Part A

Aim:

1. Greedy Method
2. Single Source Shortest Path using Dijkstra algorithm.

Prerequisite: Any programming language

Outcome: Algorithms and their implementation

Theory:

Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.

Procedure:

1. Design algorithm and find best, average and worst-case complexity
2. Implement algorithm in any programming language.
3. Paste output

Practice Exercise:

S.no	Statement
1	Implement the Dijkstra Algorithm.
2	Find the run time complexity of the above algorithm

Instructions:

1. Design, analysis and implement the algorithms.
2. Paste the snapshot of the output in input & output section.

Part B

Algorithm : DIJKSTRA's algorithm
Input : Graph with 0 as source node
Output : Shortest path tree (shortest path to other nodes of the graph)

Algorithm:

```
def dijktras(source):
    assign each element of distance list(length of vertices) to 999999
    assign distance[source] to 0
    visited=[False]*vertices

    for i in range(vertices):
        min=999999
        for j in range(vertices):
            if distance[j]<min and visited[j]==False:
                min_vertex=i
                min=distance[j]

        visited[min_vertex]=True
        for j in range(vertices):
            if graph[min_vertex][j]>0 and visited[j]==False and
distance[j]>distance[min_vertex]+graph[min_vertex][j],
         then
                distance[j]=distance[min_vertex]+graph[min_vertex][j]

    print the distances obtained for each vertice using a for loop
    for i in range(vertices):
        print(i,distance[i])
```

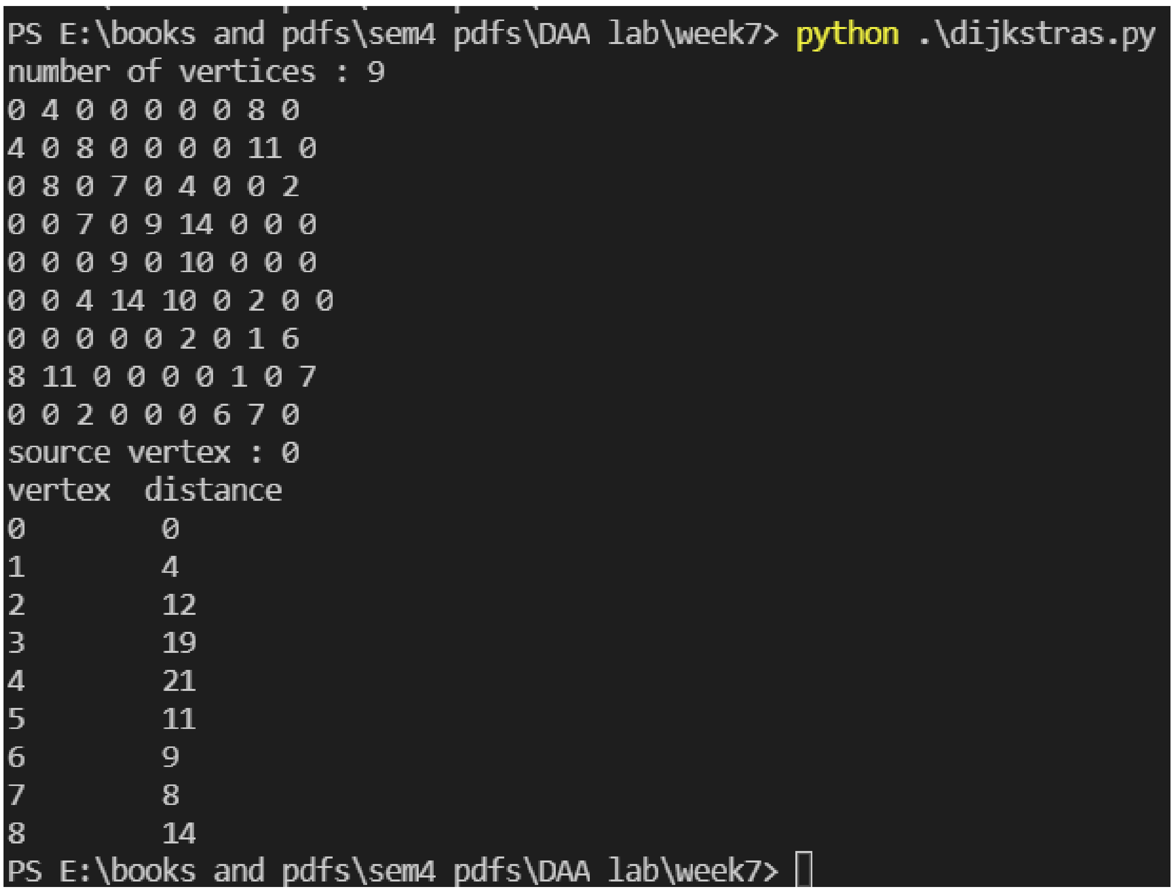
Code:

```
def dijktras(src):
   global vertices,graph
   distance=[999999]*vertices
   distance[src]=0
   visited=[False]*vertices
   for i in range(vertices):
      min=999999
      for j in range(vertices):
         if distance[j]<min and visited[j]==False:
            min_vertex=j
            min=distance[j]
      visited[min_vertex]=True

      for j in range(vertices):
         if graph[min_vertex][j]>0 and visited[j]==False and
distance[j]>distance[min_vertex]+graph[min_vertex][j]:
            distance[j]=distance[min_vertex]+graph[min_vertex][j]
   print('vertex\tdistance ')
   for i in range(vertices):
      print(i,"\t",distance[i])

vertices=int(input('number of vertices : '))
graph=[list(map(int,input().split())) for i in range(vertices)]
dijktras(int(input('source vertex : ')))
```

Output:



```
PS E:\books and pdfs\sem4 pdfs\DAA lab\week7> python .\dijkstras.py
number of vertices : 9
0 4 0 0 0 0 0 8 0
4 0 8 0 0 0 0 11 0
0 8 0 7 0 4 0 0 2
0 0 7 0 9 14 0 0 0
0 0 0 9 0 10 0 0 0
0 0 4 14 10 0 2 0 0
0 0 0 0 0 2 0 1 6
8 11 0 0 0 0 1 0 7
0 0 2 0 0 0 6 7 0
source vertex : 0
vertex  distance
0        0
1        4
2        12
3        19
4        21
5        11
6        9
7        8
8        14
PS E:\books and pdfs\sem4 pdfs\DAA lab\week7>
```

Run time complexity of the above algorithm

The time complexity of the above algorithm is O(V^2) as it is implemented using an adjacency list.
But it can be reduced to O(E log V) if we use binary heap.

Time complexity will be O(V) +O(E log V) where O(V) is obtained by visiting all nodes and O(log V) for relaxation of 1 node. As we have E such nodes, it is O(E log V)
---> O(V) +O(E log V) = O(E log V)

Therefore, Dijkstra's shortest path algorithm's time complexity is O(ElogV) where:
V is the number of vertices
E is the total number of edges

Space complexity of Dijkstra's algorithm:

Using adjacency list, Space complexity s O(V^2)

Observation & Learning:

I have observed and learned that :
i) Dijkstra's algorithm doesn't work for graphs with negative weight cycles, it may or may not give correct results for a graph with negative edges .

ii) For negative edges Bellman-Ford algorithm can be used.

iii)Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree

Conclusion:

I have successfully implemented Dijkstra's algorithm in the python programming language.