```
This is the implementation of Dijkstra`s algorithm and Kruskal algorithm for the graph defined in data.txt
----- Data.txt----
5 9 D
A B 10
A C 3
B C 1
B D 2
C B 4
C D 8
C E 2
D E 7
E D 9
Here, the first two numbers represent the number of vertices and edges. The letter D stands for
Directed Acyclic graph. This letter might be U meaning undirected graph. From the second line on we have
edges and its weight (e.g. edge(A,B) and its weight is 10. The last line is optional. If given, it
represents the source node.
------------------

Data structures used
-----------------------

The code is using lists, dictonary and set data structures to hold approprates graph data.
the nodes are held in lists and the edges are held in list.
the combination of edges and distances are held in dictonary.

Input
-----
Run the graph.py.
Dijkstra`s - The code will find the shortest path and print the cost table and paths involved.
Kruskal - The code will find the minimum spanning tree and print all the nodes involved in minimum
spanning tree as a set.
-----
Output
-----
==========================================
        Dijkstra`s Algorithm analysis
==========================================
----------------------------------
Current node selected: A
neighbor of A : ['B', 'C']
distance between A and B : 10
distance between A and C : 3
visited: [1, 1, 1, 0, 0]
Path: ['A']
Cost: [0, 10, 3, 999, 999]
----------------------------------
Current node selected: C
neighbor of C : ['B', 'D', 'E']
distance between C and B : 4
distance between C and D : 8
distance between C and E : 2
visited: [1, 1, 1, 1, 1]
Path: ['A', 'C']
Cost: [0, 7, 3, 11, 5]
----------------------------------
Current node selected: E
neighbor of E : ['D']
distance between E and D : 9
visited: [1, 1, 1, 1, 1]
Path: ['A', 'C', 'E']
Cost: [0, 7, 3, 11, 5]
----------------------------------
Current node selected: D
neighbor of D : ['E']
distance between D and E : 7
visited: [1, 1, 1, 1, 1]
Path: ['A', 'C', 'E', 'D']
```

```
Cost: [0, 7, 3, 11, 5]
==========================================
        Kruskal`s Algorithm analysis
==========================================
Checking edge:  (1, 'B', 'C')
adding  (1, 'B', 'C') to minimum spanning tree
------------------------------------------
Checking edge:  (2, 'B', 'D')
adding  (2, 'B', 'D') to minimum spanning tree
------------------------------------------
Checking edge:  (2, 'C', 'E')
adding  (2, 'C', 'E') to minimum spanning tree
------------------------------------------
Checking edge:  (3, 'A', 'C')
adding  (3, 'A', 'C') to minimum spanning tree
------------------------------------------
Checking edge:  (4, 'C', 'B')
Edge  (4, 'C', 'B') is not added to minimum spanning tree
------------------------------------------
Checking edge:  (7, 'D', 'E')
Edge  (7, 'D', 'E') is not added to minimum spanning tree
------------------------------------------
Checking edge:  (8, 'C', 'D')
Edge  (8, 'C', 'D') is not added to minimum spanning tree
------------------------------------------
Checking edge:  (9, 'E', 'D')
Edge  (9, 'E', 'D') is not added to minimum spanning tree
------------------------------------------
Checking edge:  (10, 'A', 'B')
Edge  (10, 'A', 'B') is not added to minimum spanning tree
------------------------------------------
==========================================
        Dijkstra`s Algorithm Report
==========================================
The final path is: ['A', 'C', 'E', 'D']
The final costs are: [0, 7, 3, 11, 5]
Not a dag the execution terminates here.
Total execution time: 0.000759124755859  Seconds
==========================================
        Kruskal`s Algorithm Report
==========================================
The minimum spanning tree is a graph contaning set([(3, 'A', 'C'), (2, 'B', 'D'), (2, 'C', 'E'), (1, 'B',
'C')])
Total execution time: 0.000321865081787  Seconds
-----
```