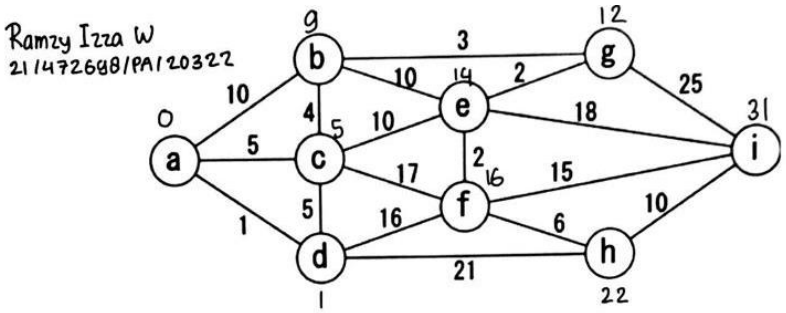
Name : Ramzy Izza Wardhana
NIM : 21/472698/PA/20322
Class : IUP CS1

Assignment 7 – Lab Algorithm & Data Structures

Zipped Complete Source Code:
https://drive.google.com/file/d/19yjil9RBsphBggQD3QnswWR1UHlpYd-2/view?usp=sharing

1. For the following graph, calculate the distance of the shortest path when the starting node is node-a and the destination node is node-i. Try calculating it manually (using pen and paper) and match the result with the code implementation. Do you get the same result?



Vertex	Steps of Dijkstra									Distance from A
A	0	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	~~∞~~	A = 0
B	~~∞~~	10	~~10~~	9	~~9~~	~~9~~	~~9~~	~~9~~	~~9~~	B = 9
C	~~∞~~	5	5	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	~~5~~	C = 5
D	~~∞~~	1	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	D = 1
E	∞	∞	~~∞~~	15	~~15~~	14	~~14~~	~~14~~	~~14~~	E = 14
F	∞	~~∞~~	17	17	17	~~17~~	16	~~16~~	~~16~~	F = 16
G	∞	∞	∞	∞	12	~~12~~	~~12~~	~~12~~	~~12~~	G = 12
H	∞	~~∞~~	22	22	22	22	22	22	~~22~~	H = 22
I	∞	∞	∞	∞	~~∞~~	~~31~~	~~31~~	31	31	I = 31
PV ⇒	A	D	C	B	g	E	F	H	i	

```
Adjacency Matrix Represtation of Undirected Weighted Graph:
   0   10    5    1    0    0    0    0    0
  10    0    4    0   10    0    3    0    0
   5    4    0    5   10   17    0    0    0
   1    0    5    0    0   16    0   21    0
   0   10   10    0    0    2    2    0   18
   0    0   17   16    2    0    0    6   15
   0    3    0    0    2    0    0    0   25
   0    0    0   21    0    6    0    0   10
   0    0    0    0   18   15   25   10    0

Distance from Node-a to Node-i =  31
PS C:\Users\themi\Documents\VSCode\DFS\DFS>
```

From the steps of calculation that I have written on paper and the output of Dijkstra implementation code given above, we can deduce that it has the same result where the distance from node a to node i using the shortest path algorithm is equal to **31.** Hence, I got the same result.

2. Try to fully understand the code and modify it so that it also prints the actual shortest path. For example, the shortest path from A to H (graph at 7.2.1) is via A-D-G-H. Show which parts of the code (and how) did you modify it

Output:

```
Distance from A to B is : 9
Traversing along the path: A C B

Distance from A to C is : 5
Traversing along the path: A C

Distance from A to D is : 1
Traversing along the path: A D

Distance from A to E is : 14
Traversing along the path: A C B G E

Distance from A to F is : 16
Traversing along the path: A C B G E F

Distance from A to G is : 12
Traversing along the path: A C B G

Distance from A to H is : 22
Traversing along the path: A D H

Distance from A to I is : 31
Traversing along the path: A C B G E F I

PS C:\Users\themi\Documents\VSCode\DFS\DFS>
```

To approach this problem, I created several new variables such as previsit, nodeParent, nearest, and range to help find the shortest path. First of all, I set the pre visit as -1 since at starting point of vertex (vertex a) parent does not exist. Now, the approach is almost the same as calculating the distance where we need to traverse the vertex one by one (greedy approach) until we find the most least distance that connected to each vertex. The most least distance will be stored in the range array which will contain the vertex with lowest weight and nearest array that will contain neighbor vertex. Then, to find the path itself, we start by using recursive method where it will loop the current vertex, then interchange it with the next destination's parent, and so on, so forth until the value of parent of current vertex is -1. Hence we will get the output of the shortest path from the explanation above.

Screenshot of Code:

```
    //find the shortest path algorithm
    for (int i = 1; i < totalNode; i++){
        //set the first previsit to -1, since there's no such parent on starting vertex
        int previsit = -1;
        //nearest vertex at initialization will be infinite
        int nearest = Integer.MAX_VALUE;
        //loop to find the shortest path by comparing the distance of nearest node
        //and the current range stored in range[j]
        for (int j = 0; j < totalNode; j++){
            //if not vertex not yet set as fixed/final, and range is closer
            //then update the value of previsit & nearest value
            if (nearest > range[j] && !computed[j] ){
                nearest = range[j];
                previsit = j;
            }
        }
        //insert to boolean array for vertex that have final value
        computed[previsit] = true;

        //iterate till final node to find the shortest path
        for (int j = 0; j < totalNode; j++){
            int edgeLength = adjacencyMatrix[previsit][j];
            //if new range is smaller then previous range, update the value
            if ((range[j] > (nearest + edgeLength)) && edgeLength > 0) {
                //set node parent to previsit
                nodeParent[j] = previsit;
                //update new range with the closer one
                range[j] = nearest + edgeLength;
            }
        }
    }
    //recall the findpath method
    findPath(start, range, nodeParent);
}
```

```
  //algorithm to find the shortest path
  public void findPath(int start, int[] distance, int[] nodeParent){
      //iterate to each node/vertex
      for (int i = 0; i < distance.length; i++) {
          if (i != start){
              //+65 to convert to ascii char
              System.out.print("Distance from A" + " to " + (char)(i+65));
              System.out.print(" is : " + distance[i]); //output the array containing distance
              System.out.print( s: "\nTraversing along the path: ");
              //invoke the display path method
              displayPath(i, nodeParent);
              System.out.println();
          }
          System.out.println();
      }
  }

  //method to display the shortest path
  public void displayPath(int currentVertex, int[] nodeParent) {
      //basecase
      if (currentVertex == -1)
          return;
      //recursion process
      displayPath(nodeParent[currentVertex], nodeParent);

      //convert int to ascii character
      int convert = currentVertex+65;
      char alphabet = (char)convert;
      //output the shortest path
      System.out.print(alphabet + " ");
  }
```