination point. Finally, for Dijkstra's algorithm, once I updated a vertex's neighbors and finalized the vertex, I needed to determine the next non-finalized vertex with the currently smallest cost to be the next current vertex to analyze its neighbors and become finalized. Well, the non-finalized vertices were easy to put together, but it was the idea of determining how to pick the smallest cost vertex to be the next current. To do this I used a priority queue to add on each non finalized vertex into a list, so the list would be sorted from lowest cost to highest cost, so I can easily choose the first element in the list to be the next current vertex because it is the one with the lowest cost. These are the data structures I used to implement my solution.

2. <u>Implementation of Rome data</u>

```
Enter starting point: 10
Enter destination point: 3139
Minimum distance between 10 and 3139 is:  35355
Shortest Path:  ['10', '9', '11', '172', '173', '174', '175', '188', '191', '336', '338', '339', '354', '356',
'359', '363', '364', '366', '2990', '2991', '3033', '3059', '3066', '3107', '3135', '3136', '3139']
```

3. <u>Test Files</u> **TEST GRAPHS ARE IN LOWERCASE LETTERS**

   - graph1

     - In this graph, All the weights are equal which means the shortest number of vertices visited from start to destination is the most optimal. This is going to search through and update each vertex on its ways to its destination. Therefore, there can be multiple optimal paths since each edge carries the same weight. For example, in this graph, starting from 'a' and going to 'c' has two solutions, either [a,b,c] or [a,f,c]. However, Dijkstra's algorithm is not the most optimal solution with unweighted graphs because we are can just use BFS to find when the destination vertex is hit and then complete the algorithm rather than updating each node.

- graph2
  - In this graph, it contains only one solid path. The vertices are connected like the following
  a – b – c – d – e – f
  Basically it shows that the graph can only move in two directions. So if I were to start at 'a' and end at 'f' the expected output should be [a,b,c,d,e,f]. The algorithm should touch and update all vertices because there is only one solid path between these two vertices.

- graph3
  - In the graph, it tests one starting vertex that maps to the endpoint. The start point would be 'a' and the endpoint would be 'b'. However, even it maps directly to the endpoint, the weight for this path will greater than visiting through various other nodes. Im basically testing to see if the algorithm I implemented chooses its path based on the cost rather than the number of nodes visited, which shouldnt matter since it should be less than mapping directly from 'a' to 'b' which is 42 which is greater than visiting multiple vertices and having a weight of 39.