Assignment 3

Name: Minh Duy Le
Id: 7438321

1. A high-level description (in pseudo-code) of the overall solution strategy.

```
class A3:
//declare the variables
private int verticesNum;
   private int edgeNum;
   //Storing graph
   private Vertex[] vertices;
   private Node[][] adjacencyList;
   //Start and goal
   private int startVertex;
   private int goalVertex;
   //
   private double[] dist;
   private int[] prev;
   private double maxDist = 0;
   private int[] longestPath;
   private int[] tempPath;
   private int pathIndex;

   function dijkstra:
      Initialize dist[] = INFINITY
      Initialize prev[] = NULL
      Distance[src] = 0
      priorityQueue.add(src)

      while priorityQueue is not empty:
         u = node from priorityQueue with smallest distance
         if u is settled:
            continue
          Saved U on Queue

         for each neighbor v of u:
            if dist through u to v is smaller than known dist[v]:
               update dist[v]
```

```
            set prev[v] = u
            add or update v in priorityQueue with new priority

function findLongestPath(src, dest, currentDist):
   if src is destination:
      if currentDist > known longest distance:
         update longestPath
   else:
      for each neighbor of src:
         if neighbor is not visited:
            recursively call findLongestPath with increased distance

function readFile(filename: String):
   // Read the file
   Parse the number of vertices and edges
   Initialize adjacency list and vertices array
   Read the vertices
   Read the edges
   Read the start and destincation of vertices

function print():
   Print basic graph info
   Print Euclidean distance between start and end vertices
   Call dijkstra function
   Calculate and print the shortest path
   Call findLongestPath
   Print the longest path

main:
   Input filename
   Initialize classes and values
       Call the Graph Object
                  Read file
       Print the results
```

2. A complexity analysis of your solution with big-O notation and sufficient justification.

Best Case:
Array: O(1)
Priority Queue: O(logn)
Dijkstra's Algorithm (dijkstra function): O(nlogn)
Depth-First Search: O(n)

Average Case:
Array: O(1)
Priority Queue: O(logn)
Dijkstra's Algorithm (dijkstra function): O(nlogn)
Depth-First Search: O(n)

Worst Case:
Array: O(1)
Priority Queue: O(logn)
Dijkstra's Algorithm (dijkstra function): O(n^2logn)
Depth-First Search (used in findLongestPath function): O(n)

Total Complexity:
Best Case: O(nlogn)
Average Case: O(nlogn)
Worst Case: O(n^2logn)

3. A list of all of the data structures used, and the reasons for using them.

Data Structures	
Array	Data structures with constant-time access to individual elements are known as arrays. Several times throughout this programme, arrays have been used, including dist to record shortest distances, prior to store predecessor nodes, and vertices to store vertex information. The main benefits of using arrays are their effectiveness in offering quick access to data and their ability to save space while requiring little overhead. They help to ensure that data can be retrieved or updated quickly in situations when the amount of data is known and largely unchanging.
2D Array	The 2D array provides a substantial improvement in the program's representation of graphs, especially when used as an adjacency list. Individual components in each row of this structure point to adjacent vertices, representing each row as a representation of a vertex. The selection of this format is crucial for enabling quick access to and alteration of graph edges. It is frequently necessary for graph algorithms to do rapid lookups of nearby nodes, and doing so is most effectively accomplished by employing a 2D array to describe adjacency connections
Priority Queue	A sophisticated data structure called a priority queue always displays the element with the highest (or lowest, depending on the comparison function) priority for extraction. The priority queue is essential to the program's inbuilt Dijkstra's algorithm. It prevents the requirement to traverse over all vertices in search of the following node to process by ensuring that the vertex with the smallest known distance is always treated first. This not only streamlines the process but also optimises it such that

	insertion and removal operations run in logarithmic time.
Node – Vertex (Graph Components)	The Node and Vertex classes are both unique data structures created to encompass key elements in the field of graph theory. In order to abstract the idea of a network node, the Node class concentrates on details like node ID and related cost. The Vertex class, on the other hand, has a method to calculate the Euclidean distance between vertices in addition to encapsulating vertex characteristics like its ID and spatial coordinates. The programme simplifies processes involving these core things by developing these specific classes, assuring uniformity, clarity, and reusability. In addition to encouraging a more organised approach, classifying related features and functionalities makes future adjustments and scalability issues easier to manage.
Algorithms	
Dijkstra	A key component of shortest-path issues in graph theory is Dijkstra's method. This approach, developed by Edsger W. Dijkstra, effectively determines the shortest route from a source node to every other node in a weighted network. The basic idea behind the method is to keep track of the visited nodes and repeatedly update the shortest distances using a "greedy" strategy, making sure that at each step, the node with the smallest known distance gets processed next. The effectiveness of the method depends on a priority queue, which ensures that nodes are handled in the order of their most recent shortest distances. The algorithm's output will show the shortest route from the source to any given node
Depth-First Search	A fundamental graph traversal technique is depth-first search (DFS). It explores a network in great detail, stopping at nodes as far along each branch as feasible before

	turning around. The longest path between two nodes is determined using DFS in the context of this programme. Although determining the ultimate longest path through a broad graph is an NP-hard issue, we may effectively explore and assess alternative pathways by utilising DFS. Using this recursive approach, the programme may determine the longest path by investigating all options between the source and destination
Euclidean Distance	A measurement of the "straight-line" separation between two places in Euclidean space is provided by the Euclidean distance. This computation is used by the programme to calculate the separations between vertices depending on their spatial coordinates. This procedure, which is based on Pythagoras' theorem, determines the square root of the sum of squared differences between the two points' (or vertices') respective coordinates. This measure is used in a variety of graph-related situations as a heuristic or true distance metre.

4. A snapshot of the compilation and the execution of your program on the provided “a3-sample.txt” file.

```
OUTPUT   PROBLEMS 1   JUPYTER   SQL CONSOLE   TERMINAL

TERMINAL                                                              Run: A3
macbookmyduy@Elvans1 203_A3 %  /usr/bin/env /Users/macbookmyduy/Library/Java/JavaVirtualMachines/openjdk-20.0.1/Contents/Home/bin/java -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/macbookmyduy/Library/Application\ Support/Code/User/workspaceStorage/691bab4a6c836b33e8b6388c655c53e0/redhat.java/jdt_ws/203_A3_d4c5a0a4/bin A3
Enter the file name: a3-sample.txt
======================================================================================
The number of vertexes in the graph: 20
The number of edges in the graph: 100
The start vertexes: 2
The end vertexes: 13
======================================================================================
The Euclidean distance between the start and the goal vertices: 77.87811
Shortest path: 2 -> 13
The length of the shortest path: 85
Longest path: 2 -> 17 -> 9 -> 16 -> 4 -> 18 -> 14 -> 8 -> 6 -> 19 -> 3 -> 12 -> 5 -> 20
The length of the longest path: 1595.0
======================================================================================
macbookmyduy@Elvans1 203_A3 %
```

5. The outputs (the shortest and longest paths) are produced by your program on the provided "a3-sample.txt" file.

```
macbookmyduy@Elvans1 203_A3 %  /usr/bin/env
/Users/macbookmyduy/Library/Java/JavaVirtualMachines/openjdk-
20.0.1/Contents/Home/bin/java -XX:+ShowCo
deDetailsInExceptionMessages -cp /Users/macbookmyduy/Library/Application\
Support/Code/User/workspaceStorage/691bab4a6c836b33e8b6388c655c53e0/re
dh
at.java/jdt_ws/203_A3_d4c5a0a4/bin A3
Enter the file name: a3-sample.txt
=====================================================================
===================
The number of vertexes in the graph: 20
The number of edges in the graph: 100
The start vertexes: 2
The end vertexes: 13
=====================================================================
===================
The Euclidean distance between the start and the goal vertices: 77.87811
Shortest path: 2 -> 13
The length of the shortest path: 85
Longest path: 2 -> 17 -> 9 -> 16 -> 4 -> 18 -> 14 -> 8 -> 6 -> 19 -> 3 -> 12 -> 5 -> 20
The length of the longest path: 1595.0
=====================================================================
===================
```