Algorithm: Assignment 2:
Name: Abhishek Sharma
Professor: Mohan Kumar

Explanation For Algorithms:

1) Dijkstra's algorithm: for calculating single source shortest path

   Data set Used: Array- to store the Nodes and their adjacent node list.
   For making the Breath First Search I maintained the array list corresponding to each node. This will give the running time complexity of O(V+E).
   Second array is for keeping the list of all node for the traversal.
   Maintained one index respective to each node to keep track which nodes have been visited.

2) Kruskal's Algorithm: Minimum Spanning Tree:

   Data set used: array: to keep the record of all the edges
   Maintained union-find data structure to have minimum time complexity to check for disjoint vertex. This will give the time complexity of find() function approximately O(1).
   Maintained one Index for Root of set to each node to check in which set it belongs, and one index.

3) Floyd Warshall's: All Pairs Shortest Paths:
   Data set used: Maintained two dimensional array for keep the adjacency matrix.

4) Transitive Closure :
   Data set used: Maintained two dimensional array for keep the adjacency matrix.

Note:

- For data set generator I used the reference provided.
- User need to provide the number of vertices and edges at command line in same order. I maintained a two dimensional array (matrix[][]) to sore the data matrix get generated and while generating the data we need to keep the sequence of each data set generated and its calculation first before the second dataset get generated.
- Below are the table for providing the CPU time taken by each algorithm for each dataset
- Each time is in milliseconds.

Table:

	Case 1	Case 2	Case 3
CPU Running Time For : Dijkstra's algorithm	For connection <10 Avg CPU Time = 1	For Connections>N/2 a) =40 : Avg Time: 0 b) = 60: Avg Time: 0 C) = 100: Avg Time: 1	For Random Graph: for: 150 Nodes a)weighted, directed: 42 b) weighted,Non-directed :50 c) nonweighted, directed :26 d) nonweighted,Non-directed :18

CPU Running Time For : Kruskal's algorithm	For connection <10 Avg CPU Time = 1	For Connections>N/2 a) =40 : Avg Time: 0 b) = 60: Avg Time: 0 C) = 100: Avg Time: 1	For Random Graph: for: 150 Nodes a)weighted, directed: 30 b) weighted, Non-directed: 25 c) nonweighted, directed :37 d) nonweighted,Non-directed :18

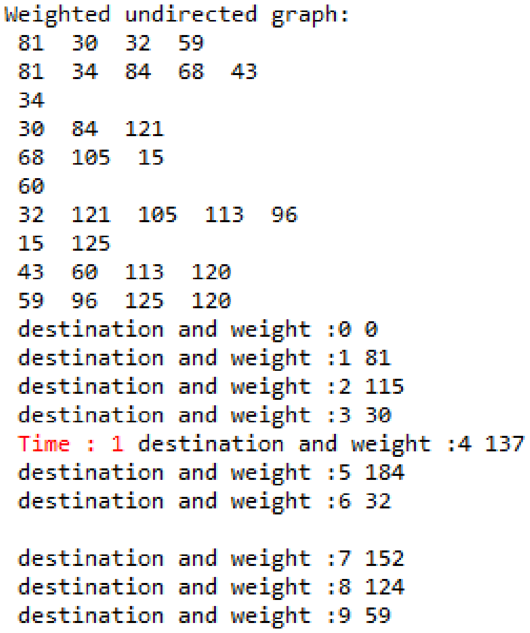
CPU Running Time For : Floyd Warshall	For connection <10 Avg CPU Time = 1	For Connections>N/2 a) =40 : Avg Time: 9 b) = 60: Avg Time: 16 C) = 100:Avg Time:126	For Random Graph: for: 150 Nodes a)weighted, directed: 210 b) weighted,Non-directed:200 c) non weighted, directed :190 d) non-weighted,Non-directed :200

CPU Running Time For : Transitive Clouser	For connection <10 Avg CPU Time = 1	For Connections>N/2 a) =40 : Avg Time: 11 b) = 60: Avg Time: 15 C) = 100:Avg Time:140	For Random Graph: for: 150 Nodes a)weighted, directed: 200 b) weighted,Non-directed:205 c) non weighted, directed :190 d) non-weighted,Non-directed :180

Example execution:

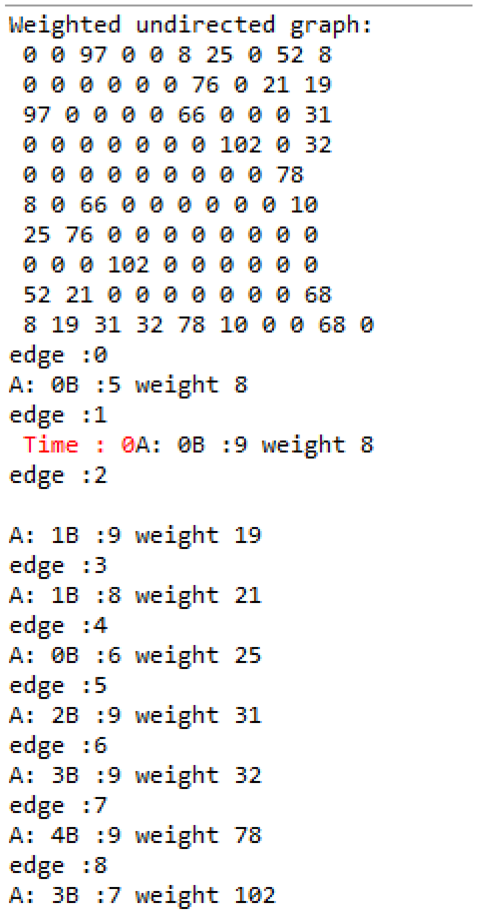
Weighted undirected graph: for nodes = 10 edge = 16

1) Dijkstra's Algorithm:
The out-put representation of minimum traversal weight from source (0) to each other vertices   :



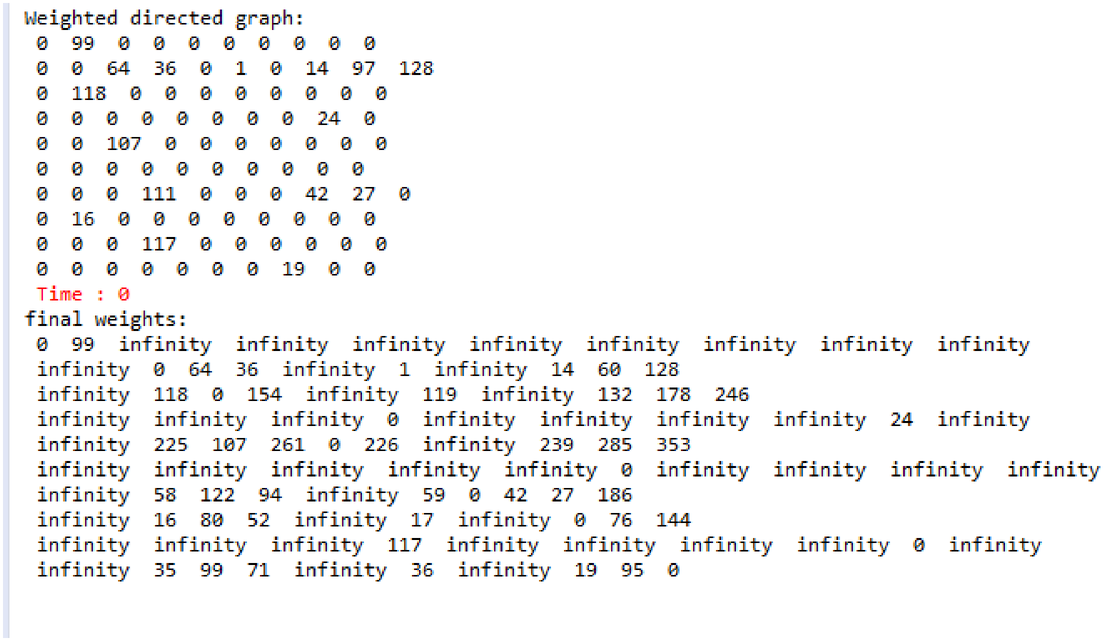
```
Weighted undirected graph:
 81  30  32  59
 81  34  84  68  43
 34
 30  84  121
 68  105  15
 60
 32  121  105  113  96
 15  125
 43  60  113  120
 59  96  125  120
 destination and weight :0 0
 destination and weight :1 81
 destination and weight :2 115
 destination and weight :3 30
 Time : 1 destination and weight :4 137
 destination and weight :5 184
 destination and weight :6 32

 destination and weight :7 152
 destination and weight :8 124
 destination and weight :9 59
```

Kruskals algorithm:
The representation of out put is which all edges has been included, then its two vertices (A and B) and then the weight for that edge.



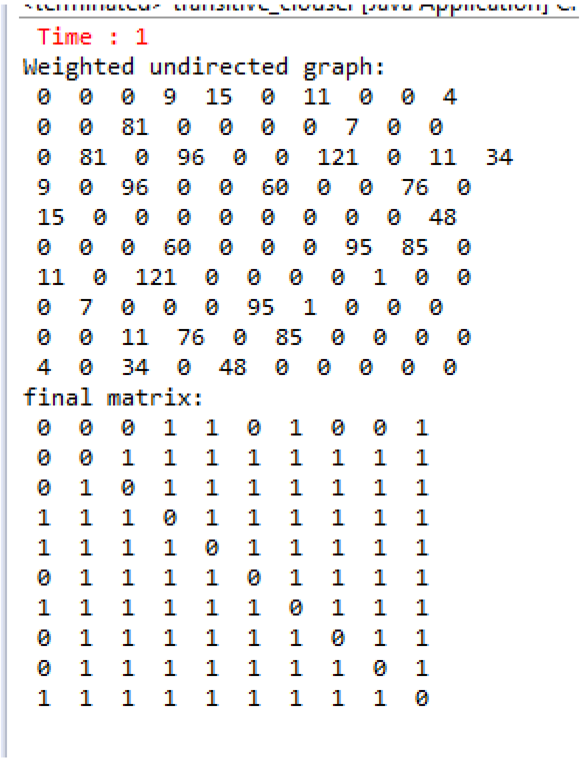
```
Weighted undirected graph:
 0 0 97 0 0 8 25 0 52 8
 0 0 0 0 0 0 76 0 21 19
 97 0 0 0 0 66 0 0 0 31
 0 0 0 0 0 0 0 102 0 32
 0 0 0 0 0 0 0 0 0 78
 8 0 66 0 0 0 0 0 0 10
 25 76 0 0 0 0 0 0 0 0
 0 0 0 102 0 0 0 0 0 0
 52 21 0 0 0 0 0 0 0 68
 8 19 31 32 78 10 0 0 68 0
edge :0
A: 0B :5 weight 8
edge :1
 Time : 0A: 0B :9 weight 8
edge :2

A: 1B :9 weight 19
edge :3
A: 1B :8 weight 21
edge :4
A: 0B :6 weight 25
edge :5
A: 2B :9 weight 31
edge :6
A: 3B :9 weight 32
edge :7
A: 4B :9 weight 78
edge :8
A: 3B :7 weight 102
```

Floyd Warshall Algorithm:
The out put representation is simply in form of matrix which gives the minimum weight between the two nodes.



```
Weighted directed graph:
 0  99  0  0  0  0  0  0  0  0
 0  0  64  36  0  1  0  14  97  128
 0  118  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  24  0
 0  0  107  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0
 0  0  0  111  0  0  0  42  27  0
 0  16  0  0  0  0  0  0  0  0
 0  0  0  117  0  0  0  0  0  0
 0  0  0  0  0  0  0  19  0  0
 Time : 0
final weights:
 0  99  infinity  infinity  infinity  infinity  infinity  infinity  infinity  infinity
 infinity  0  64  36  infinity  1  infinity  14  60  128
 infinity  118  0  154  infinity  119  infinity  132  178  246
 infinity  infinity  infinity  0  infinity  infinity  infinity  infinity  24  infinity
 infinity  225  107  261  0  226  infinity  239  285  353
 infinity  infinity  infinity  infinity  infinity  0  infinity  infinity  infinity  infinity
 infinity  58  122  94  infinity  59  0  42  27  186
 infinity  16  80  52  infinity  17  infinity  0  76  144
 infinity  infinity  infinity  117  infinity  infinity  infinity  infinity  0  infinity
 infinity  35  99  71  infinity  36  infinity  19  95  0
```

Transitive Clouser:
In this also the out-put is the matrix describing if the there is path between any two pair of vertices.



```
<terminated> transitive_clouser [Java Application] C:
 Time : 1
Weighted undirected graph:
 0  0  0  9  15  0  11  0  0  4
 0  0  81  0  0  0  0  7  0  0
 0  81  0  96  0  0  121  0  11  34
 9  0  96  0  0  60  0  0  76  0
 15  0  0  0  0  0  0  0  0  48
 0  0  0  60  0  0  0  95  85  0
 11  0  121  0  0  0  0  1  0  0
 0  7  0  0  0  95  1  0  0  0
 0  0  11  76  0  85  0  0  0  0
 4  0  34  0  48  0  0  0  0  0
final matrix:
 0  0  0  1  1  0  1  0  0  1
 0  0  1  1  1  1  1  1  1  1
 0  1  0  1  1  1  1  1  1  1
 1  1  1  0  1  1  1  1  1  1
 1  1  1  1  0  1  1  1  1  1
 0  1  1  1  1  0  1  1  1  1
 1  1  1  1  1  1  0  1  1  1
 0  1  1  1  1  1  1  0  1  1
 0  1  1  1  1  1  1  1  0  1
 1  1  1  1  1  1  1  1  1  0
```