DESIGN AND ANALYSIS OF ALGORITHMS

LAB MANUAL

Academic Year	**:**	**2019 - 2020**
Course Code	**:**	**AITB07**
Regulations	**:**	**IARE - R18**
Semester	**:**	**IV**
Branch	**:**	**CSE/IT**

Prepared by

Ms. E Uma Shankari, Assistant Professor

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal, Hyderabad - 500 043

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal, Hyderabad - 500 043

Program Outcomes	
PO 1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO 2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO 3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO 4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO 5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO 6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO 7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO 8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO 9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO 10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO 11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO 12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.
Program Specific Outcomes	
PSO 1	**Professional Skills:** The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient design of computer-based systems of varying complexity.
PSO 2	**Problem-Solving Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.
PSO 3	**Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies.

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad – 500043

ATTAINMENT OF PROGRAM OUTCOMES
& PROGRAM SPECIFIC OUTCOMES

S No	Experiment	Program Outcome Attained	Program Specific Outcomes Attained
1	QUICK SORT	PO2, PO3	PSO1
2	MERGE SORT	PO2, PO3	PSO1
3	WARSHALL’S ALGORITHM	PO2	
4	KNAPSACK PROBLEM	PO3	PSO1
5	SHORTEST PATHS ALGORITHM	PO3	PSO1
6	MINIMUM COST SPANNING TREE	PO3	PSO1
7	TREE TRAVESRSALS	PO2, PO3	
8	GRAPH TRAVERSALS	PO2, PO3	
9	SUM OF SUB SETS PROBLEM	PO3	
10	TRAVELLING SALES PERSON PROBLEM	PO3, PO12	PSO1
11	MINIMUM COST SPANNING TREE	PO3	PSO1
12	ALL PAIRS SHORTEST PATHS	PO3	PSO1
13	N QUEENS PROBLEM	PO3	PSO1

DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY

IV Semester: CSE / IT								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
AITB07	Core	L	T	P	C	CIA	SEE	Total
		-	-	3	1.5	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes: 36		

OBJECTIVES:

The course should enable the students to:

I. Learn how to analyze a problem and design the solution for the problem.

II. Design and implement efficient python programming for a specified application.

III. Identify and apply the suitable algorithm for the given real world problem.

COURSE OUTCOMES:

The student will have the ability to:

CO 1: Implement Quick sort, Merge sort and Warshall's algorithm.

CO 2: Implement Dynamic Programming algorithm for the 0/1 Knapsack problem and greedy algorithm for job sequencing with deadlines.

CO 3: Implement Dijkstra's, Prim's, Kruskal's algorithm on spanning tree.

CO 4: Implement Tree Traversal and Graph Traversals techniques using BFS and DFS.

CO 5: Implement Floyd's algorithm for the all pair's shortest path problem and N-queens problem.

COURSE LEARNING OUTCOMES:

1. Understand the basic concepts of python.
2. Understand the different sorting techniques to organize the data in ascending or descending order using quick sort and merge sort.
3. Computing the transitive closure of a given directed graph using Warshall's algorithm.
4. Implementation of dynamic programming for knapsack problem.
5. Identify the shortest paths to other vertices using Dijkstra's algorithm.
6. Analyze the concept of minimum cost spanning trees using Kruskal's algorithm
7. Implementation of tree traversal algorithms for given graphs.
8. Understand graphs and graph traversal techniques like Depth first search and Breadth first search.
9. Understand and implement the sum of subsets problem.
10. Implement the travelling salesperson problem.
11. Analyze the concept of minimum cost spanning trees using Prim's algorithm
12. Implementation of All-Pairs Shortest Paths Problem using Floyd's algorithm and N-Queens problem

LIST OF EXPERIMENTS

WEEK-1	QUICK SORT

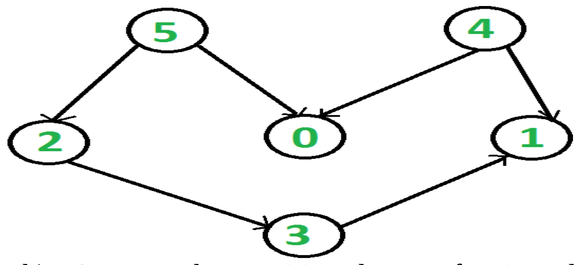
Sort a given set of elements using the quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the 1st to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

WEEK-2	MERGE SORT

Implement merge sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

WEEK-3	WARSHALL’S ALGORITHM

a) Obtain the Topological ordering of vertices in a given digraph.



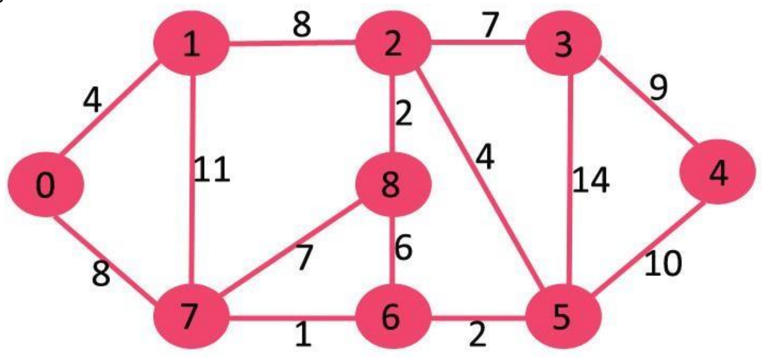
b) Compute the transitive closure of a given directed graph using Warshall's algorithm.

WEEK-4	KNAPSACK PROBLEM

Implement 0/1 Knapsack problem using Dynamic Programming.

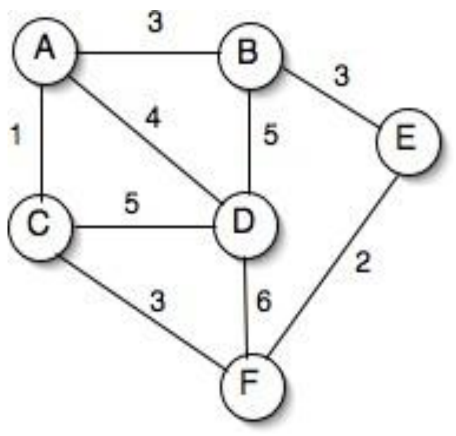
WEEK-5	SHORTEST PATHS ALGORITHM

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.



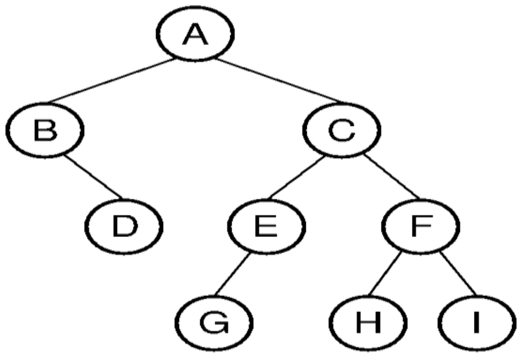
WEEK-6	MINIMUM COST SPANNING TREE

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.



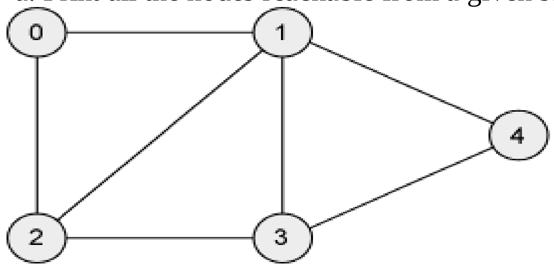
WEEK-7	TREE TRAVESRSALS

Perform various tree traversal algorithms for a given tree.

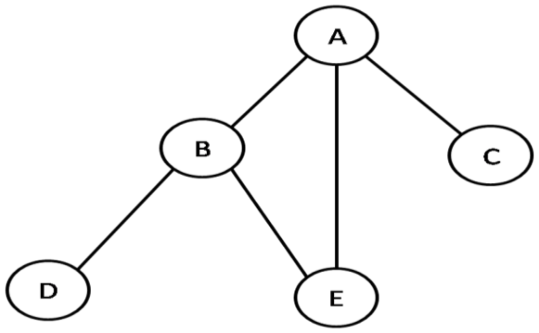


WEEK-8 GRAPH TRAVERSALS

a. Print all the nodes reachable from a given starting node in a digraph using BFS method.



b. Check whether a given graph is connected or not using DFS method.



WEEK-9 SUM OF SUB SETS PROBLEM

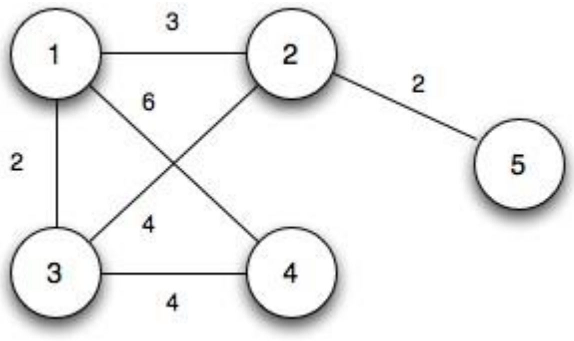
Find a subset of a given set S = {sl, s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1, 2, 6} and {1,8}.A suitable message is to be displayed if the given problem instance doesn't have a solution.

WEEK-10 TRAVELLING SALES PERSON PROBLEM

Implement any scheme to find the optimal solution for the Traveling Sales Person problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation.

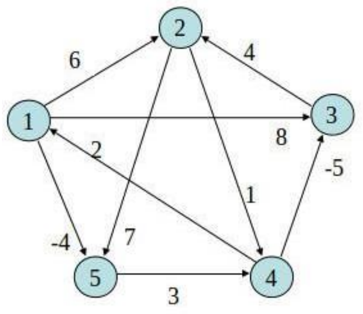
WEEK-11 MINIMUM COST SPANNING TREE

Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.



WEEK-12	**ALL PAIRS SHORTEST PATHS**

Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.



	1	2	3	4	5
1	0	6	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	3	0

WEEK-13	**N QUEENS PROBLEM**

Implement N Queen's problem using Back Tracking.

Reference Books:

1. Levitin A, “Introduction to the Design And Analysis of Algorithms”, Pearson Education, 2008.
2. Goodrich M.T.,R Tomassia, “Algorithm Design foundations Analysis and Internet Examples”, John Wileyn and Sons, 2006.
3. Base Sara, Allen Van Gelder ,“ Computer Algorithms Introduction to Design and Analysis”, Pearson, 3rd Edition, 1999.

Web References:

1. http://www.personal.kent.edu/~rmuhamma/Algorithms/algorithm.html
2. http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=IntroToAlgorithms
3. http://www.facweb.iitkgp.ernet.in/~sourav/daa.html

WEEK-1

QUICK SORT

1.1 OBJECTIVE:

Sort a given set of elements using the Quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

1.2 RESOURCES:

IDLE(python GUI)

1.3 PROGRAM LOGIC:

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

There are many different versions of QuickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in QuickSort is partition. Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

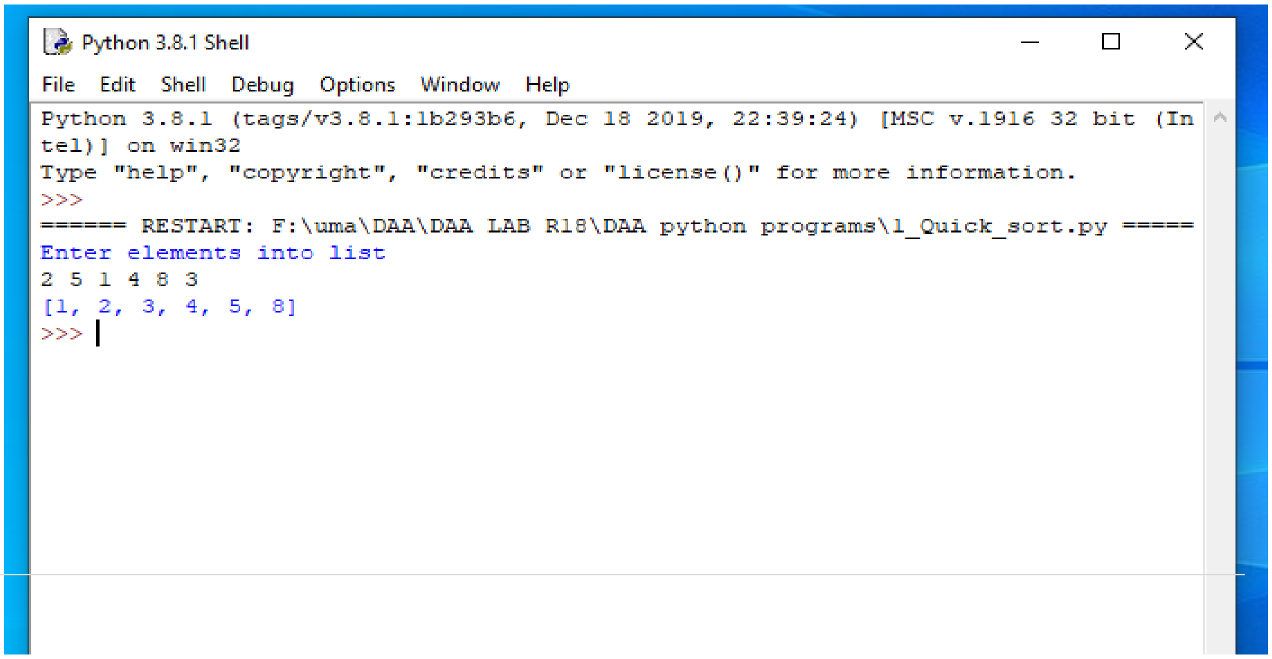
1.4 PROCEDURE:

1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module (F5)

1.5 SOURCE CODE:

```
def q_sort(a,low,high):
    if low<high:
        pivotpos=partition(a,low,high)
        q_sort(a,low,pivotpos-1)
        q_sort(a,pivotpos+1,high)

def partition(a,low,high):
    pivotvalue=a[low]
    up=low+1
    down=high
    done=False
    while not done:
        while up<=down and a[up]<=pivotvalue:
            up+=1
        while down>=up and a[down]>=pivotvalue:
            down-=1
        if down<up:
            done=True
        else:
            temp=a[up]
            a[up]=a[down]
            a[down]=temp
    temp=a[low]
    a[low]=a[down]
    a[down]=temp
    return down

print("Enter elements into list")
a=[int(x) for x in input().split()]
high=len(a)
q_sort(a,0,high-1)
print(a)
```

1.6 INPUT/ OUTPUT



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: F:\uma\DAA\DAA LAB R18\DAA python programs\1_Quick_sort.py =====
Enter elements into list
2 5 1 4 8 3
[1, 2, 3, 4, 5, 8]
>>>
```

1.7 LAB VIVA QUESTIONS:

1. What is the average case time complexity of quick sort.
2. Explain is divide and conquer.
3. Define in place sorting algorithm.
4. List different ways of selecting pivot element.

WEEK-2

MERGE SORT

2.1 OBJECTIVE:

Implement merge sort algorithm to sort a given set of elements and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

2.2 RESOURCES:

IDLE(python GUI)

2.3 PROGRAM LOGIC:

Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves.
The merge() function is used for merging two halves. The merge(a, low, mid, high) is key process that assumes that a[low..mid] and a[mid+1..high] are sorted and merges the two sorted sub-arrays into one.

2.4 PROCEDURE:

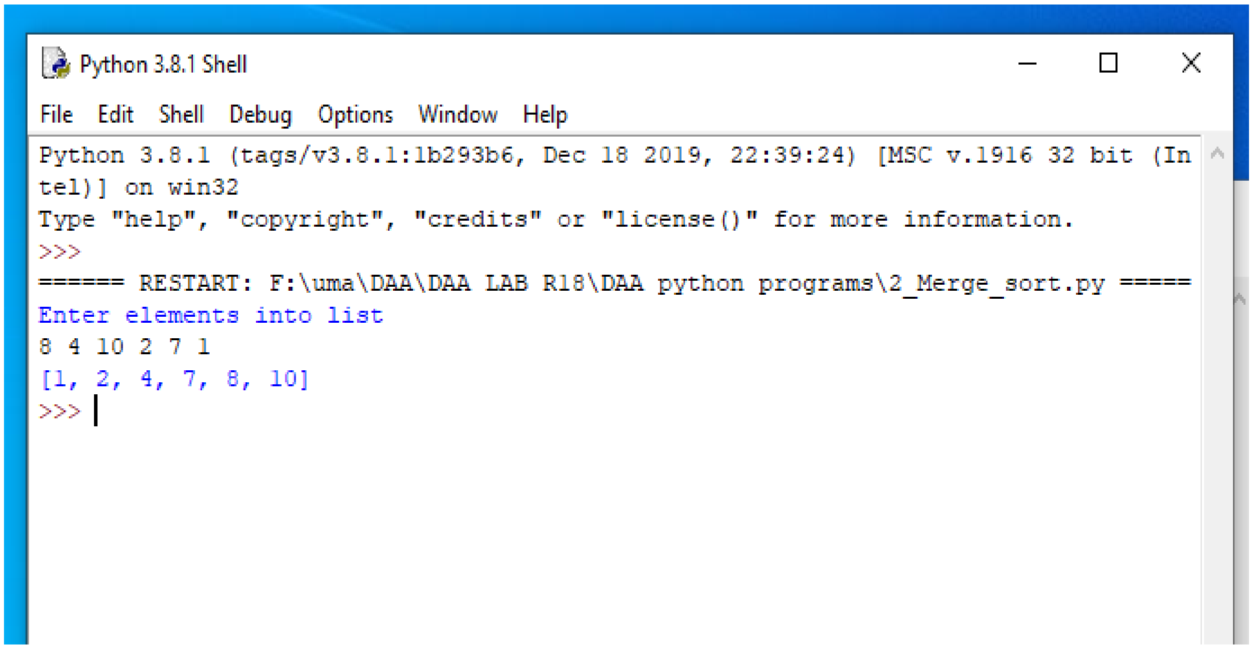
1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module(F5)

1.8 SOURCE CODE:

```
def m_sort(a):
    for i in
range(len(a)):
        if i>1:

mid=len(a)//2

l_half=a[:mid]
r_half=a[mid:]
m_sort(l_half)
m_sort(r_half)
i=j=k=0
while i<len(l_half) and j<len(r_half):
                if l_half[i]<r_half[j]:

a[k]=l_half[i]
                    i+=1
                else:

a[k]=r_half[j]
                    j+=1
                k+=1
```

```
            while
i<len(l_half):

a[k]=l_half[i]
                i+=1
                k+=1
            while
j<len(r_half):

a[k]=r_half[j]
                j+=1
                k+=1

print("Enter
elements into list")
a=[int(x) for x in
input().split()]
m_sort(a)
print(a)
```

1.9 INPUT/ OUTPUT



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: F:\uma\DAA\DAA LAB R18\DAA python programs\2_Merge_sort.py =====
Enter elements into list
8 4 10 2 7 1
[1, 2, 4, 7, 8, 10]
>>>
```

2.7 LAB VIVA QUESTIONS:

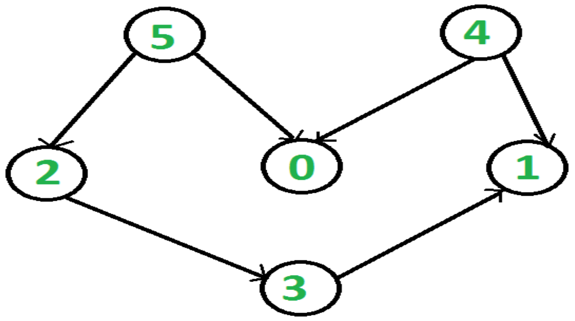
1. What is the running time of merge sort?
2. What technique is used to sort elements in merge sort?
3. Is merge sort in place sorting algorithm?
4. Define stable sort algorithm.

WEEK-3

WARSHALL'S ALGORITHM

3.1 OBJECTIVE:

1. Obtain the Topological ordering of vertices in a given digraph.



2. Compute the transitive closure of a given directed graph using Warshall's algorithm.

3.2 RESOURCES:

IDLE(python GUI)

3.3 PROGRAM LOGIC:

Topological ordering
In topological sorting, a temporary stack is used with the name "s". The node number is not printed immediately; first iteratively call topological sorting for all its adjacent vertices, then push adjacent vertex to stack. Finally, print contents of stack. Note that a vertex is pushed to stack only when all of its adjacent vertices (and their adjacent vertices and so on) are already in stack.
Transitive closure
Given a directed graph, find out if a vertex j is reachable from another vertex i for all vertex pairs (i, j) in the given graph. Here reachable mean that there is a path from vertex i to j. The reach-ability matrix is called transitive closure of a graph.

3.4 PROCEDURE:

1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module(F5)

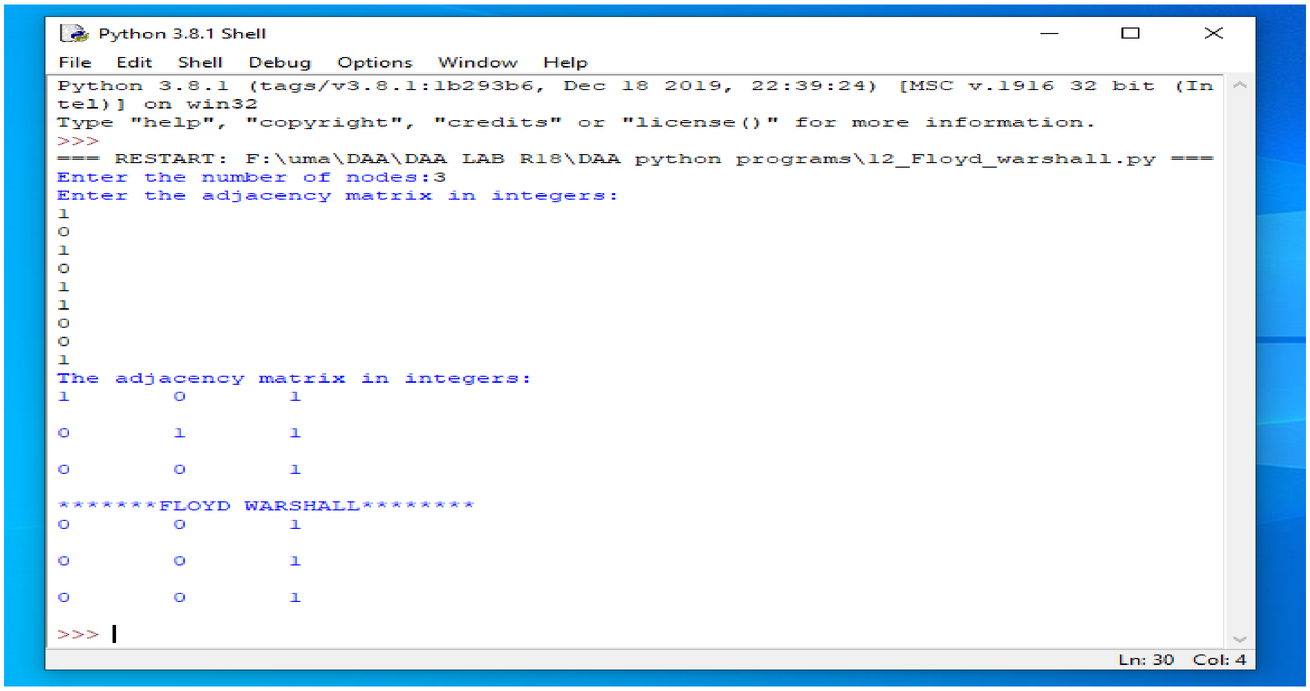
3.5 SOURCE CODE:

// Topological ordering

```
n=int(input("Enter the number of nodes:"))
cost=[[0 for i in range(n)] for j in range(n)]
print("Enter the adjacency matrix in integers:")
for i in range(n):
    for j in range(n):
        temp=int(input())
        cost[i][j]=temp
print("The adjacency matrix in integers:")
for i in range(n):
        for j in range(n):
            if cost[i][j]==999:
                print("INF",end="\t")
            else:
                print(cost[i][j],end="\t")
        print("\n")
    for k in range(n):
        for i in range(n):
            for j in range(n):
                cost[i][j]=min(cost[i][j],cost[i][k]+cost[k][j])
    print("*******FLOYD WARSHALL********")
    for i in range(n):
        for j in range(n):
            if cost[i][j]==999:
                print("INF",end="\t")
            else:
                print(cost[i][j],end="\t")
        print("\n")
```

3.6 INPUT/ OUTPUT

Topological ordering



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
=== RESTART: F:\uma\DAA\DAA LAB R18\DAA python programs\12_Floyd_warshall.py ===
Enter the number of nodes:3
Enter the adjacency matrix in integers:
1
0
1
0
1
1
0
0
1
The adjacency matrix in integers:
1        0        1

0        1        1

0        0        1

*******FLOYD WARSHALL********
0        0        1

0        0        1

0        0        1

>>> 
```

3.6 LAB VIVA QUESTIONS:

1. Define transitive closure.
2. Define topological sequence.
3. What is the time complexity of Warshall's algorithm?

WEEK-4

KNAPSACK PROBLEM

4.1 OBJECTIVE:

Implement 0/1 Knapsack problem using Dynamic Programming.

4.2 RESOURCES:

IDLE(python GUI)

4.3 PROGRAM LOGIC:

Given some items, pack the knapsack to get the maximum total profit. Each item has some Weight and some profit. Total weight that we can carry is no more than some fixed number W.

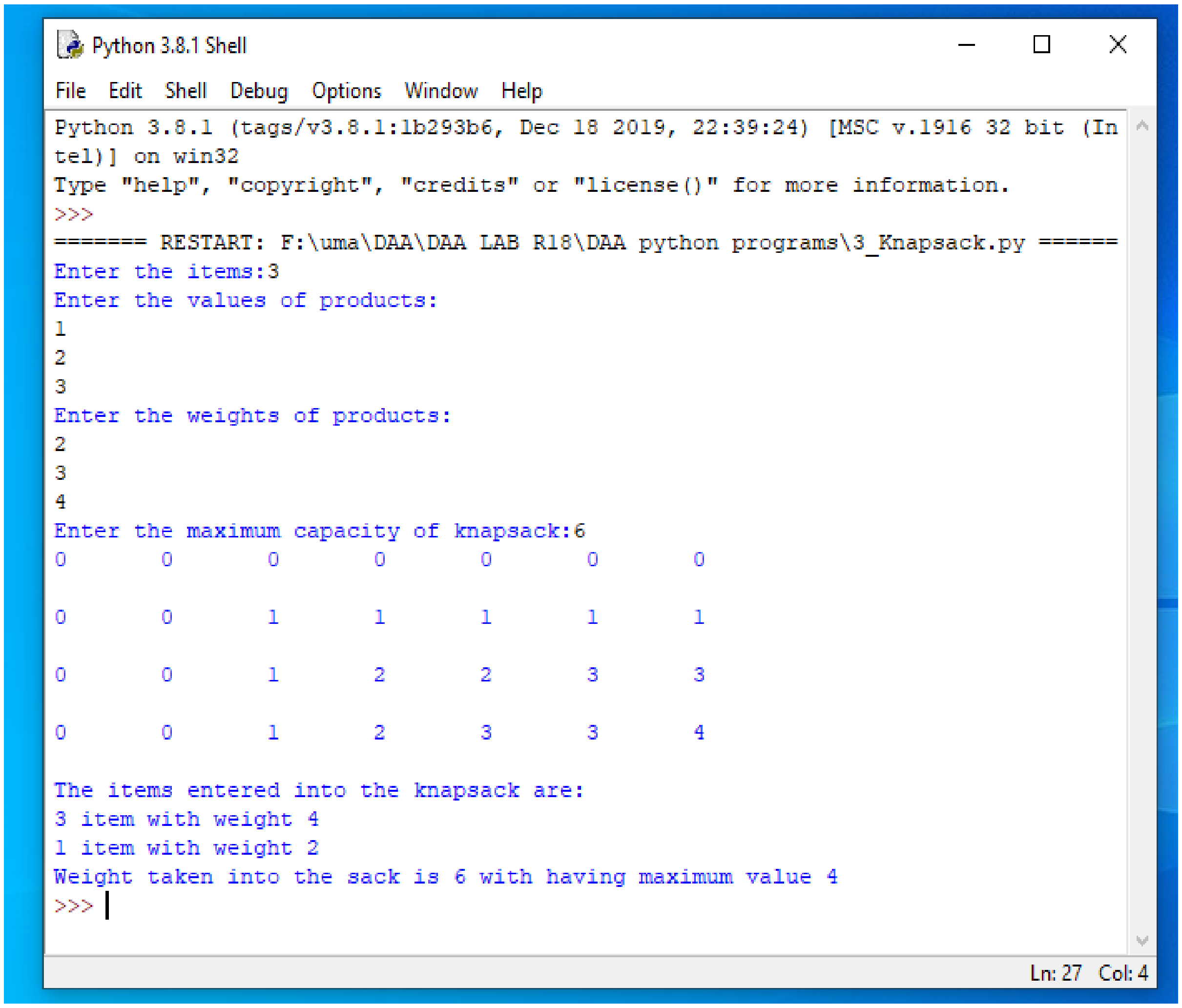
4.4 PROCEDURE:

1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module(F5)

4.5 SOURCE CODE:

```
def show(W,n,wt,V):
    i=n
    w=W
    print("The items entered into the knapsack are:")
    while i>0 and w>0:
        if V[i][w]!=V[i-1][w]:
            print(i,"item with weight",wt[i])
            w=w-wt[i]
            i=i-1
        else:
            i=i-1
    print("Weight taken into the sack is",W-w,"with having maximum value",V[n][W])
def knapsack(W,n,val,wt):
    V=[[0 for j in range(W+2)] for i in range(n+2)]
    for i in range(n+1):
        for w in range(W+1):
            if wt[i]<=w:
                V[i][w]=max(V[i-1][w],(val[i]+V[i-1][w-wt[i]]))
            else:
                V[i][w]=V[i-1][w]
            print(V[i][w],end="\t")
        print("\n")
    show(W,n,wt,V)
n=int(input("Enter the items:"))
val=[0 for i in range(0,n+1,1)]
wt=[0 for i in range(0,n+1,1)]
print("Enter the values of products:")
for i in range(1,n+1,1):
    temp=int(input())
    val[i]=temp
print("Enter the weights of products:")
for i in range(1,n+1,1):
    temp=int(input())
    wt[i]=temp
W=int(input("Enter the maximum capacity of knapsack:"))
knapsack(W,n,val,wt)
```

4.6 INPUT/ OUTPUT



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: F:\uma\DAA\DAA LAB R18\DAA python programs\3_Knapsack.py ======
Enter the items:3
Enter the values of products:
1
2
3
Enter the weights of products:
2
3
4
Enter the maximum capacity of knapsack:6
0       0       0       0       0       0       0

0       0       1       1       1       1       1

0       0       1       2       2       3       3

0       0       1       2       3       3       4

The items entered into the knapsack are:
3 item with weight 4
1 item with weight 2
Weight taken into the sack is 6 with having maximum value 4
>>>
```

4.7 LAB VIVA QUESTIONS:

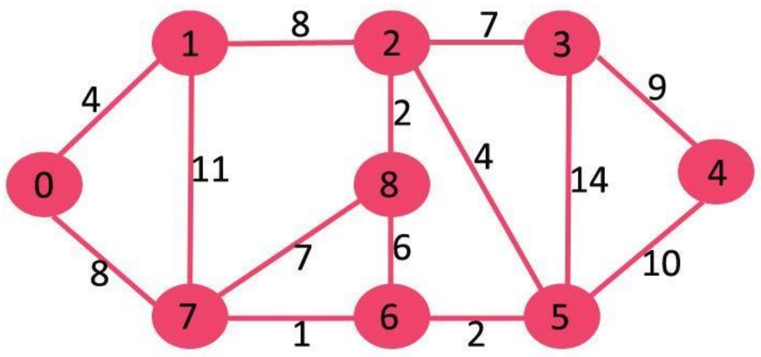
1. Define knapsack problem.
2. Define principle of optimality.
3. What is the optimal solution for knapsack problem?
4. What is the time complexity of knapsack problem?

WEEK-5

SHORTEST PATHS ALGORITHM

5.1 OBJECTIVE:

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.



5.2 RESOURCES:

IDLE(python GUI)

5.3 PROGRAM LOGIC:

1) Create a set S that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE.Assign distance value as 0 for the source vertex so that it is picked first.
3) While *S* doesn't include all vertices
a) Pick a vertex u which is not there in *S* and has minimum distance value.
b)Include u to *S*.
c) Update distance value of all adjacent vertices of u.

To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

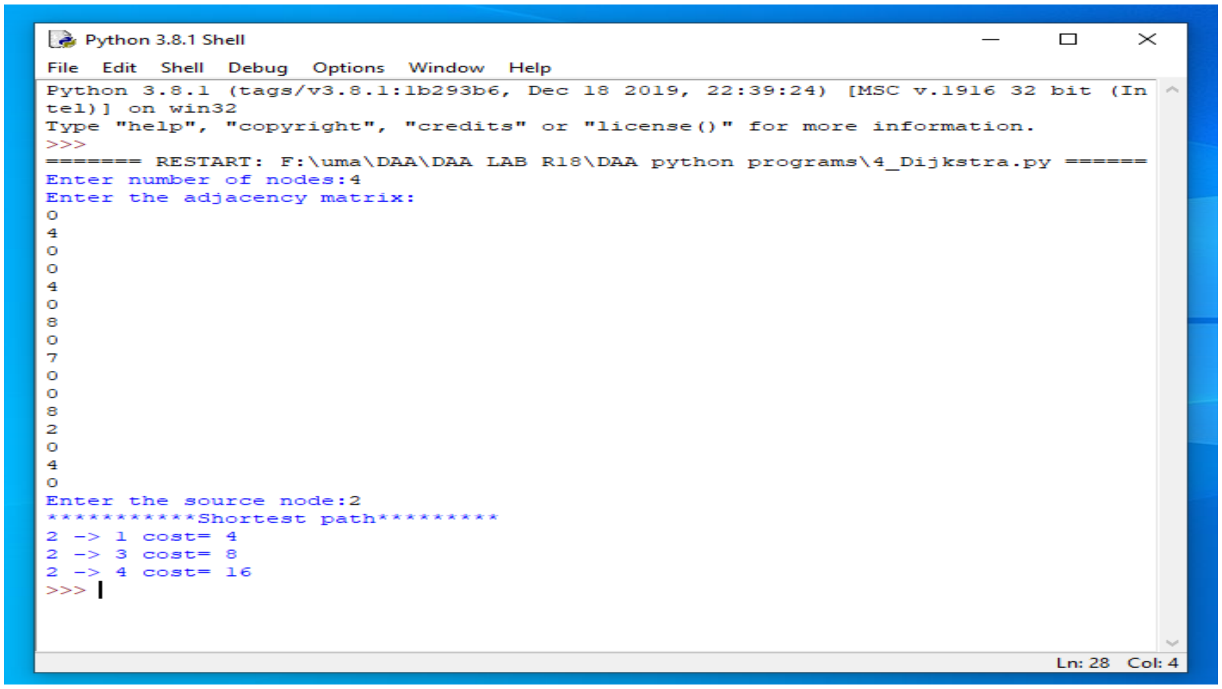
5.4 PROCEDURE:

1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module(F5)

5.5 SOURCE CODE:

```
def dij(n,v,cost,dist):
    flag=[0 for x in range(1,n+2,1)]
    for i in range(1,n+1,1):
        flag[i]=0
        dist[i]=cost[v][i]
    count=2
    while count<=n:
        mini=99
        for w in range(1,n+1,1):
            if (dist[w] < mini and not(flag[w])):
                mini=dist[w]
                u=w
        flag[u]=1
        count=count+1
        for w in range(1,n+1,1):
            if ((dist[u]+cost[u][w]<dist[w]) and not(flag[w])):
                dist[w]=dist[u]+cost[u][w]
n=int(input("Enter number of nodes:"))
cost=[[0 for j in range(1,n+2,1)] for i in range(1,n+2,1)]
print("Enter the adjacency matrix:")
for i in range(1,n+1,1):
    for j in range(1,n+1,1):
        temp=int(input())
        cost[i][j]=temp
        if cost[i][j]==0:
            cost[i][j]=999
v=int(input("Enter the source node:"))
dist=[0 for i in range(1,n+2,1)]
dij(n,v,cost,dist)
print("***********Shortest path*********")
for i in range(1,n+1,1):
    if(i!=v):
        print(v,"->",i,"cost=",dist[i])
```

5.6 INPUT/ OUTPUT



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: F:\uma\DAA\DAA LAB R18\DAA python programs\4_Dijkstra.py ======
Enter number of nodes:4
Enter the adjacency matrix:
0
4
0
0
4
0
8
0
7
0
0
8
2
0
4
0
Enter the source node:2
***********Shortest path*********
2 -> 1 cost= 4
2 -> 3 cost= 8
2 -> 4 cost= 16
>>> 
```

5.7 LAB VIVA QUESTIONS:

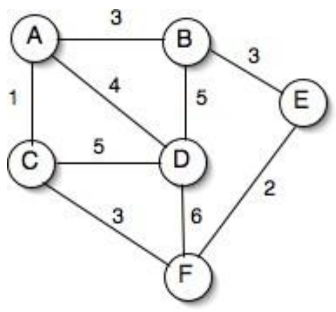
1. What is the time complexity of Dijkstra’s algorithm?
2. Define cost matrix.
3. Define directed graph.
4. Define connected graph.

WEEK-6

MINIMUM COST SPANNING TREE

6.1 OBJECTIVE:

Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.



6.2 RESOURCES:

IDLE(python GUI)

6.3 PROGRAM LOGIC:

1. Sort all the edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat step#2 until there are (V-1) edges in the spanning tree.

6.4 PROCEDURE:

1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

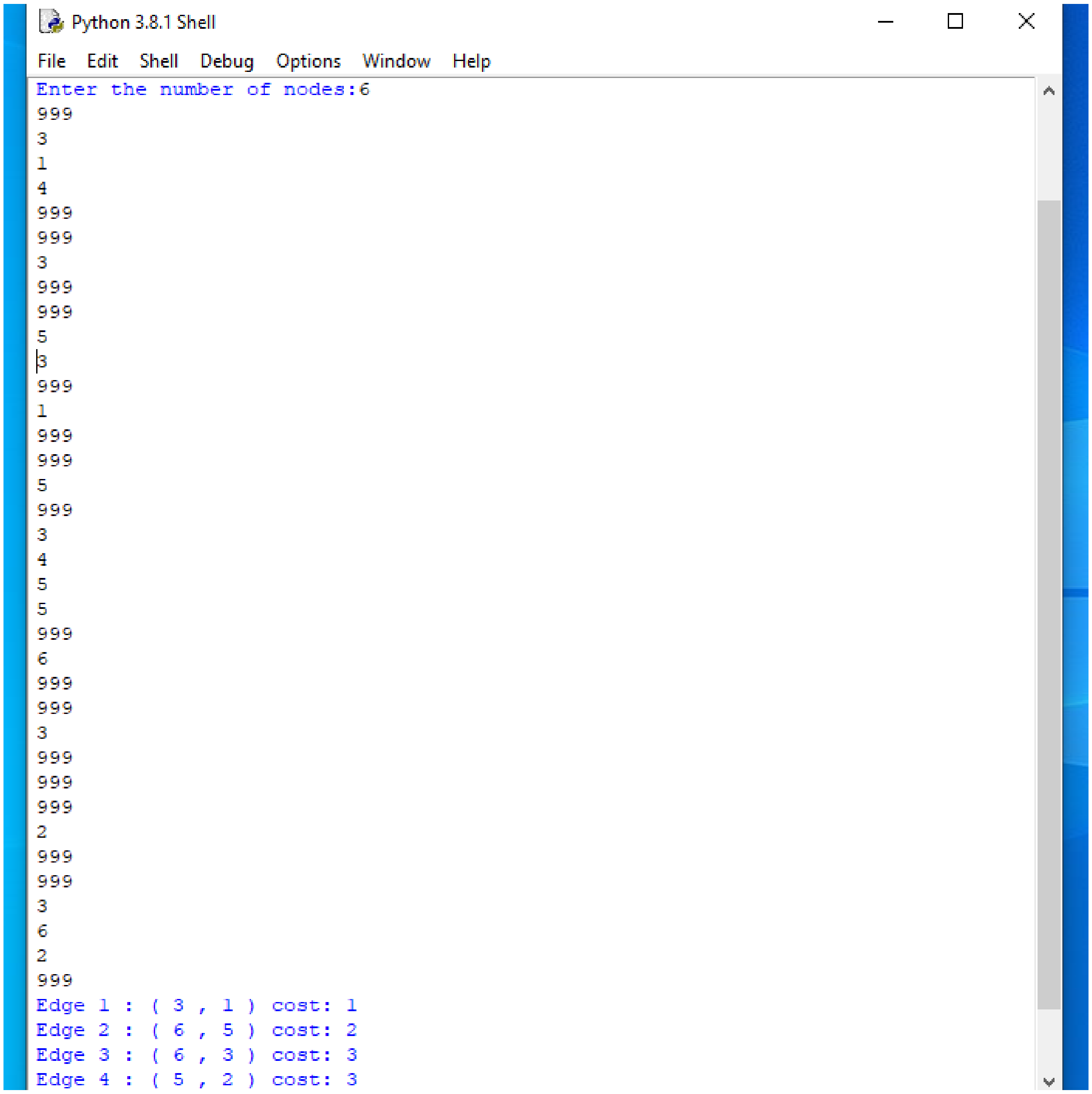
6.5 SOURCE CODE:

```
maxi=9999999
n=int(input("Enter the number of nodes:"))
parent=[0 for i in range(n)]
def find(i):
    while parent[i]!=i:
        i=parent[i]
    return i
def uni(i,j):
    x=find(i)
    y=find(j)
    parent[x]=y
def main(n,cost):
    mincost=0
    for i in range(n):
        parent[i]=i
    e_ctr=0
    while e_ctr<n-1:
        mini=maxi
        a=0
        b=0
        for i in range(n):
            for j in range(n):
                if (cost[i][j]<=mini and find(i)!=find(j)):
                    mini=cost[i][j]
                    a=i
                    b=j
        uni(a,b)
        print("Edge",e_ctr+1,": (",a+1,",",b+1,") cost:",mini)
        e_ctr+=1
        mincost=mincost+mini
    print("Minimum cost=",mincost)
cost=[[0 for i in range(n)] for j in range(n)]
for i in range(n):
    for j in range(n):
        temp=int(input())
        cost[i][j]=temp
        if cost[i][j]==0:
            cost[i][j]=maxi
main(n,cost)
"""
0
2
0
6
0
2
0
3
8
5
0
3
0
0
7
6
8
0
0
9
0
5
7
9
0"""
```

```
"""
0
28
0
0
0
10
0
28
0
16
0
0
0
14
0
16
0
12
0
0
0
0
0
12
0
22
0
18
0
0
0
22
0
25
24
10
0
0
0
25
0
0
0
14
0
18
```

```
24
0
0"""
```

6.6 INPUT/ OUTPUT



```
Enter the number of nodes:6
999
3
1
4
999
999
3
999
999
5
3
999
1
999
999
5
999
3
4
5
5
999
6
999
999
3
999
999
999
2
999
999
3
6
2
999
Edge 1 : ( 3 , 1 ) cost: 1
Edge 2 : ( 6 , 5 ) cost: 2
Edge 3 : ( 6 , 3 ) cost: 3
Edge 4 : ( 5 , 2 ) cost: 3
```

6.7 LAB VIVA QUESTIONS:

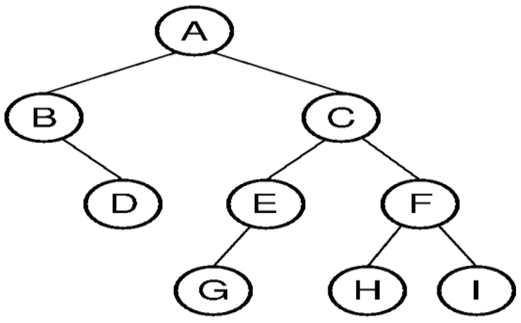
1. What is the time complexity of Kruskal's algorithm.
2. Define spanning tree.
3. Define minimum cost spanning tree.

WEEK-7

TREE TRAVESRSALS

7.1 OBJECTIVE:

Perform various tree traversal algorithms for a given tree.



7.2 RESOURCES:

IDLE(python GUI)

7.3 PROGRAM LOGIC:

Traversal is a process to visit all the nodes of a tree and may print their values too.

```
Inorder(tree)
  1. Traverse the left subtree, i.e., call Inorder(left-subtree)
  2. Visit the root.
  3. Traverse the right subtree, i.e., call Inorder(right-subtree)

Postorder(tree)
  1. Traverse the left subtree, i.e., call Postorder(left-subtree)
  2. Traverse the right subtree, i.e., call Postorder(right-subtree)
  3. Visit the root.

Preorder(tree)
  1. Visit the root.
  2. Traverse the left subtree, i.e., call Preorder(left-subtree)
  3. Traverse the right subtree, i.e., call Preorder(right-subtree)
```

7.4 PROCEDURE:

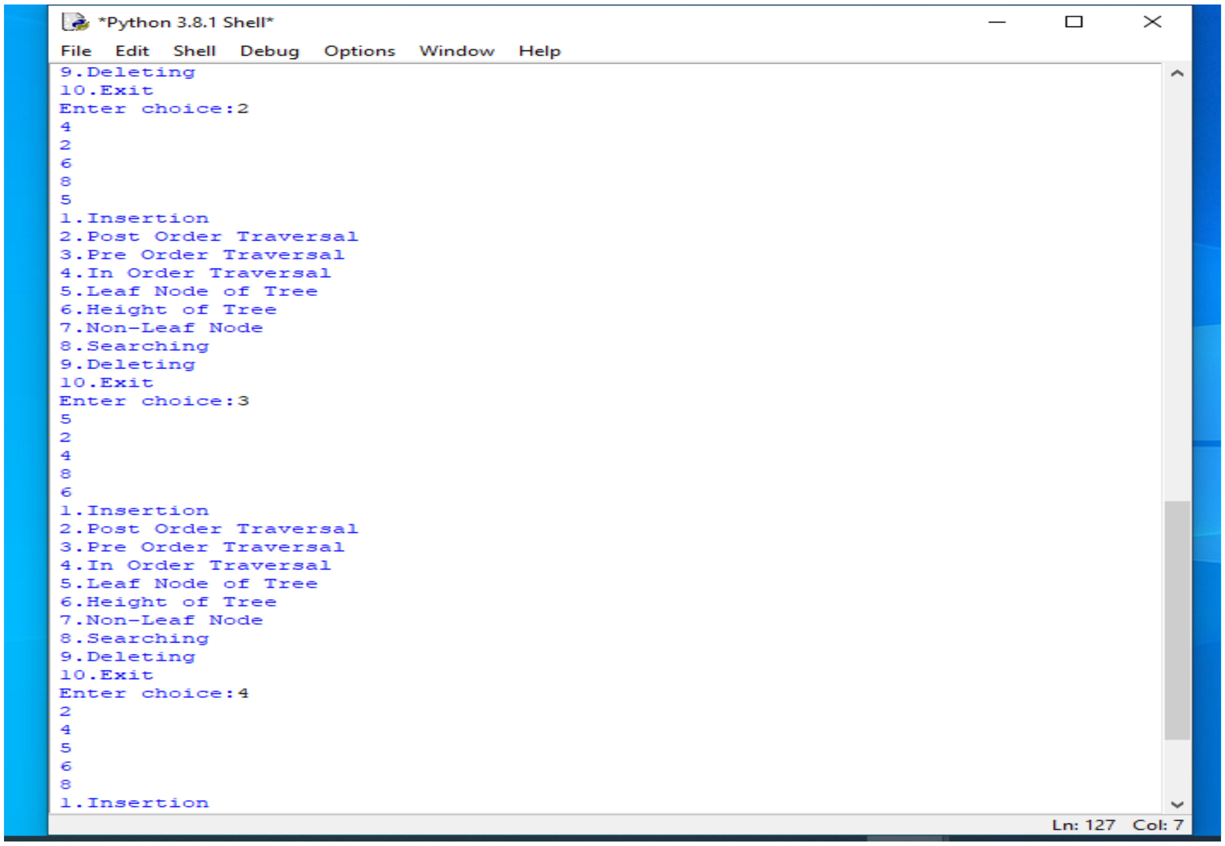
1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module(F5)

7.5 SOURCE CODE:

```
class Node:
    def __init__(self,data):
        self.data=data
        self.l=None
        self.r=None
class BST:
    def __init__(self):
        self.root=None
    def insert(self,data):
        ctr=0
        node=Node(data)
        if self.root==None:
            self.root=node
        else:
            temp=self.root
            while True:
                if data<temp.data and temp.l is None:
                    temp.l=node
                    break
                if data>temp.data and temp.r is None:
                    temp.r=node
                    break
                if data<temp.data and temp.l!=None:
                    temp=temp.l
                if data>temp.data and temp.r!=None:
                    temp=temp.r
            print("Node inserted")
            ctr+=1
    def searching(self,root,n):
        if n==root.data:
            print("Element found",root.data)
            return
        elif n>root.data:
            if self.root.r is not None:
                self.searching(root.r,n)
            else:
                return "No match found"
        else:
            if self.root.l is not None:
                self.searching(root.l,n)
            else:
                return "Element not found"
    def postorder(self,root):
        if root:
            self.postorder(root.l)
            self.postorder(root.r)
            print(root.data)
    def preorder(self,root):
        if root:
            print(root.data)
            self.preorder(root.l)
            self.preorder(root.r)
    def inorder(self,root):
        if root:
            self.inorder(root.l)
            print(root.data)
            self.inorder(root.r)
    def leafnode(self,root):
        if root:
            self.leafnode(root.l)
            self.leafnode(root.r)
            if root.l == None and root.r==None:
                print(root.data)
    def height(self,root):
        if root==None:
            return -1
        else:
            return 1+max(self.height(root.l),self.height(root.r))
    def nleafnode(self,root):
        if root:
            self.nleafnode(root.l)
            self.nleafnode(root.r)
            if root.l != None and root.r != None:
                print(root.data)

    def minValueNode(node):
        current = node
        while(current.left is not None):
            current = current.l
        return current
    def deleteNode(self,root,key):
        if root is None:
            return root
        if key < root.key:
            root.l = self.deleteNode(root.l,key)
        elif(key > root.key):
            root.right =self.deleteNode(root.r,key)
        else:
            if root.l is None :
                temp = root.r
                root = None
                return temp
            elif root.r is None :
                temp = root.l
                root = None
                return temp
            temp = minValueNode(root.r)
            root.key = temp.key
            root.r =self.deleteNode(root.r,temp.key)
        return root

b=BST()
print("******************TREE USING DOUBLE LINKED LIST*************************")
while True:
    print("1.Insertion \n2.Post Order Traversal\n3.Pre Order Traversal")
    print("4.In Order Traversal\n5.Leaf Node of Tree\n6.Height of Tree\n7.Non-Leaf Node")
    print("8.Searching\n9.Deleting\n10.Exit")
    ch=int(input("Enter choice:"))
```

```
if ch==1:
      data=eval(input("Enter data:"))
      b.insert(data)
elif ch==2:
    b.postorder(b.root)
elif ch==3:
    b.preorder(b.root)
elif ch==4:
    b.inorder(b.root)
elif ch==5:
    b.leafnode(b.root)
elif ch==6:
    print(b.height(b.root))
elif ch==7:
    b.nleafnode(b.root)
elif ch==8:
    n=eval(input("Enter search element:"))
    b.searching(b.root,n)
elif ch==9:
    data=eval(input("Enter delete element:"))
    b.deleteNode(b.root,data)
    print(data,"Deleted")
else:
    print("Exit")
    break
```

7.6 INPUT/ OUTPUT



```
9.Deleting
10.Exit
Enter choice:2
4
2
6
8
5
1.Insertion
2.Post Order Traversal
3.Pre Order Traversal
4.In Order Traversal
5.Leaf Node of Tree
6.Height of Tree
7.Non-Leaf Node
8.Searching
9.Deleting
10.Exit
Enter choice:3
5
2
4
8
6
1.Insertion
2.Post Order Traversal
3.Pre Order Traversal
4.In Order Traversal
5.Leaf Node of Tree
6.Height of Tree
7.Non-Leaf Node
8.Searching
9.Deleting
10.Exit
Enter choice:4
2
4
5
6
8
1.Insertion
```

7.7 LAB VIVA QUESTIONS:

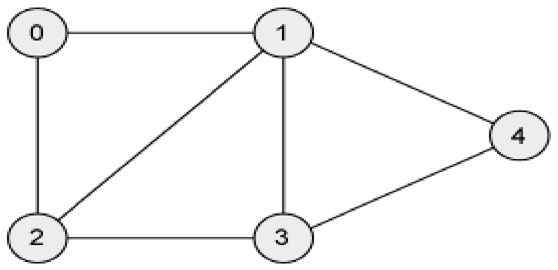
1. Define binary tree.
2. List different tree traversals.
3. Explain inorder travels with example.
4. Explain preorder travels with example.
5. Explain postorder travels with example.

WEEK-8

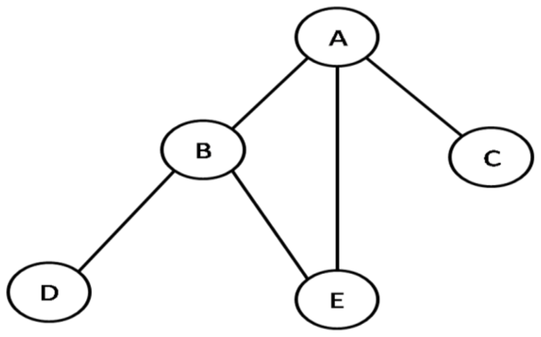
GRAPH TRAVERSALS

8.1 OBJECTIVE:

1. Print all the nodes reachable from a given starting node in a digraph using BFS method.



2. Check whether a given graph is connected or not using DFS method.



8.2 RESOURCES:

IDLE(python GUI)

8.3 PROGRAM LOGIC:

Breadth first traversal

Breadth First Search (BFS) algorithm traverses a graph in a breadth ward motion and uses a queue to remember to get the next vertex to start a search.

1. Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
2. If no adjacent vertex is found, remove the first vertex from the queue.
3. Repeat Rule 1 and Rule 2 until the queue is empty.

Depth first traversal

Depth First Search (DFS) algorithm traverses a graph in a depth ward motion and uses a stack to remember to get the next vertex to start a search.

1. Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.
2. If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)
3. Repeat Rule 1 and Rule 2 until the stack is empty.

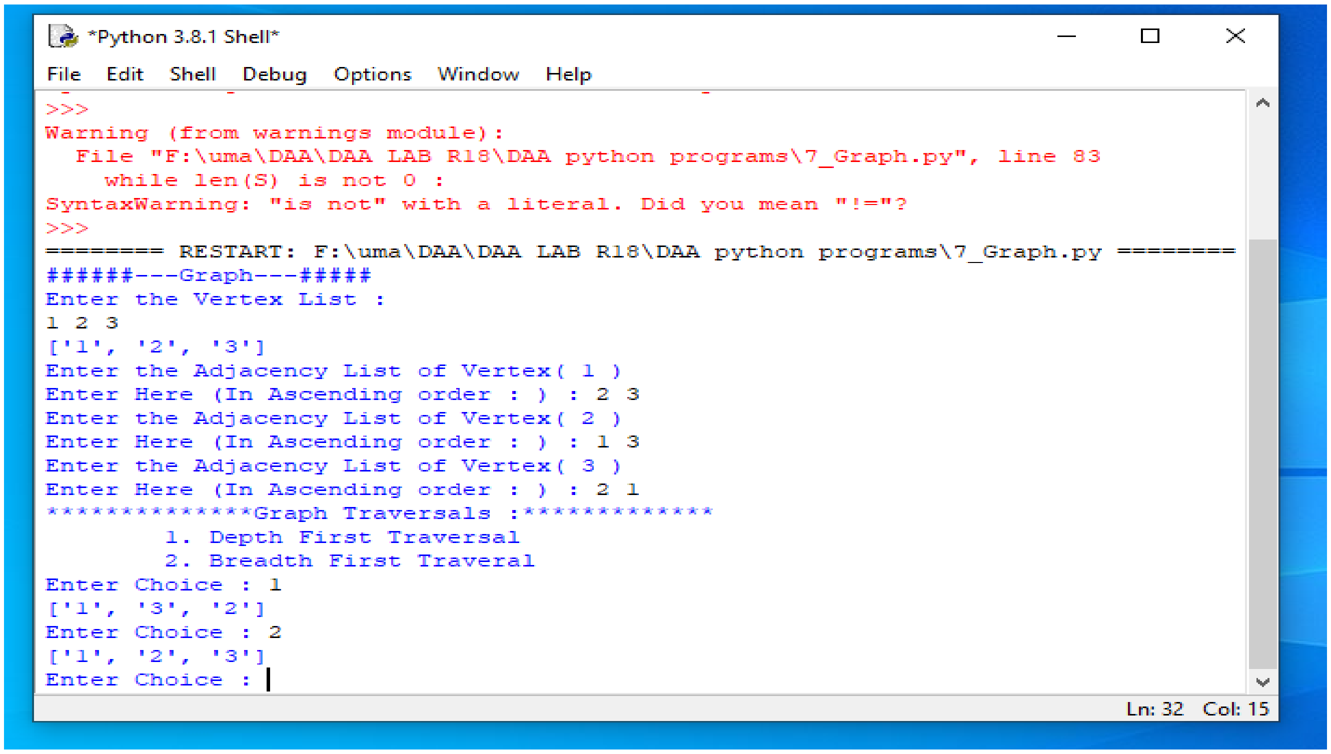
8.4 PROCEDURE:

1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module(F5)

8.5 SOURCE CODE:

```
class Vertex:
    def __init__(self, data) :
        self.data = data
        self.nxtVertex = None
        self.AdjHead = None
        self.trvState = None

class AdjNode :
    def __init__(self) :
        self.reference = None
        self.nxt = None

class Graph :
    def __init__(self) :
        self.root = None
        self.vertexCtr = 0

    def create(self) :


        print('Enter the Vertex List : ')
        vertexLst = input().split()
        print(vertexLst)
        for i in range(len(vertexLst)) :
            new_node = Vertex(vertexLst[i])
            if self.root == None :
                self.root = new_node
            else :
                temp = self.root
                while temp.nxtVertex is not None :
                    temp = temp.nxtVertex
                temp.nxtVertex = new_node

        # Creation of AdjacetList
        temp = self.root
        while temp is not None :
            print('Enter the Adjacency List of Vertex(',temp.data,')')
            adjLst = input('Enter Here (In Ascending order : ) : ').split()
            for i in range(len(adjLst)) :
                refPtr = self.getReference(adjLst[i])
                new_node = AdjNode()
                new_node.reference = refPtr
                if temp.AdjHead == None :
                    temp.AdjHead = new_node
                else :
                    temp1 = temp.AdjHead
                    while temp1.nxt is not None :
                        temp1 = temp1.nxt
                    temp1.nxt = new_node
            temp = temp.nxtVertex

    def getReference(self, nodeData) :
        temp = self.root
        while temp is not None :
            if temp.data == nodeData :

                return temp
            temp = temp.nxtVertex
    def LinearSearch(self, Lst, Key) :
        for i in range(len(Lst)) :
            if Lst[i] == Key :
                return True
        return False
    def DFT(self) :
        Path = []
        S = []
        V = self.root
        S.append(V)
        while len(S) is not 0 :
            V = S.pop()
            if  self.LinearSearch(Path, V.data) == False :
                Path.append(V.data)
                temp = V.AdjHead
                while temp is not None :
                    S.append(temp.reference)
                    temp = temp.nxt
        return Path
    def BFT(self) :
        Path = []
        S = []
        V = self.root
        S.append(V)
        while len(S) is not 0 :
            V = S.pop(0)
            if  self.LinearSearch(Path, V.data) == False :
                Path.append(V.data)
                temp = V.AdjHead
                while temp is not None :
                    S.append(temp.reference)
                    temp = temp.nxt
        return Path

    #Driver Code

print("######---Graph---#####")
G = Graph()
G.create()
print('***************Graph Traversals :*************')
print('\t1. Depth First Traversal')
print('\t2. Breadth First Traveral')

while True :
    ch = int(input('Enter Choice : '))
    if ch == 1 :
        print(G.DFT())
    elif ch == 2 :
        print(G.BFT())
    else:
        print("Exit")
        break
```

8.6 INPUT/ OUTPUT



```
>>>
Warning (from warnings module):
  File "F:\uma\DAA\DAA LAB R18\DAA python programs\7_Graph.py", line 83
    while len(S) is not 0 :
SyntaxWarning: "is not" with a literal. Did you mean "!="?
>>>
======== RESTART: F:\uma\DAA\DAA LAB R18\DAA python programs\7_Graph.py ========
######---Graph---#####
Enter the Vertex List :
1 2 3
['1', '2', '3']
Enter the Adjacency List of Vertex( 1 )
Enter Here (In Ascending order : ) : 2 3
Enter the Adjacency List of Vertex( 2 )
Enter Here (In Ascending order : ) : 1 3
Enter the Adjacency List of Vertex( 3 )
Enter Here (In Ascending order : ) : 2 1
**************Graph Traversals :*************
        1. Depth First Traversal
        2. Breadth First Traveral
Enter Choice : 1
['1', '3', '2']
Enter Choice : 2
['1', '2', '3']
Enter Choice :
```

8.7 LAB VIVA QUESTIONS:

1. Define graph, connected graph.
2. List the different graph traversals.
3. Explain DFS traversal.
4. Explain BFS traversal.
5. What are the time complexities of BFS and DFS algorithms?

WEEK-9

SUM OF SUB SETS PROBLEM

9.1 OBJECTIVE:

Find a subset of a given set S = {sl, s2.....sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S= {1, 2, 5, 6, 8} and d = 9 there are two solutions {1, 2, 6} and {1, 8}.A suitable message is to be displayed if the given problem instance doesn't have a solution.

9.2 RESOURCES:

IDLE(python GUI)

9.3 PROGRAM LOGIC:

Given a set of non-negative integers, and a value *sum*, determine if there is a subset of the given set with sum equal to given *sum*.

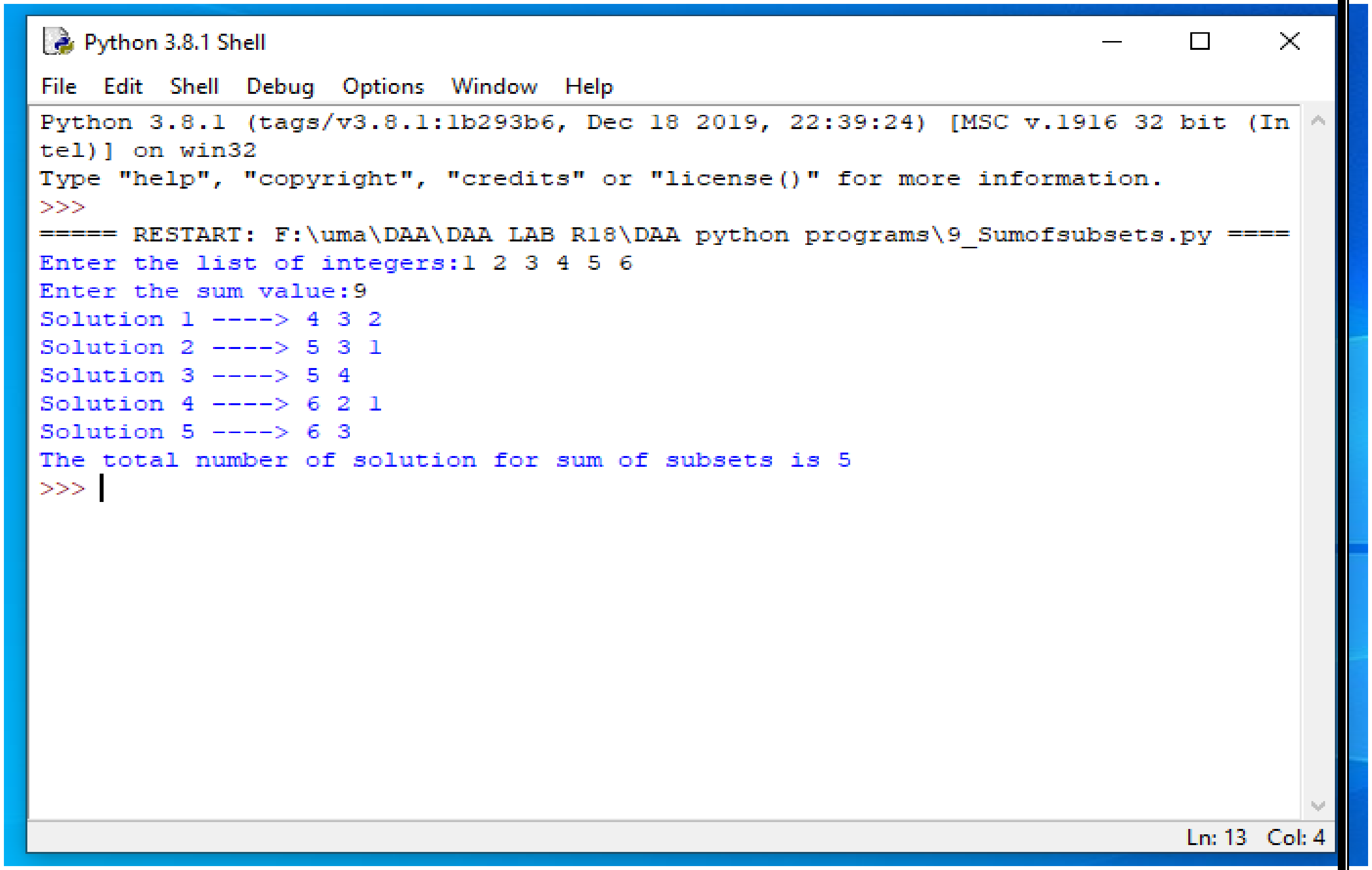
9.4 PROCEDURE:

1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module(F5)

9.5 SOURCE CODE:

```
b=[]
def solution(a,n,subset,s):
    global ctr
    ctr=0
    if s==0:
        b.append(subset)
        return
    if n==0:
        return
    if a[n-1]<=s:
        solution(a,n-1,subset,s)
        solution(a,n-1,subset+str(a[n-1])+" ",s-a[n-1])
    else:
        solution(a,n-1,subset,s)
print("Enter the list of integers",end=":")
a=[int(x) for x in input().split()]
s=int(input("Enter the sum value:"))
subset=""
solution(a,len(a),subset,s)
ctr=1
for i in b:
    print("Solution",ctr,"---->",i)
    ctr=ctr+1
print("The total number of solution for sum of subsets is",len(b))
```

9.6 INPUT/ OUTPUT



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: F:\uma\DAA\DAA LAB R18\DAA python programs\9_Sumofsubsets.py ====
Enter the list of integers:1 2 3 4 5 6
Enter the sum value:9
Solution 1 ----> 4 3 2
Solution 2 ----> 5 3 1
Solution 3 ----> 5 4
Solution 4 ----> 6 2 1
Solution 5 ----> 6 3
The total number of solution for sum of subsets is 5
>>> 
```

9.7 LAB VIVA QUESTIONS:

1. Define is Back-Tracking.
2. Explain Sum of subset problem.
3. What is time complexity of sum of subset problem?

WEEK-10

TRAVELLING SALES PERSON PROBLEM

10.1 OBJECTIVE:

Implement any scheme to find the optimal solution for the Traveling Sales Person problem and then solve the same problem instance using any approximation algorithm and determine the error in the approximation

10.2 RESOURCES:

IDLE(python GUI)

10.3 PROGRAM LOGIC:

1. Check for the disconnection between the current city and the next city
2. Check whether the travelling sales person has visited all the cities
3. Find the next city to be visited
4. Find the solution and terminate

10.4 PROCEDURE:

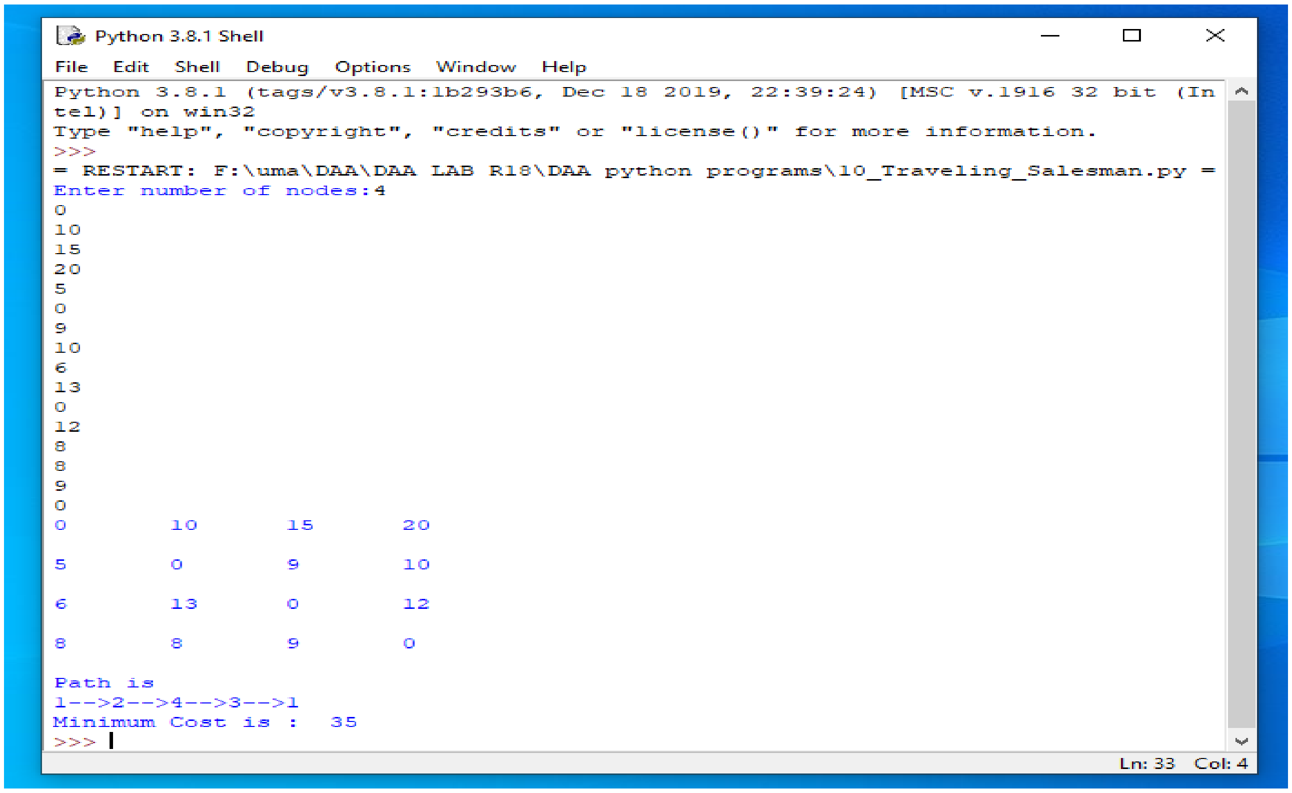
1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module(F5)

10.5 SOURCE CODE:

```
n=int(input("Enter number of nodes:"))
cost=0
def main():
    global a,visit
    a=[[0 for i in range(n)]
       for j in range(n)]
    visit=[0 for i in range(n)]
    for i in range(n):
        for j in range(n):
            temp=int(input())
            a[i][j]=temp
    for i in range(n):
        for j in range(n):
            print(a[i][j],end="\t")
        print("\n")
    print("Path is")
    mincost(0)
def mincost(c):
    global cost
    visit[c]=1
    print(c+1,end="-->")
    x=least(c)
```

```
        if x==999:
            x=0
            print(x+1)
            cost=cost+a[c][x]
            return
        mincost(x)
    def least(u):
        global cost
        mini=v=999
        for i in range(n):
            if a[u][i]!=0 and visit[i]==0:
                if a[u][i]+a[i][u]<mini:
                    mini=a[i][u]+a[u][i]
                    kmin=a[u][i]
                    v=i
        if mini!=999:
            cost=cost+kmin
        return v
    main()
    print("Minimum Cost is : ",cost)
```

10.6 INPUT/ OUTPUT



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: F:\uma\DAA\DAA LAB R18\DAA python programs\10_Traveling_Salesman.py =
Enter number of nodes:4
0
10
15
20
5
0
9
10
6
13
0
12
8
8
9
0
0       10      15      20

5       0       9       10

6       13      0       12

8       8       9       0

Path is
1-->2-->4-->3-->1
Minimum Cost is :  35
>>> 
```

10.7 LAB VIVA QUESTIONS:

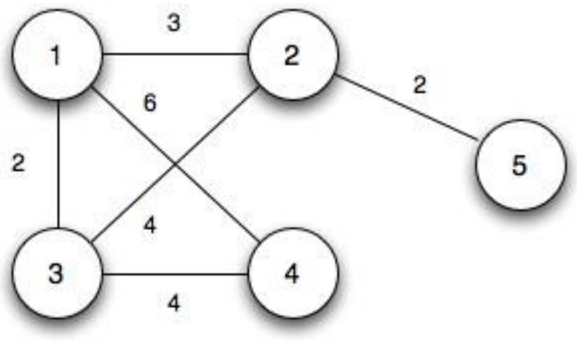
1. Define Optimal Solution.
2. Explain Travelling Sales Person Problem.
3. What is the time complexity of Travelling Sales Person Problem?

WEEK-11

MINIMUM COST SPANNING TREE

11.1 OBJECTIVE:

Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.



11.2 RESOURCES:

IDLE(python GUI)

11.3 PROGRAM LOGIC:

1) Create a set Sthat keeps track of vertices already included in MST.
2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3) While S doesn't include all vertices.

a) Pick a vertex *u* which is not there in Sand has minimum key value.
b) Include *u* to S.
c) Update key value of all adjacent vertices of *u*.

To update the key values, iterate through all adjacent vertices. For every adjacent vertex *v*, if weight of edge *u*-*v* is less than the previous key value of *v*, update the key value as weight of *u*-*v*

The idea of using key values is to pick the minimum weight edge from cut. The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST.

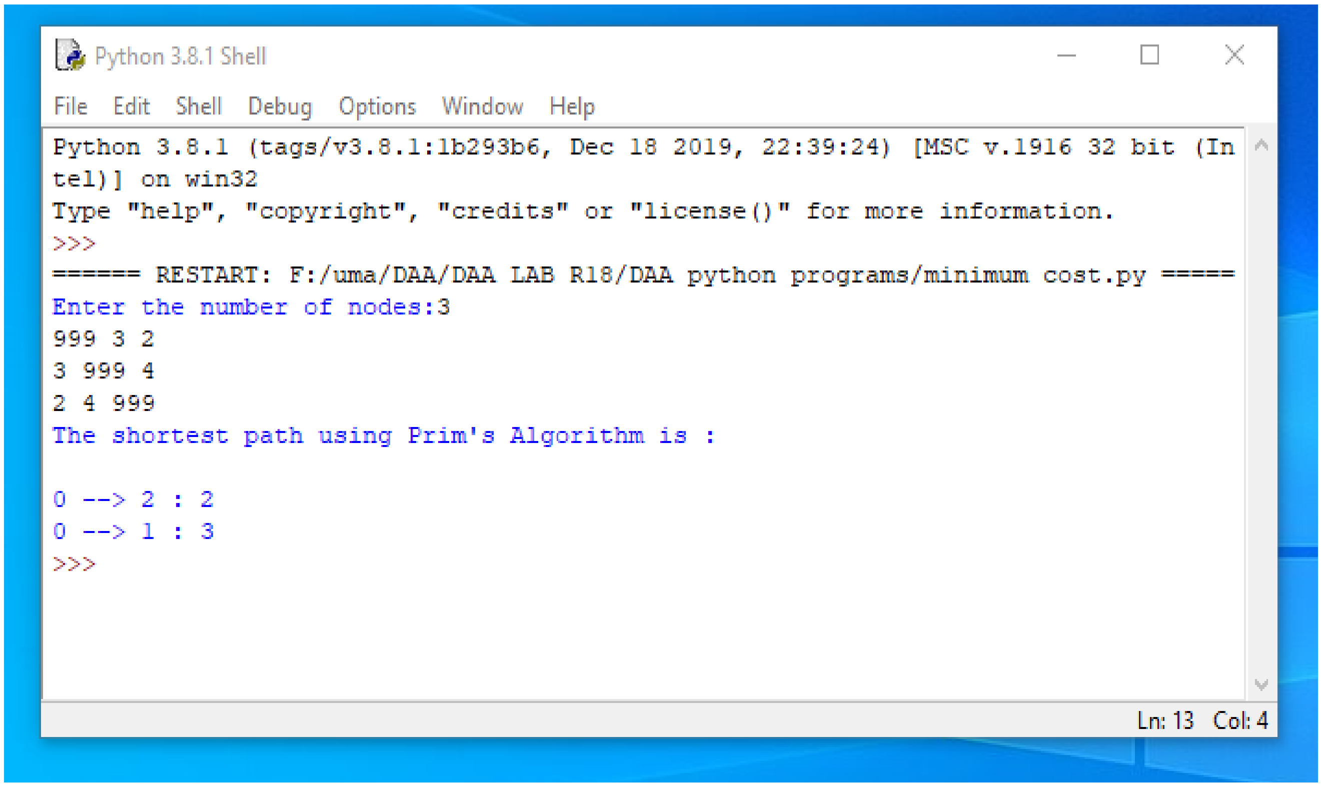
11.4 PROCEDURE:

1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module(F5)

11.5 SOURCE CODE.

```
maxi=9999999
n=int(input("Enter the number of nodes:"))
selected=[False for i in range(n)]
def main(n,cost):
    n_edge=0
    selected[0]=True
    while n_edge<n-1:
        minimum=maxi
        x=y=0
        for i in range(n):
            if selected[i]:
                for j in range(n):
                    if not selected[j] and cost[i][j]:
                        if minimum>cost[i][j]:
                            minimum=cost[i][j]
                            x=i
                            y=j
        print(x,'-->',y,':',cost[x][y])
        selected[y]=True
        n_edge+=1
cost=[[int(x) for x in input().split()]
      for j in range(n)]
for i in range(n):
    for j in range(n):
         if cost[i][j]==0:
             cost[i][j]=maxi
print("The shortest path using Prim's Algorithm is :\n")
main(n,cost)
```

11.6 INPUT/ OUTPUT



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
====== RESTART: F:/uma/DAA/DAA LAB R18/DAA python programs/minimum cost.py =====
Enter the number of nodes:3
999 3 2
3 999 4
2 4 999
The shortest path using Prim's Algorithm is :

0 --> 2 : 2
0 --> 1 : 3
>>>
```

11.7 LAB VIVA QUESTIONS:

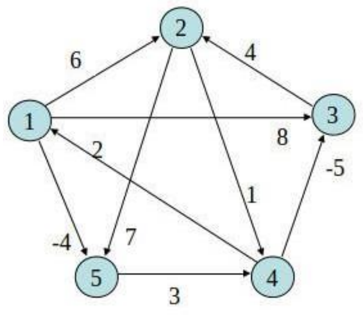
1. What is Minimum Cost spanning Tree.
2. Explain Prim's ALGORITHM.
3. What is time complexity of Prim's algorithm.

WEEK-12

ALL PAIRS SHORTEST PATHS

12.1 OBJECTIVE:

Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.



	1	2	3	4	5
1	0	6	8	∞	-4
2	∞	0	∞	1	7
3	∞	4	0	∞	∞
4	2	∞	-5	0	∞
5	∞	∞	∞	3	0

12.2 RESOURCES:

IDLE(python GUI)

12.3 PROGRAM LOGIC:

Initialize the solution matrix same as the input graph matrix as a first step. Then we update the solution matrix by considering all vertices as an intermediate vertex. The ideas is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.
When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices.

For every pair (i, j) of source and destination vertices respectively, there are two possible cases.

1) k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.

2) k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j].

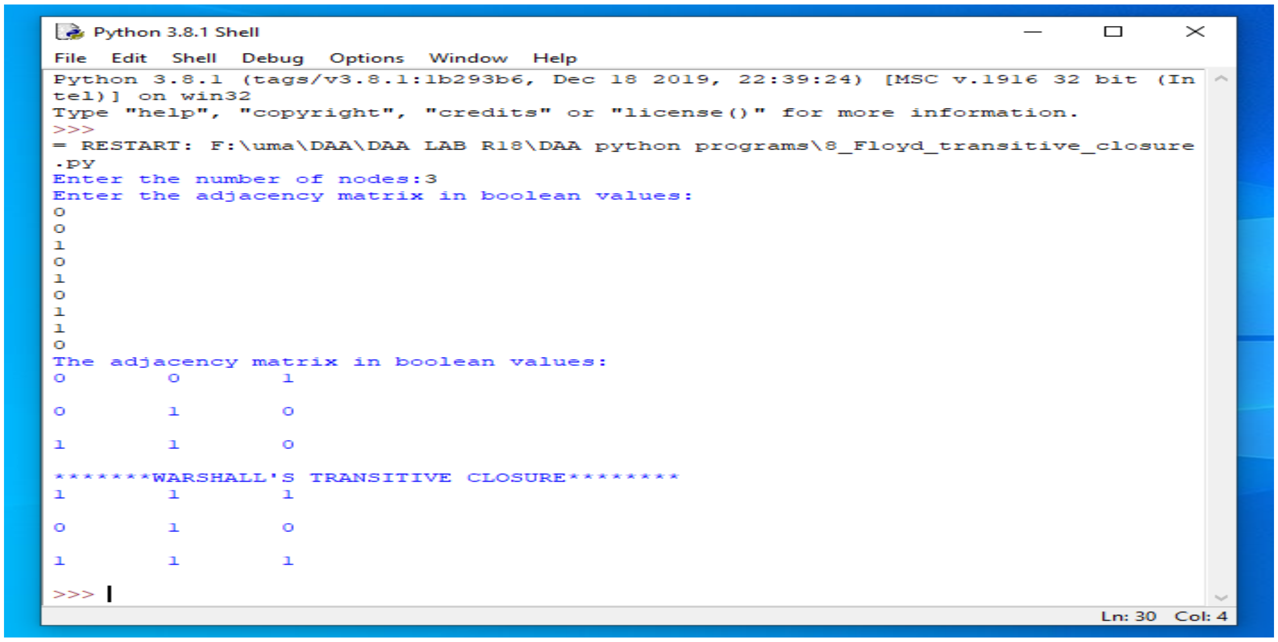
12.4 PROCEDURE:

1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module(F5)

12.5 SOURCE CODE.

```
n=int(input("Enter the
number of nodes:"))
graph=[[0 for i in range(n)]
for j in range(n)]
print("Enter the adjacency
matrix in boolean values:")
for i in range(n):
    for j in range(n):
        temp=int(input())
        graph[i][j]=temp
print("The adjacency matrix
in boolean values:")
for i in range(n):
    for j in range(n):

print(graph[i][j],end="\t")
    print("\n")
for k in range(n):
    for i in range(n):
        for j in range(n):
            if graph[i][k]==1
and graph[k][j]==1:
                graph[i][j]=1
print("*******WARSHAL
L'S TRANSITIVE
CLOSURE********")
for i in range(n):
    for j in range(n):

print(graph[i][j],end="\t")
    print("\n")
```

12.6 INPUT/ OUTPUT



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: F:\uma\DAA\DAA LAB R18\DAA python programs\8_Floyd_transitive_closure
.py
Enter the number of nodes:3
Enter the adjacency matrix in boolean values:
0
0
1
0
1
0
1
1
0
The adjacency matrix in boolean values:
0        0        1

0        1        0

1        1        0

*******WARSHALL'S TRANSITIVE CLOSURE********
1        1        1

0        1        0

1        1        1

>>> 
```

12.7 LAB VIVA QUESTIONS:

1. What is Floyd’s algorithm?
2. What is the time complexity of Floyd’s algorithm?
3. Define Distance Matrix.

WEEK-13

N QUEENS PROBLEM

13.1 OBJECTIVE:

Implement N Queen's problem using Back Tracking.

13.2 RESOURCES:

IDLE(python GUI)

13.3 PROGRAM LOGIC:

1) Start in the leftmost column
2) If all queens are placedreturn true
3) Try all rows in the current column. Do following for every tried row.
   a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
   b) If placing queen in [row, column] leads to a solution then return true.
   c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.
3) If all rows have been tried and nothing worked, return false to trigger Backtracking.

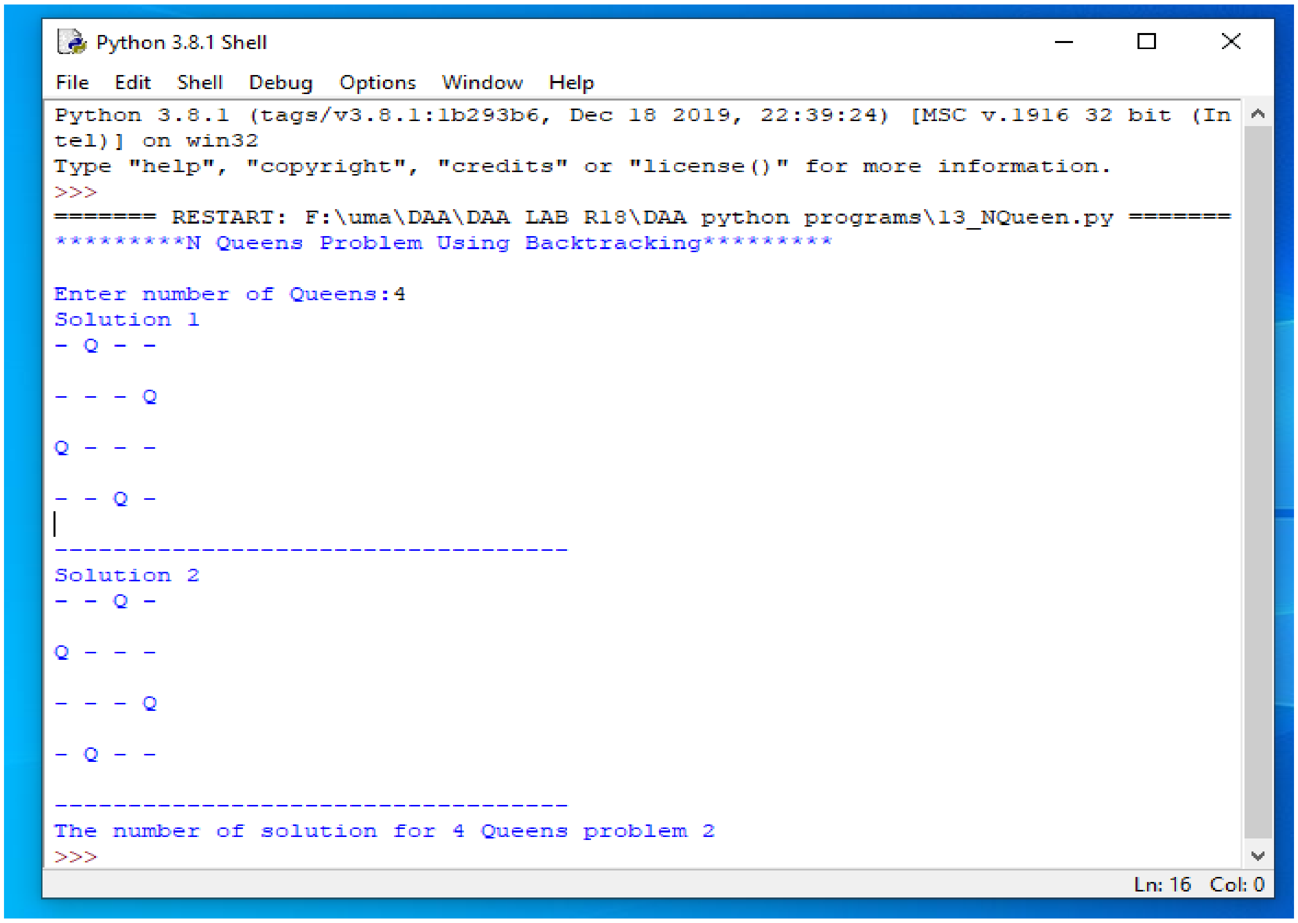
13.4 PROCEDURE:

1. Create: Open IDLE(python GUI), write a program after that save the program with .c extension.
2. Run Module(F5)

13.5 SOURCE CODE:

```
board=[0 for i in range(20)]
ctr=0
def printx(n):
    global ctr
    ctr=ctr+1
    print("Solution",ctr)
    for i in range(1,n+1,1):
        for j in range(1,n+1,1):
            if(board[i]==j):
                print("Q",end=' ')
            else:
                print("-",end=' ')
        print("\n")
    print("-----------------------------------")
def place(r,c):
    for i in range(1,r,1):
        if(board[i]==c):
            return 0
        else:
            if(abs(board[i]-c)==abs(i-r)):
                return 0
    return 1
def queen(r,n):
    for c in range(1,n+1,1):
        if(place(r,c)):
            board[r]=c
            if(r==n):
                printx(n)
            else:
                queen(r+1,n);
print("*********N Queens Problem Using Backtracking*********")
n=int(input("\nEnter number of Queens:"))
queen(1,n)
print("The number of solution for",n,"Queens problem",ctr)
```

13.6 INPUT/ OUTPUT



```
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: F:\uma\DAA\DAA LAB R18\DAA python programs\13_NQueen.py =======
*********N Queens Problem Using Backtracking*********

Enter number of Queens:4
Solution 1
- Q - -

- - - Q

Q - - -

- - Q -

-----------------------------------
Solution 2
- - Q -

Q - - -

- - - Q

- Q - -

-----------------------------------
The number of solution for 4 Queens problem 2
>>>
```

13.7 LAB VIVA QUESTIONS:

1. Define backtracking.
2. Define live node, dead node.
3. Define implicit and explicit constraints.
4. What is the time complexity of n-queens problem.