Graph – Adjacency Matrix **Lab Value - 100**

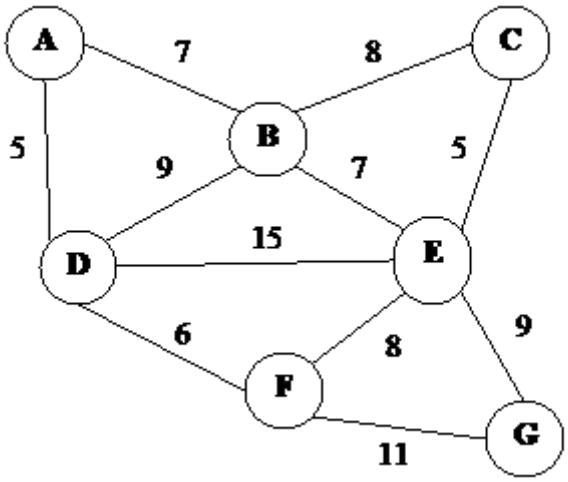
Lab Goal : This lab was designed to teach you how to solve a weighted shortest path problem using dijkstra's algorithm on an adjacency matrix.

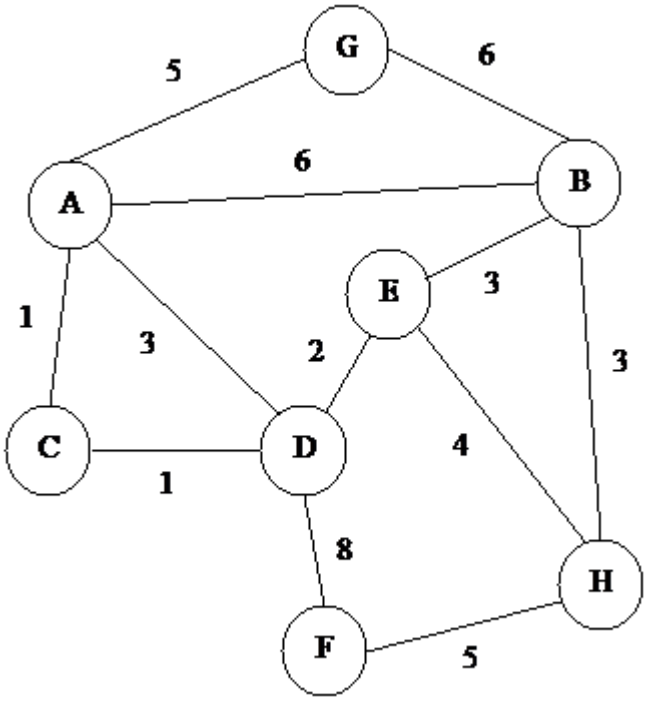
Files Needed ::

`Graph.java`

Lab Description : Given two different adjacency matrices that are provided below pictorially, find all of the shortest paths from vertex A.

Sample Data :





Sample Output :

```
Graph one:

Path from A to B:  A -> B
Path from A to C:  A -> B -> C
Path from A to D:  A -> D
Path from A to E:  A -> B -> E
Path from A to F:  A -> D -> F
Path from A to G:  A -> D -> F -> G

Graph two:

Path from A to B:  A -> B
Path from A to C:  A -> C
Path from A to D:  A -> C -> D
Path from A to E:  A -> C -> D -> E
Path from A to F:  A -> C -> D -> F
Path from A to G:  A -> G
Path from A to H:  A -> C -> D -> E -> H

If you need more help for graph one!
```

algorithm help

set all values in cost array to Integer.MAX_VALUE and set cost[0] =0
loop thru all nodes until they all are visited(n times)
{
  find the minimum path not yet visited and update currNode (one loop)
  mark currNode as visited
  update the cheapest cost and prev if applicable by looping thru current node's row of the matrix. Only check nodes where matrix entry != 0. Also, remember to only update the cost array entry if you found a cheaper route.
}

```
    A    B    C    D    E    F    G
A   0,   7,   0,   5,   0,   0,   0
B   7,   0,   8,   9,   7,   0,   0
C   0,   8,   0,   0,   5,   0,   0
D   5,   9,   0,   0,  15,   6,   0
E   0,   7,   5,  15,   0,   8,   9
F   0,   0,   0,   6,   8,   0,  11
G   0,   0,   0,   0,   9,  11,   0

prev[ 0,   0,   0,   0,   0,   0,   0]     // set up
vis [ f,   f,   f,   f,   f,   f,   f]
cost[ 0,   ∞,   ∞,   ∞,   ∞,   ∞,   ∞]
```

1. Go thru cost and find the minimum value from nodes not visited. The min cost is 0 at index 0(node A) and mark it visited.
2. Now go thru the matrix of A's row and check if the distance from A to each node plus the min cost is less than the value in the corresponding cost array. Only check the one's that haven't been visited yet as well as the one's that have a path. Update the cost and prev if applicable.

```
prev[ 0,   0,   0,   0,   0,   0,   0]     // first pass
vis [ t,   f,   f,   f,   f,   f,   f]
cost[ 0,   7,   ∞,   5,   ∞,   ∞,   ∞]  // after updating
```

3. repeat steps 1 and 2
    1. Go thru cost and find the minimum value from nodes not visited. The min cost is 5 at index 3(node D) and mark it visited.
    2. Now go thru the matrix of D's row and check if the distance from D to each node plus the min cost is less than the value in the corresponding cost array. Only check the one's that haven't been visited yet as well as the one's that have a path. Update the cost and prev if applicable.

```
prev[ 0,   0,   0,   0,   3,   3,   0]     // second pass
vis [ t,   f,   f,   t,   f,   f,   f]
cost[ 0,   7,   ∞,   5,  20,  11,   ∞]   // after updating
```

4. repeat steps 1 and 2
   1. min cost of unvisited nodes is 7 at index 1(node B) – mark it now visited
   2. go thru B's row of the matrix and check if the cost to B plus the distance from B is less than the corresponding value in the cost array. Only check the one's that haven't been visited yet as well as the one's that have a path. Update the cost and prev if applicable.

```
prev[ 0,   0,   1,   0,   1,   3,   0]     // third pass
vis [ t,   t,   f,   t,   f,   f,   f]
cost[ 0,   7,  15,   5,  14,  11,   ∞]   // after updating
```

5. repeat steps 1 and 2.

1. min cost of unvisited nodes is 11 at index 5(node F) – mark it now visited
2. go thru F's row of the matrix and check if the cost to F plus the distance from F is less than the corresponding value in the cost array. Only check the one's that haven't been visited yet as well as the one's that have a path. Update the cost and prev if applicable.

```
prev[ 0,   0,   1,   0,   1,   3,   5]     // third pass
vis [ t,   t,   f,   t,   f,   t,   f]
cost[ 0,   7,  15,   5,  14,  11,  22]   // after updating
```

6. repeat this process a total of n times

The following is the complete print out:

```
Graph one:

Verify: pass #1
[0, 0, 0, 0, 0, 0, 0]
[true, false, false, false, false, false, false]
[0, 7, 2147483647, 5, 2147483647, 2147483647, 2147483647]

Verify: pass #2
[0, 0, 0, 0, 3, 3, 0]
[true, false, false, true, false, false, false]
[0, 7, 2147483647, 5, 20, 11, 2147483647]

Verify: pass #3
[0, 0, 1, 0, 1, 3, 0]
[true, true, false, true, false, false, false]
[0, 7, 15, 5, 14, 11, 2147483647]

Verify: pass #4
[0, 0, 1, 0, 1, 3, 5]
[true, true, false, true, false, true, false]
[0, 7, 15, 5, 14, 11, 22]

Verify: pass #5
[0, 0, 1, 0, 1, 3, 5]
[true, true, false, true, true, true, false]
[0, 7, 15, 5, 14, 11, 22]

Verify: pass #6
[0, 0, 1, 0, 1, 3, 5]
[true, true, true, true, true, true, false]
[0, 7, 15, 5, 14, 11, 22]

Verify: pass #7
[0, 0, 1, 0, 1, 3, 5]
[true, true, true, true, true, true, true]
[0, 7, 15, 5, 14, 11, 22]
```