UCS1403 Design and Analysis of Algorithms
AY: 2021-22

Name: *Krithika Swaminathan*
Reg. no.: *205001057*

DAA Lab – Assignment 7

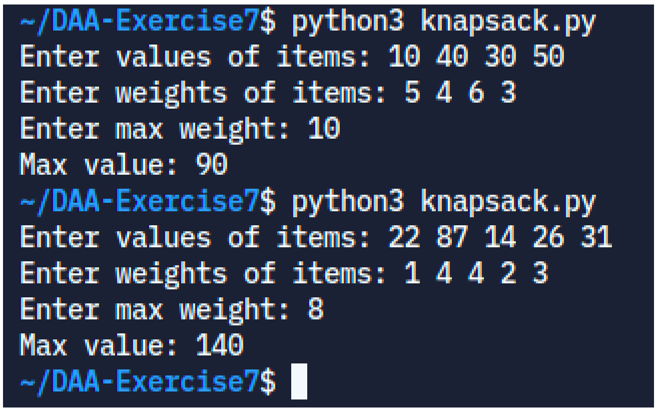
Greedy Technique and Dynamic Programming

1. To Implement Knapsack Algorithm using DP.

Code:

```
# Python program to implement the Knapsack problem using Dynamic Programming

def knapSack(W, wt, val, n):
      K = [[0 for x in range(W + 1)] for x in range(n + 1)]

      for i in range(n + 1):
            for w in range(W + 1):
                  if i == 0 or w == 0:
                        K[i][w] = 0
                  elif wt[i-1] <= w:
                        K[i][w] = max(val[i-1]
                                    + K[i-1][w-wt[i-1]],
                                          K[i-1][w])
                  else:
                        K[i][w] = K[i-1][w]

      return K[n][W]

val = list(map(int,input("Enter values of items: ").split()))
wt = list(map(int,input("Enter weights of items: ").split()))
W = int(input("Enter max weight: "))
print("Max value:",knapSack(W, wt, val, len(val)))
```

Output:



```
~/DAA-Exercise7$ python3 knapsack.py
Enter values of items: 10 40 30 50
Enter weights of items: 5 4 6 3
Enter max weight: 10
Max value: 90
~/DAA-Exercise7$ python3 knapsack.py
Enter values of items: 22 87 14 26 31
Enter weights of items: 1 4 4 2 3
Enter max weight: 8
Max value: 140
~/DAA-Exercise7$
```

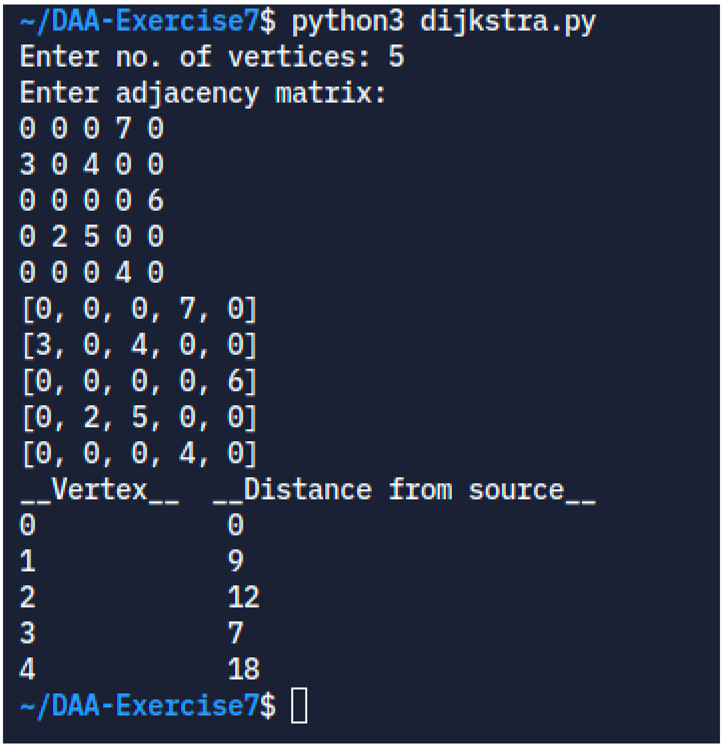
2. To Implement Dijkstra’s Algorithm for Shortest Path Algorithm.

Code:

```
# Program to implement Djikstra's algorithm using Greedy approach

inf = 99

def getAdj(n):
	print("Enter adjacency matrix: ")
	graph = []
	for i in range(n):
		row = list(map(int,input().split()))
		if (len(row) != n):
			print("Invalid no. of columns entered. Enter again.")
			i-=1
		else:
			graph.append(row)
	return graph

def printGraph(graph):
	for i in range(n):
		print(graph[i])

def printSolution(graph,dist):
	n = len(graph)
	print("__Vertex__ \t__Distance from source__")
	for vertex in range(n):
		print(vertex,"\t\t\t",dist[vertex])

def dijPath(graph,src):
	n = len(graph)
	dist = [inf for i in range(n)]
	path = [False for i in range(n)]
	dist[src] = 0

	for i in range(n):
		min = inf
		for vertex in range(n):
			if (dist[vertex] < min and path[vertex] == False):
				min = dist[vertex]
				u = vertex

		path[u] = True
		for v in range(n):
			if (graph[u][v]>0 and path[v]==False and dist[v]>dist[u]
+graph[u][v]):
				dist[v] = dist[u]+graph[u][v]

	printSolution(graph,dist)


n = int(input("Enter no. of vertices: "))
g = getAdj(n)
```

```
printGraph(g)
dijPath(g,0)
```

Output:



```
~/DAA-Exercise7$ python3 dijkstra.py
Enter no. of vertices: 5
Enter adjacency matrix:
0 0 0 7 0
3 0 4 0 0
0 0 0 0 6
0 2 5 0 0
0 0 0 4 0
[0, 0, 0, 7, 0]
[3, 0, 4, 0, 0]
[0, 0, 0, 0, 6]
[0, 2, 5, 0, 0]
[0, 0, 0, 4, 0]
__Vertex__  __Distance from source__
0            0
1            9
2            12
3            7
4            18
~/DAA-Exercise7$
```