King Abdulaziz University
Faculty of Computing and Information Technology
Department of Computer Science
CPCS-324, Second semester 2020

[Mid Term Project]

CPCS324 – Algorithm and data structure – Spring2020 – Group Project I

09/04/2020 – Group #3

Student Name	Student Number
Arwa Alahamdi	1606184
Elham Saleem	1779096
Reem khalil	1505841
Ghadeer Qalas	1778773

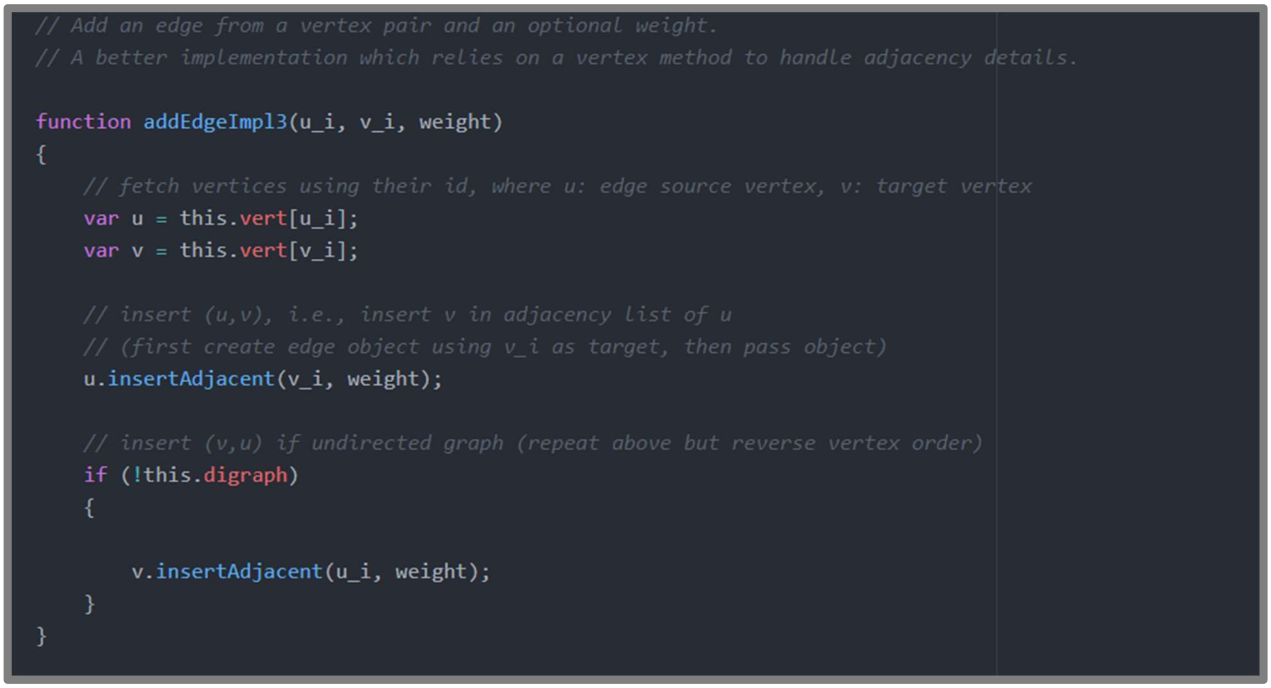
1. Introduction

In this project, we applied the different concepts of the data structure on different algorithms. The project divides into three milestones, each milestone we learned a new concept of data structure, where we got a great experience. The experience that we can summarize is how we can apply what we learned in the project with full of learning and exploring. In milestone one, we applied encapsulation concept on two methods: .makeAdjMatrix and .addEdge. In milestone two, we implemented Prim's algorithm, where compares between cites by edges in the prim method. In milestone three, here we learned a new way to implement two algorithms Prim and Dijkstra by using the Priority Queue class.

2. Milestone One:

We can say: we faced challenges in this milestone in applied encapsulation concept that we learned it, and how to do it in the project in a different way, not difficulties to solve it. In this part, the main challenge we faced is how to apply the encapsulation concept and work well with other functions in the project.

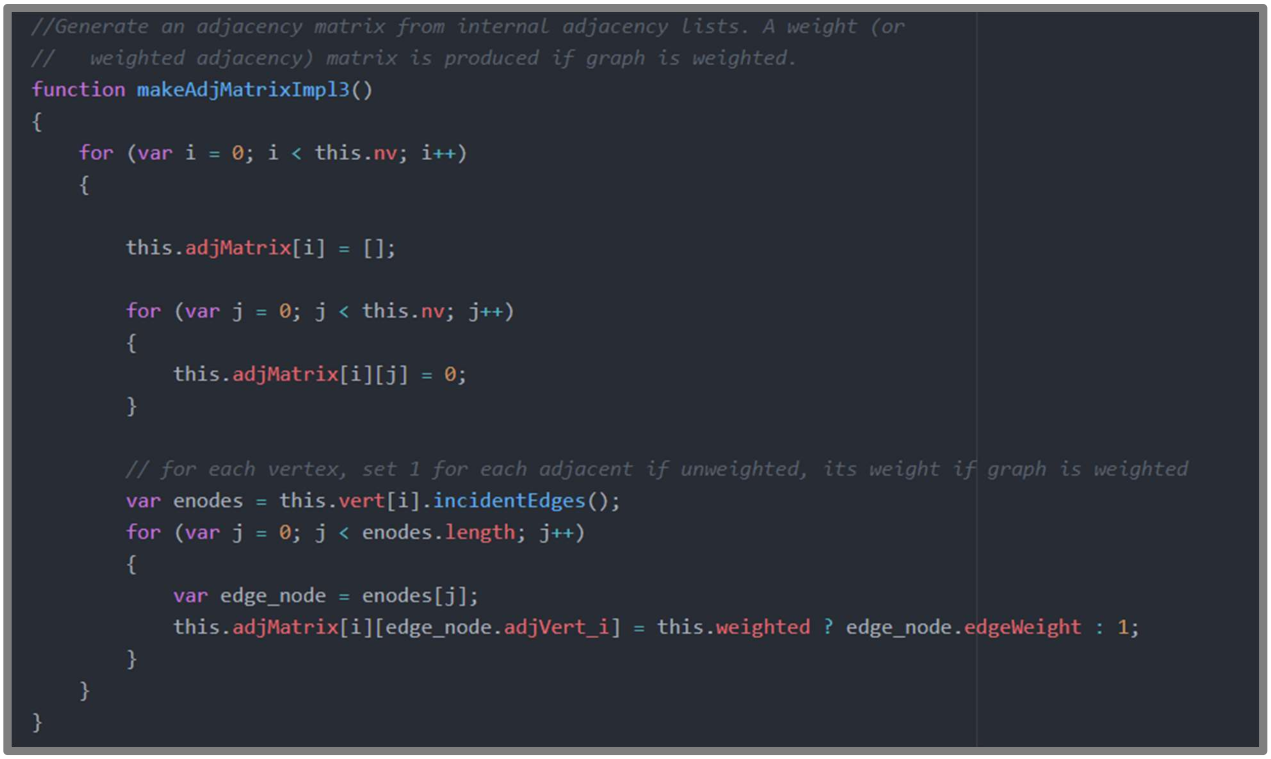
- .addEdge function:



```
// Add an edge from a vertex pair and an optional weight.
// A better implementation which relies on a vertex method to handle adjacency details.

function addEdgeImpl3(u_i, v_i, weight)
{
    // fetch vertices using their id, where u: edge source vertex, v: target vertex
    var u = this.vert[u_i];
    var v = this.vert[v_i];

    // insert (u,v), i.e., insert v in adjacency list of u
    // (first create edge object using v_i as target, then pass object)
    u.insertAdjacent(v_i, weight);

    // insert (v,u) if undirected graph (repeat above but reverse vertex order)
    if (!this.digraph)
    {

        v.insertAdjacent(u_i, weight);
    }
}
```

Figure 1: add edge function with encapsulation.

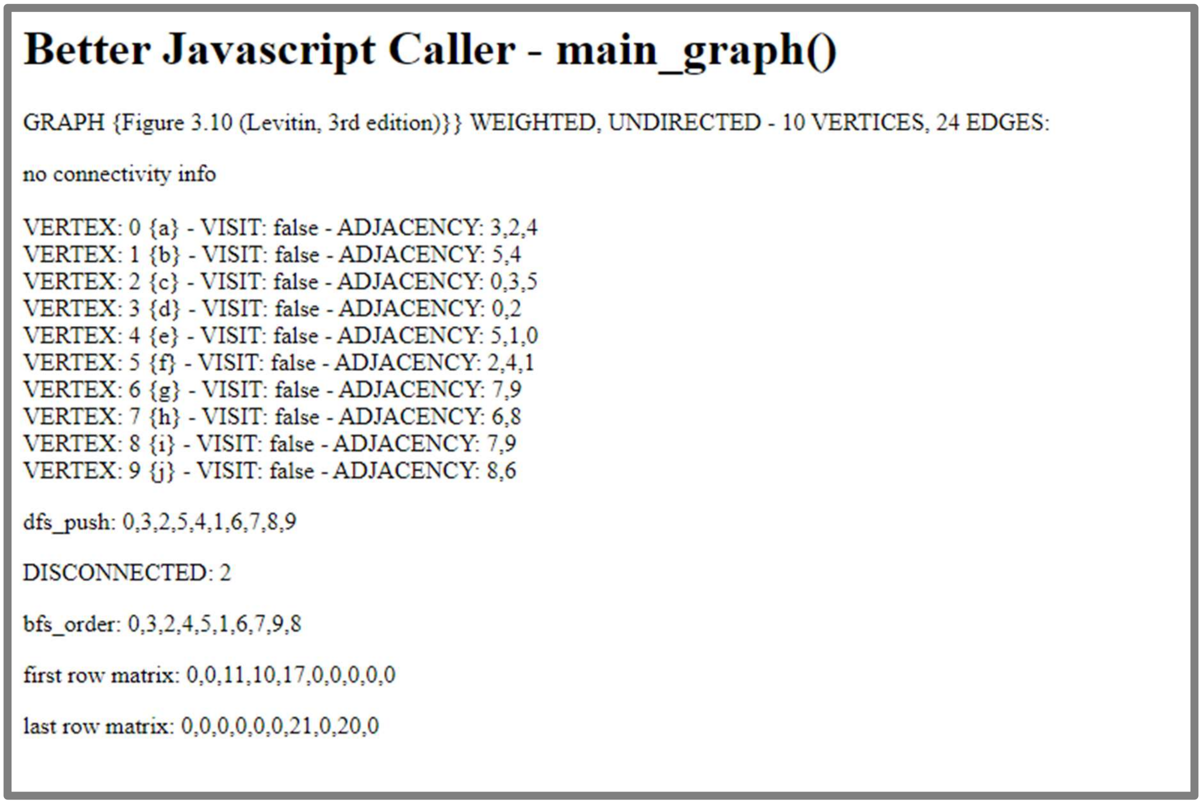
- .makeAdjMatrix function:



```
//Generate an adjacency matrix from internal adjacency lists. A weight (or
//   weighted adjacency) matrix is produced if graph is weighted.
function makeAdjMatrixImpl3()
{
    for (var i = 0; i < this.nv; i++)
    {

        this.adjMatrix[i] = [];

        for (var j = 0; j < this.nv; j++)
        {
            this.adjMatrix[i][j] = 0;
        }

        // for each vertex, set 1 for each adjacent if unweighted, its weight if graph is weighted
        var enodes = this.vert[i].incidentEdges();
        for (var j = 0; j < enodes.length; j++)
        {
            var edge_node = enodes[j];
            this.adjMatrix[i][edge_node.adjVert_i] = this.weighted ? edge_node.edgeWeight : 1;
        }
    }
}
```

Figure 2: make adjacency matrix function with encapsulation.

- The output of Milestone one:



Better Javascript Caller - main_graph()

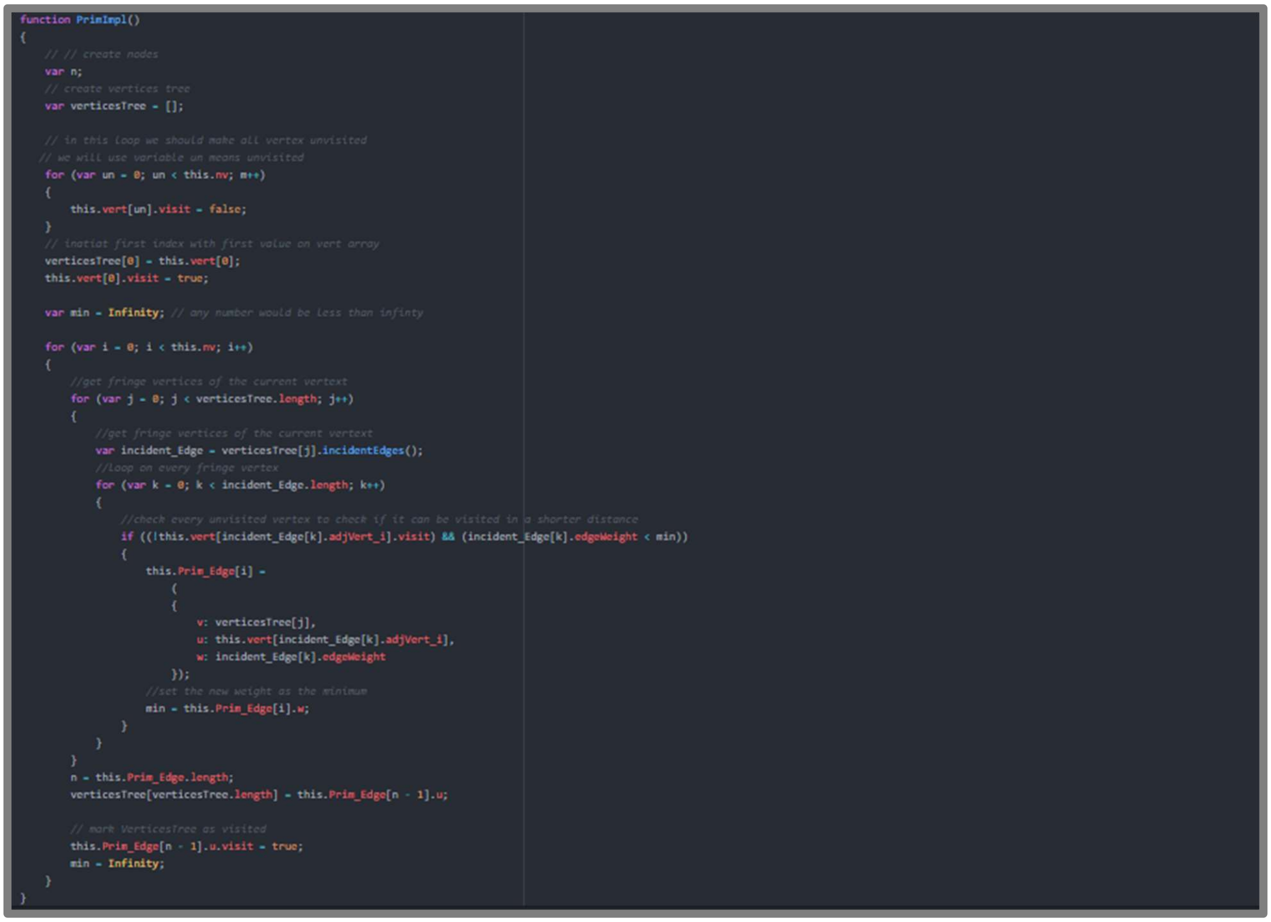
```
GRAPH {Figure 3.10 (Levitin, 3rd edition)}} WEIGHTED, UNDIRECTED - 10 VERTICES, 24 EDGES:

no connectivity info

VERTEX: 0 {a} - VISIT: false - ADJACENCY: 3,2,4
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 5,4
VERTEX: 2 {c} - VISIT: false - ADJACENCY: 0,3,5
VERTEX: 3 {d} - VISIT: false - ADJACENCY: 0,2
VERTEX: 4 {e} - VISIT: false - ADJACENCY: 5,1,0
VERTEX: 5 {f} - VISIT: false - ADJACENCY: 2,4,1
VERTEX: 6 {g} - VISIT: false - ADJACENCY: 7,9
VERTEX: 7 {h} - VISIT: false - ADJACENCY: 6,8
VERTEX: 8 {i} - VISIT: false - ADJACENCY: 7,9
VERTEX: 9 {j} - VISIT: false - ADJACENCY: 8,6

dfs_push: 0,3,2,5,4,1,6,7,8,9

DISCONNECTED: 2

bfs_order: 0,3,2,4,5,1,6,7,9,8

first row matrix: 0,0,11,10,17,0,0,0,0,0

last row matrix: 0,0,0,0,0,0,21,0,20,0
```

Figure 3: The output of milestone 1.

3. Milestone Two:

The main challenge we faced is how to apply Prim's algorithm and measure the minimum spanning-tree between cites. In this part, we learned a new way of how to use paper and pencil before applying the algorithm. It was a great experience when we think like real computer science students by analyzing algorithms before coding.

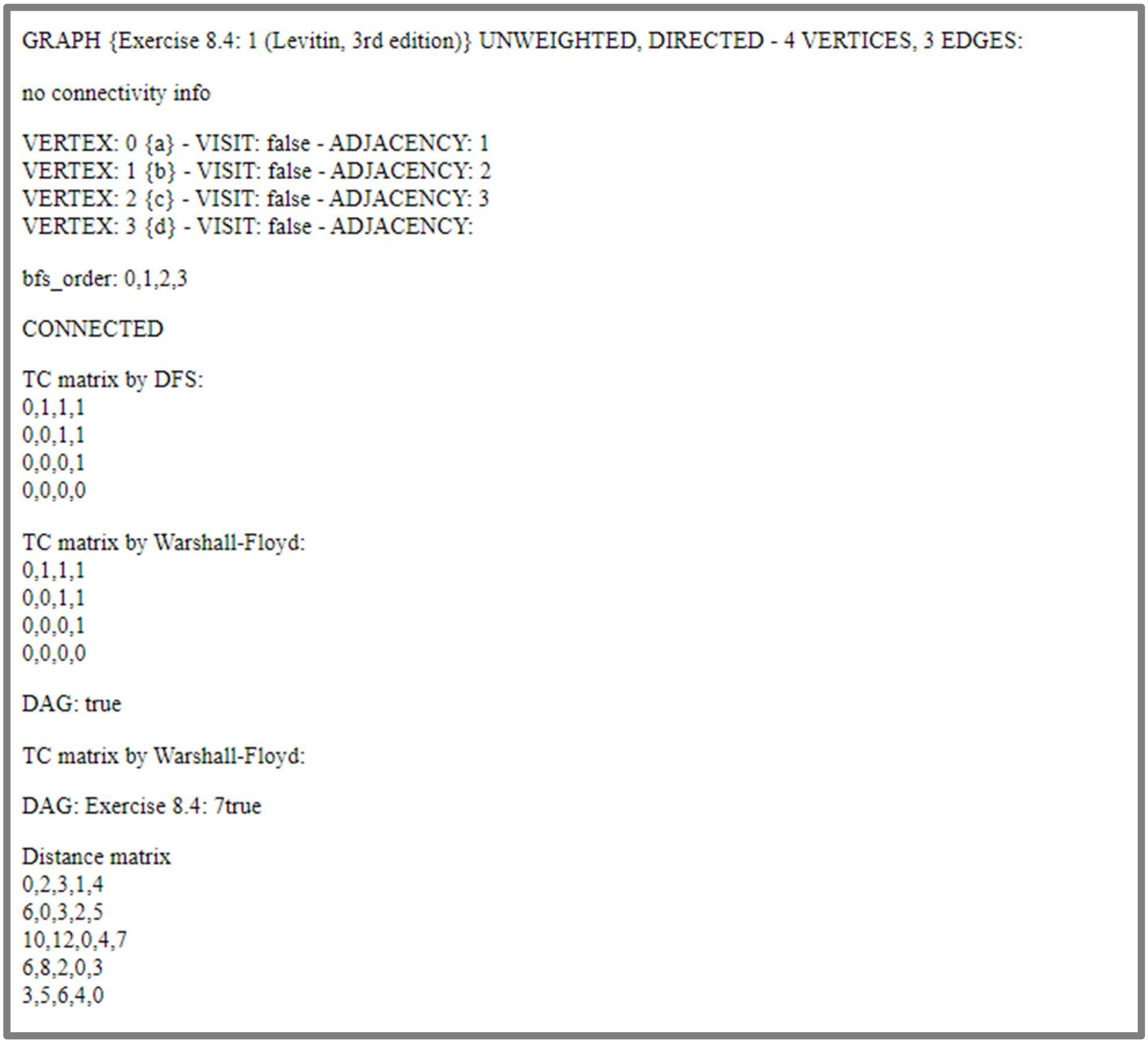
- Prim's algorithm



```
function PrimImpl()
{
    // // create nodes
    var n;
    // create vertices tree
    var verticesTree = [];

    // in this loop we should make all vertex unvisited
   // we will use variable un means unvisited
    for (var un = 0; un < this.nv; m++)
    {
        this.vert[un].visit = false;
    }
    // inatiat first index with first value on vert array
    verticesTree[0] = this.vert[0];
    this.vert[0].visit = true;

    var min = Infinity; // any number would be less than infinty

    for (var i = 0; i < this.nv; i++)
    {
        //get fringe vertices of the current vertext
        for (var j = 0; j < verticesTree.length; j++)
        {
            //get fringe vertices of the current vertext
            var incident_Edge = verticesTree[j].incidentEdges();
            //loop on every fringe vertex
            for (var k = 0; k < incident_Edge.length; k++)
            {
                //check every unvisited vertex to check if it can be visited in a shorter distance
                if ((!this.vert[incident_Edge[k].adjVert_i].visit) && (incident_Edge[k].edgeWeight < min))
                {
                    this.Prim_Edge[i] =
                        (
                        {
                            v: verticesTree[j],
                            u: this.vert[incident_Edge[k].adjVert_i],
                            w: incident_Edge[k].edgeWeight
                        });
                    //set the new weight as the minimum
                    min = this.Prim_Edge[i].w;
                }
            }
        }
        n = this.Prim_Edge.length;
        verticesTree[verticesTree.length] = this.Prim_Edge[n - 1].u;

        // mark VerticesTree as visited
        this.Prim_Edge[n - 1].u.visit = true;
        min = Infinity;
    }
}
```

- Figure 4: Prim's algorithm.

- The output of milestone two



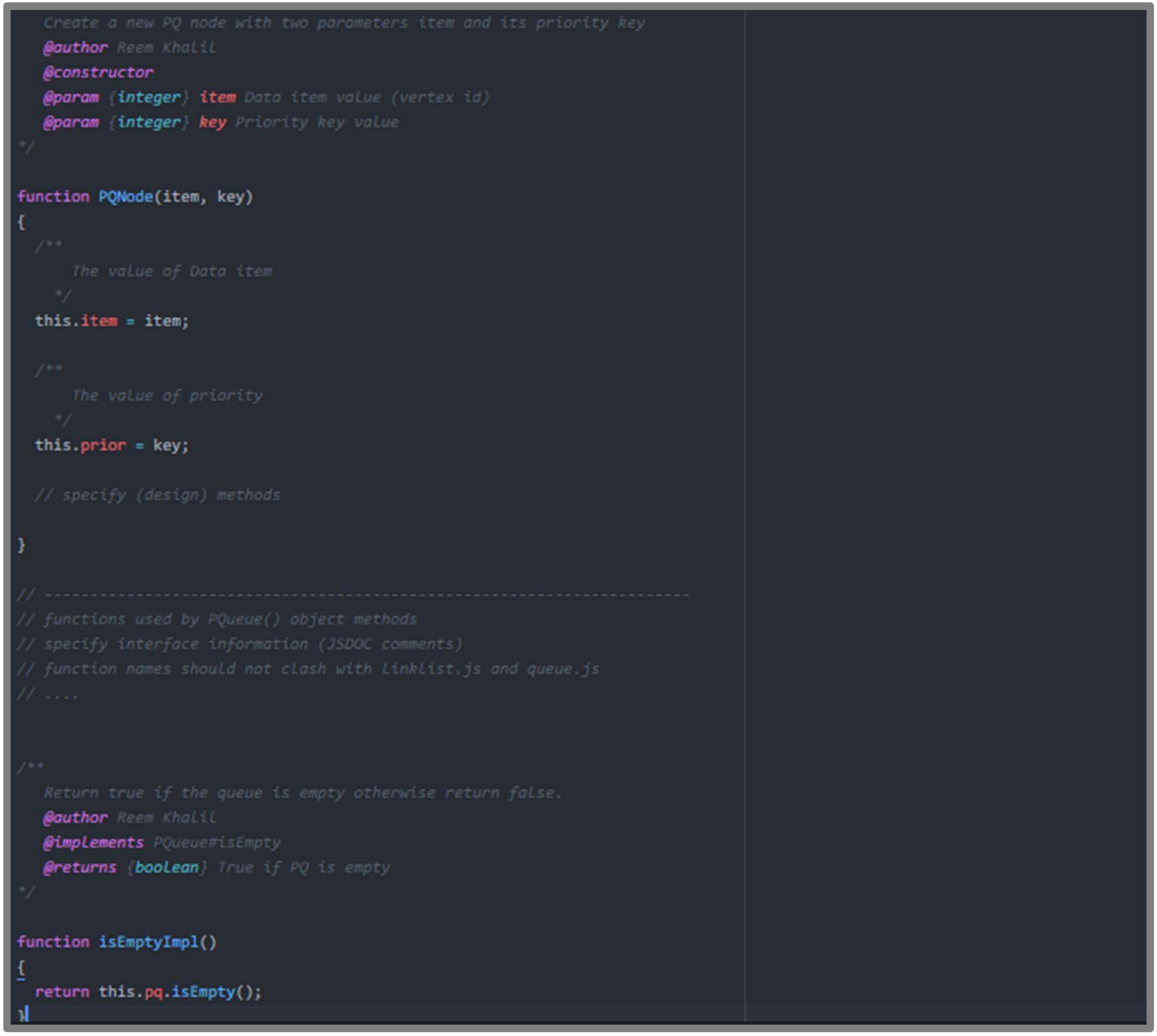
```
GRAPH {Exercise 8.4: 1 (Levitin, 3rd edition)} UNWEIGHTED, DIRECTED - 4 VERTICES, 3 EDGES:

no connectivity info

VERTEX: 0 {a} - VISIT: false - ADJACENCY: 1
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 2
VERTEX: 2 {c} - VISIT: false - ADJACENCY: 3
VERTEX: 3 {d} - VISIT: false - ADJACENCY:

bfs_order: 0,1,2,3

CONNECTED

TC matrix by DFS:
0,1,1,1
0,0,1,1
0,0,0,1
0,0,0,0

TC matrix by Warshall-Floyd:
0,1,1,1
0,0,1,1
0,0,0,1
0,0,0,0

DAG: true

TC matrix by Warshall-Floyd:

DAG: Exercise 8.4: 7true

Distance matrix
0,2,3,1,4
6,0,3,2,5
10,12,0,4,7
6,8,2,0,3
3,5,6,4,0
```

Figure 5: Output milestone 2.

4. Milestone Three:

The main challenge we faced in this part is how each member of us work and integrate the work with other members in the group. It was absolutely a great way to learn how to work with a group in a coding project. Each member of us got some challenges in her task where member 4 that challenge faced is how to apply a suitable class in the priority queue and do the function of implementation in the main class. Member 2&3: how to apply Dijkstra’s algorithm and Prim’s algorithm and integrate with the priority queue. Member 1: how to apply API docs with all algorithms.

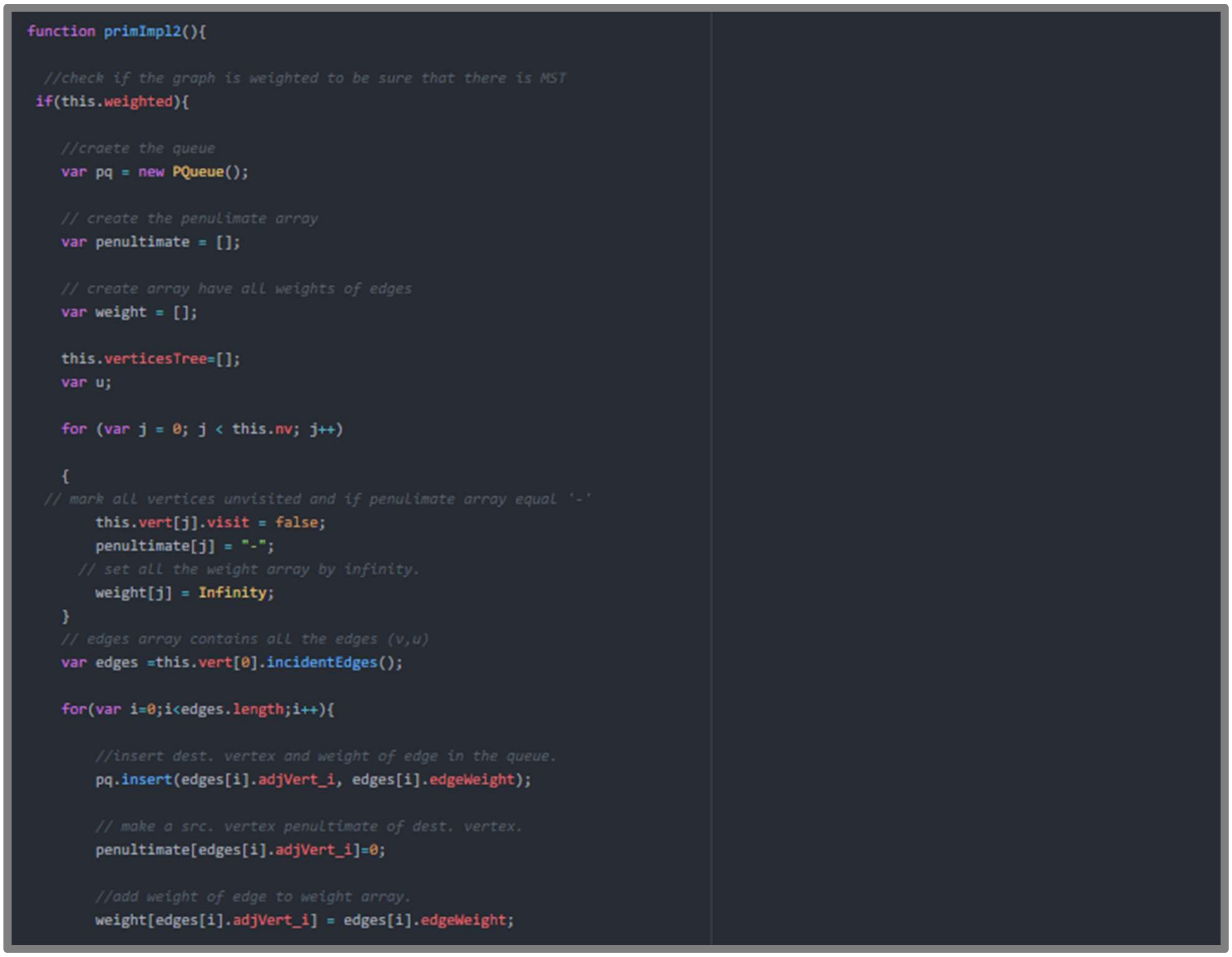
- Priority Queue



```
   Create a new PQ node with two parameters item and its priority key
   @author Reem Khalil
   @constructor
   @param {integer} item Data item value (vertex id)
   @param {integer} key Priority key value
*/

function PQNode(item, key)
{
  /**
      The value of Data item
    */
  this.item = item;

  /**
      The value of priority
    */
  this.prior = key;

  // specify (design) methods

}

// -----------------------------------------------------------------------
// functions used by PQueue() object methods
// specify interface information (JSDOC comments)
// function names should not clash with linklist.js and queue.js
// ....


/**
   Return true if the queue is empty otherwise return false.
   @author Reem Khalil
   @implements PQueue#isEmpty
   @returns {boolean} True if PQ is empty
*/

function isEmptyImpl()
{
  return this.pq.isEmpty();
}
```

Figure 6: Priority Queue Class.

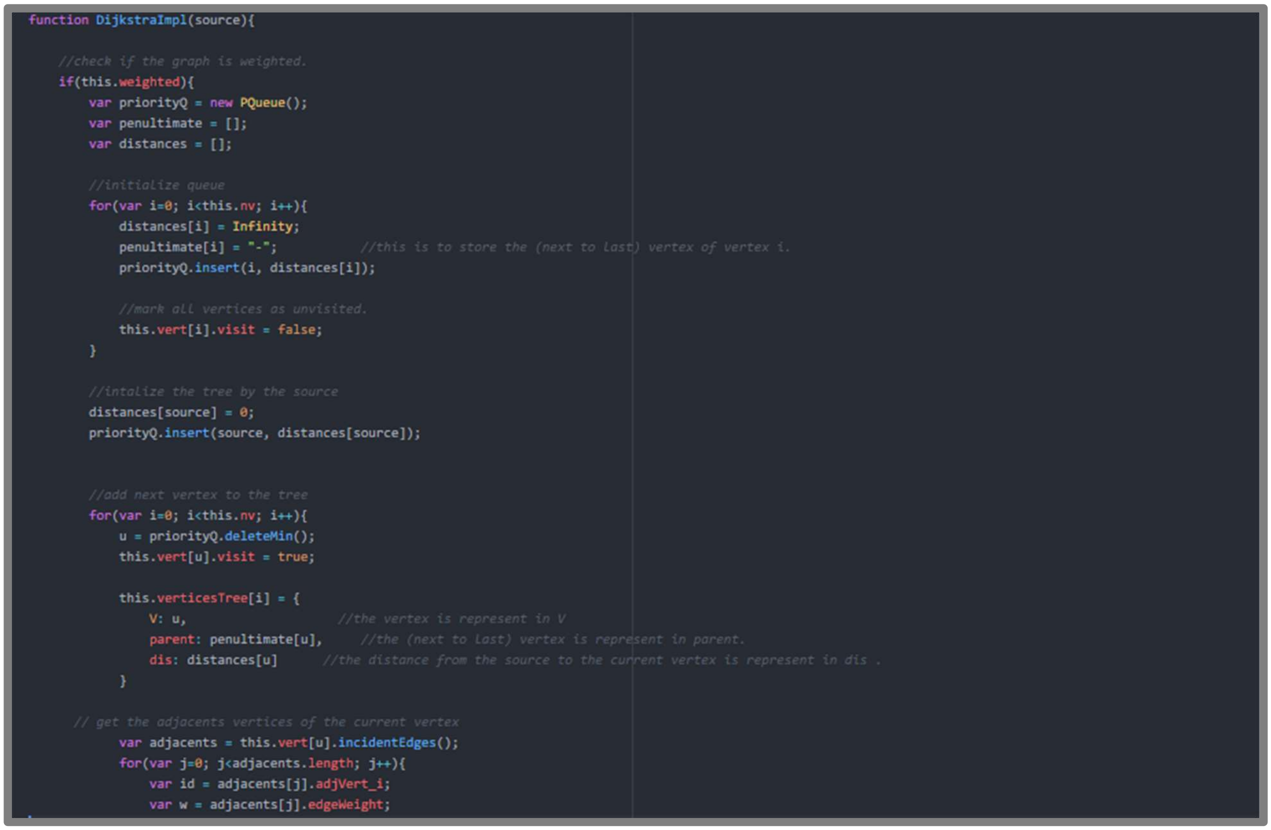
- Prim’s algorithm



```
function primImpl2(){

  //check if the graph is weighted to be sure that there is MST
 if(this.weighted){

    //craete the queue
    var pq = new PQueue();

    // create the penulimate array
    var penultimate = [];

    // create array have all weights of edges
    var weight = [];

    this.verticesTree=[];
    var u;

    for (var j = 0; j < this.nv; j++)

    {
  // mark all vertices unvisited and if penulimate array equal '-'
        this.vert[j].visit = false;
        penultimate[j] = "-";
      // set all the weight array by infinity.
        weight[j] = Infinity;
    }
    // edges array contains all the edges (v,u)
    var edges =this.vert[0].incidentEdges();

    for(var i=0;i<edges.length;i++){

        //insert dest. vertex and weight of edge in the queue.
        pq.insert(edges[i].adjVert_i, edges[i].edgeWeight);

        // make a src. vertex penultimate of dest. vertex.
        penultimate[edges[i].adjVert_i]=0;

        //add weight of edge to weight array.
        weight[edges[i].adjVert_i] = edges[i].edgeWeight;
```

Figure 7: Prim’s algorithm using PQ.

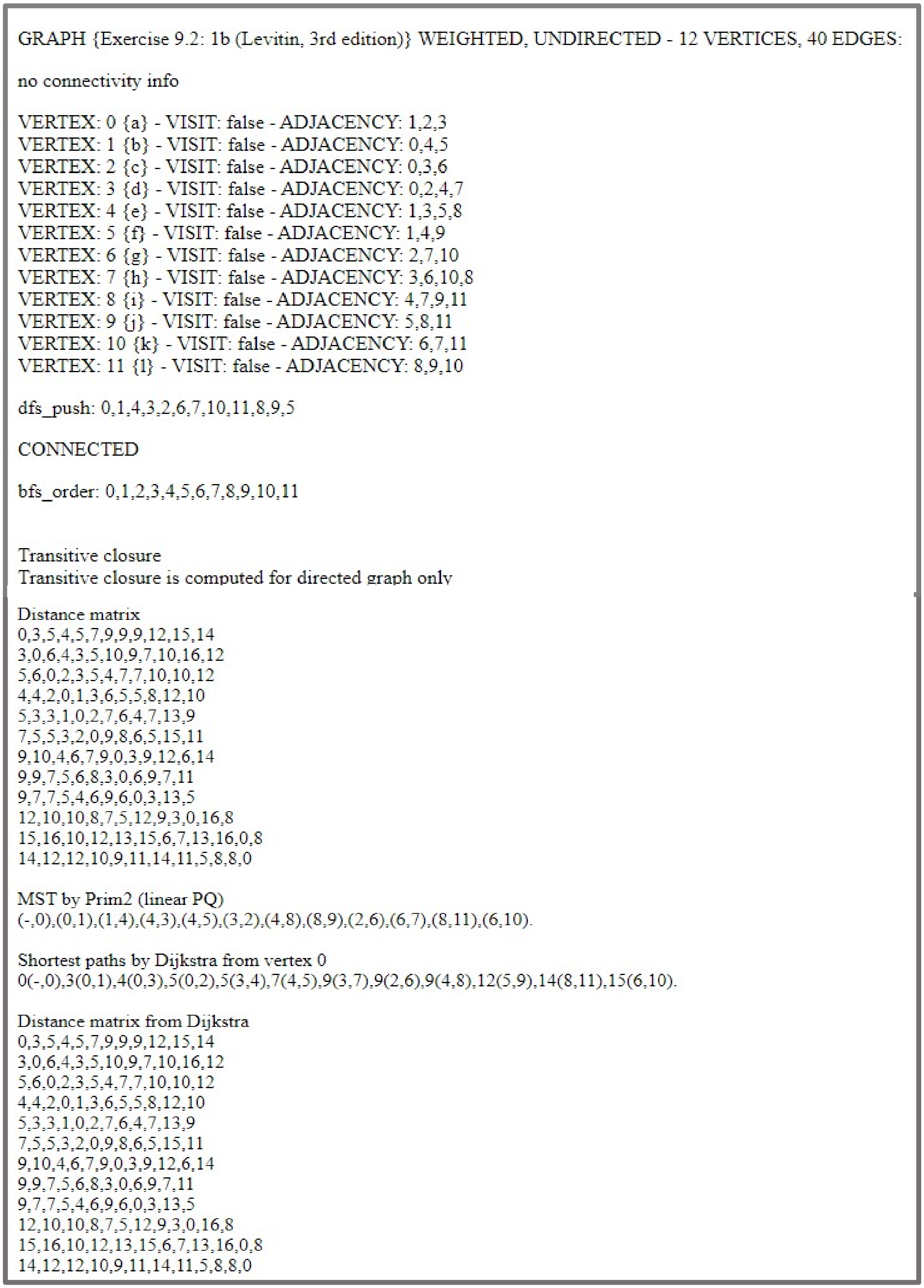
- Dijkstra’s algorithm



```
function DijkstraImpl(source){

    //check if the graph is weighted.
    if(this.weighted){
        var priorityQ = new PQueue();
        var penultimate = [];
        var distances = [];

        //initialize queue
        for(var i=0; i<this.nv; i++){
            distances[i] = Infinity;
            penultimate[i] = "-";           //this is to store the (next to last) vertex of vertex i.
            priorityQ.insert(i, distances[i]);

            //mark all vertices as unvisited.
            this.vert[i].visit = false;
        }

        //intalize the tree by the source
        distances[source] = 0;
        priorityQ.insert(source, distances[source]);


        //add next vertex to the tree
        for(var i=0; i<this.nv; i++){
            u = priorityQ.deleteMin();
            this.vert[u].visit = true;

            this.verticesTree[i] = {
                V: u,                   //the vertex is represent in V
                parent: penultimate[u],     //the (next to last) vertex is represent in parent.
                dis: distances[u]      //the distance from the source to the current vertex is represent in dis .
            }

      // get the adjacents vertices of the current vertex
            var adjacents = this.vert[u].incidentEdges();
            for(var j=0; j<adjacents.length; j++){
                var id = adjacents[j].adjVert_i;
                var w = adjacents[j].edgeWeight;
```

Figure 8: Dijkstra’s algorithm using PQ.

- The output of milestone 3



```
GRAPH {Exercise 9.2: 1b (Levitin, 3rd edition)} WEIGHTED, UNDIRECTED - 12 VERTICES, 40 EDGES:

no connectivity info

VERTEX: 0 {a} - VISIT: false - ADJACENCY: 1,2,3
VERTEX: 1 {b} - VISIT: false - ADJACENCY: 0,4,5
VERTEX: 2 {c} - VISIT: false - ADJACENCY: 0,3,6
VERTEX: 3 {d} - VISIT: false - ADJACENCY: 0,2,4,7
VERTEX: 4 {e} - VISIT: false - ADJACENCY: 1,3,5,8
VERTEX: 5 {f} - VISIT: false - ADJACENCY: 1,4,9
VERTEX: 6 {g} - VISIT: false - ADJACENCY: 2,7,10
VERTEX: 7 {h} - VISIT: false - ADJACENCY: 3,6,10,8
VERTEX: 8 {i} - VISIT: false - ADJACENCY: 4,7,9,11
VERTEX: 9 {j} - VISIT: false - ADJACENCY: 5,8,11
VERTEX: 10 {k} - VISIT: false - ADJACENCY: 6,7,11
VERTEX: 11 {l} - VISIT: false - ADJACENCY: 8,9,10

dfs_push: 0,1,4,3,2,6,7,10,11,8,9,5

CONNECTED

bfs_order: 0,1,2,3,4,5,6,7,8,9,10,11


Transitive closure
Transitive closure is computed for directed graph only

Distance matrix
0,3,5,4,5,7,9,9,9,12,15,14
3,0,6,4,3,5,10,9,7,10,16,12
5,6,0,2,3,5,4,7,7,10,10,12
4,4,2,0,1,3,6,5,5,8,12,10
5,3,3,1,0,2,7,6,4,7,13,9
7,5,5,3,2,0,9,8,6,5,15,11
9,10,4,6,7,9,0,3,9,12,6,14
9,9,7,5,6,8,3,0,6,9,7,11
9,7,7,5,4,6,9,6,0,3,13,5
12,10,10,8,7,5,12,9,3,0,16,8
15,16,10,12,13,15,6,7,13,16,0,8
14,12,12,10,9,11,14,11,5,8,8,0

MST by Prim2 (linear PQ)
(-,0),(0,1),(1,4),(4,3),(4,5),(3,2),(4,8),(8,9),(2,6),(6,7),(8,11),(6,10).

Shortest paths by Dijkstra from vertex 0
0(-,0),3(0,1),4(0,3),5(0,2),5(3,4),7(4,5),9(3,7),9(2,6),9(4,8),12(5,9),14(8,11),15(6,10).

Distance matrix from Dijkstra
0,3,5,4,5,7,9,9,9,12,15,14
3,0,6,4,3,5,10,9,7,10,16,12
5,6,0,2,3,5,4,7,7,10,10,12
4,4,2,0,1,3,6,5,5,8,12,10
5,3,3,1,0,2,7,6,4,7,13,9
7,5,5,3,2,0,9,8,6,5,15,11
9,10,4,6,7,9,0,3,9,12,6,14
9,9,7,5,6,8,3,0,6,9,7,11
9,7,7,5,4,6,9,6,0,3,13,5
12,10,10,8,7,5,12,9,3,0,16,8
15,16,10,12,13,15,6,7,13,16,0,8
14,12,12,10,9,11,14,11,5,8,8,0
```

Figure 9: The output of milestone 3

5. Conclusion

In this project, we got a great experience and learned a lot of things like teamwork, using Git and GitHub with the team. Moreover, we applied to the data structure concept and algorithms. In this experience, we got a new concept of problems in the programming world and how we can convert it to challenges to solving.

6. References

1. https://github.com/GhadeerQalas/Algorithm-Group