

DESIGN AND ANALYSIS OF ALGORITHMS

LAB MANUAL

Academic Year : 2017 - 2018
Course Code : AIT101
Regulations : IARE - R16
Semester : III
Branch : CSE and IT

Prepared by

Mr.CH SURESH KUMAR RAJU, ASSISTANT PROFESSOR
Mr.Y SUBBA RAYUDU, ASSISTNT PROFESSOR



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



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

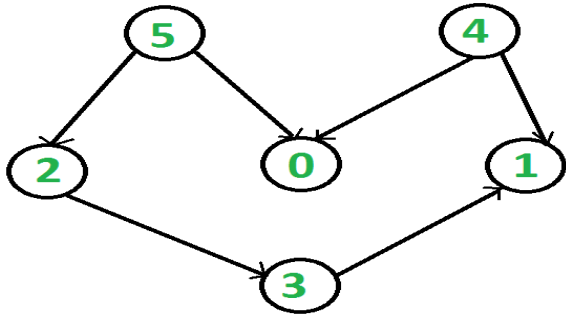
1. PROGRAM OUTCOMES:

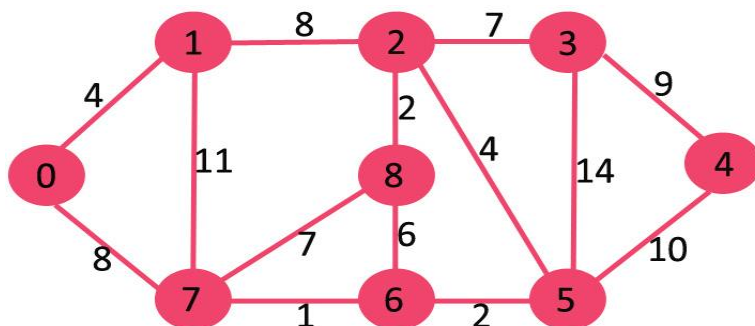
B.TECH - PROGRAM OUTCOMES (POS)	
PO-1	Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems (Engineering knowledge).
PO-2	Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences (Problem analysis).
PO-3	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 (Design/development of solutions).
PO-4	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 (Conduct investigations of complex problems).
PO-5	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 (Modern tool usage).
PO-6	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 (The engineer and society).
PO-7	Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development (Environment and sustainability).
PO-8	Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice (Ethics).
PO-9	Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings (Individual and team work).
PO-10	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 (Communication).
PO-11	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 (Project management and finance).
PO-12	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 (Life-long learning).

2. PROGRAM SPECIFIC OUTCOMES

PROGRAM SPECIFIC OUTCOMES (PSO's)	
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.

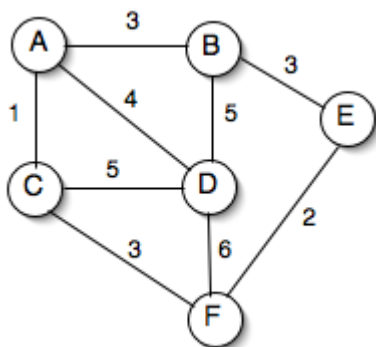
3. ATTAINMENT OF PROGRAM OUTCOMES AND PROGRAM SPECIFIC OUTCOMES:

S. No	Experiment	Program Outcomes Attained	Program Specific Outcomes Attained
WEEK-1	QUICK SORT	PO-2,PO-3	PSO-1
	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 1 st 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	PO-2,PO-3	PSO-1
	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	PO-2	
	<p>a. Obtain the Topological ordering of vertices in a given digraph.</p>  <p>b. Compute the transitive closure of a given directed graph using Warshall's algorithm.</p>		
WEEK-4	KNAPSACK PROBLEM	PO-3	PSO-1
	Implement 0/1 Knapsack problem using Dynamic Programming.		
WEEK-5	SHORTEST PATHS ALGORITHM	PO-3	PSO-1
	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.		



MINIMUM COST SPANNING TREE

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



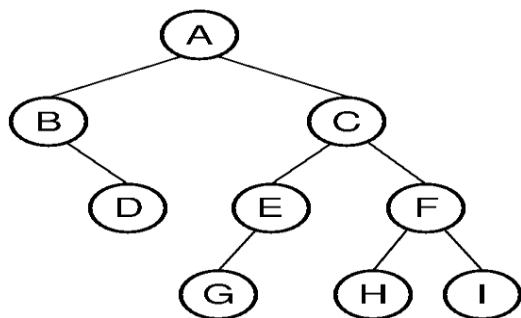
WEEK-6

PO-3

PSO-1

TREE TRAVERSALS

Perform various tree traversal algorithms for a given tree.



WEEK-7

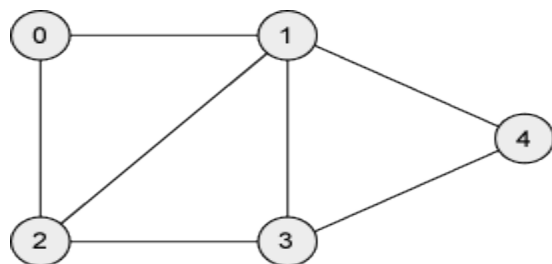
PO-2,PO-3

GRAPH TRAVERSALS

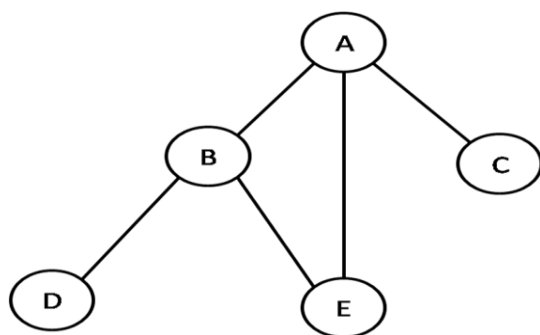
- Print all the nodes reachable from a given starting node in a digraph using BFS method.

WEEK-8

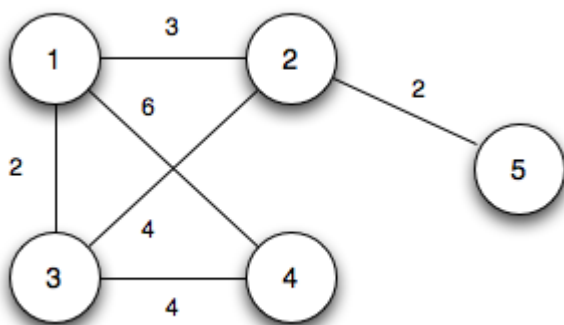
PO-2,PO-3



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

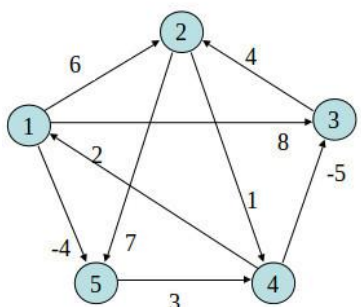


	<p>b. Check whether a given graph is connected or not using DFS method.</p>		
WEEK-9	SUM OF SUB SETS PROBLEM Find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d . For example, if $S = \{1, 2, 5, 6, 8\}$ and $d = 9$ there 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.	PO-3	
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.	PO-3, PO-12	PSO-1
WEEK-11	MINIMUM COST SPANNING TREE Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.	PO-3	PSO-1



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

PO-3

PSO-1

N QUEENS PROBLEM

Implement N Queen's problem using Back Tracking.

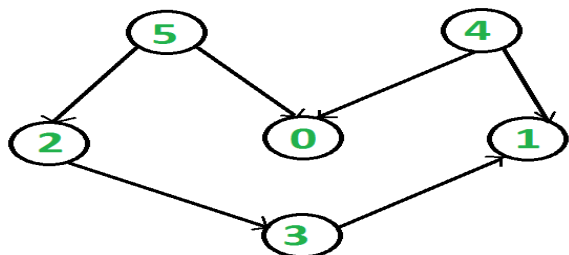
PO-3

PSO-1

**4. MAPPING COURSE OBJECTIVES LEADING TO THE ACHIEVEMENT OF
PROGRAM OUTCOMES AND PROGRAM SPECIFIC OUTCOMES:**

Course Objectives	Program Outcomes												Program Specific Outcomes		
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO 10	PO 11	PO 12	PSO1	PSO2	PSO3
I		√	√									√	√		
II		√	√									√	√		

5. SYLLABUS:

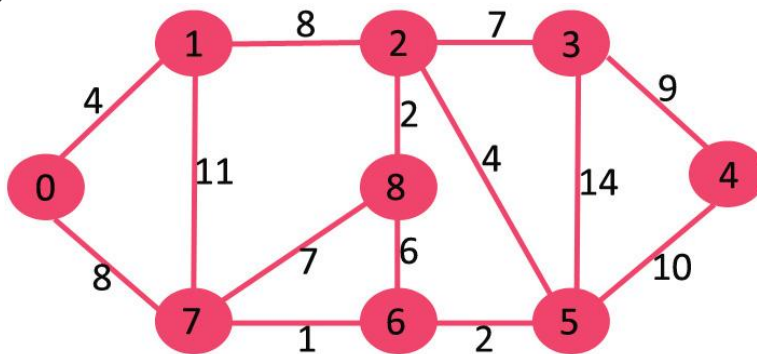
DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY								
III Semester: CSE / IT								
Course Code	Category	Hours / Week			Credits	Maximum Marks		
AIT101	Core	L	T	P	C	CIA	SEE	Total
		-	-	3	2	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 39			Total Classes: 39			
OBJECTIVES: The course should enable the students to: Learn how to analyze a problem and design the solution for the problem. I. Design and implement efficient algorithms for a specified application. II. Strengthen the ability to identify and apply the suitable algorithm for the given real world 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 1 st 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							
c. Obtain the Topological ordering of vertices in a given digraph.								
								
d. Compute the transitive closure of a given directed graph using Warshall's algorithm.								

WEEK-4 KNAPSACK PROBLEM

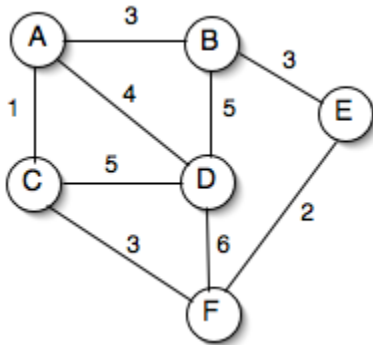
Implement 0/1 Knapsack problem using Dynamic Programming.

WEEK-5 SHORTEST PATHS ALGORITHM

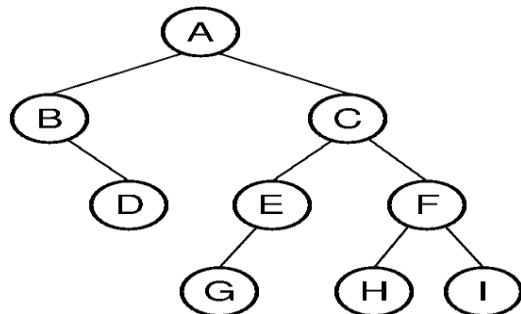
From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

**WEEK-6 MINIMUM COST SPANNING TREE**

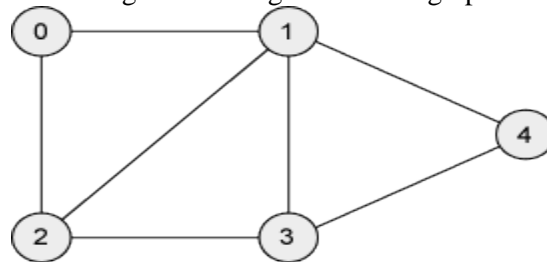
Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

**WEEK-7 TREE TRAVERSALS**

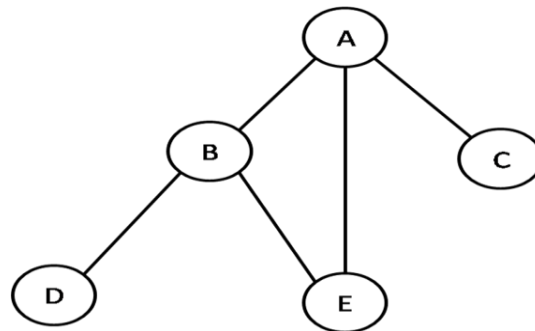
Perform various tree traversal algorithms for a given tree.

**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

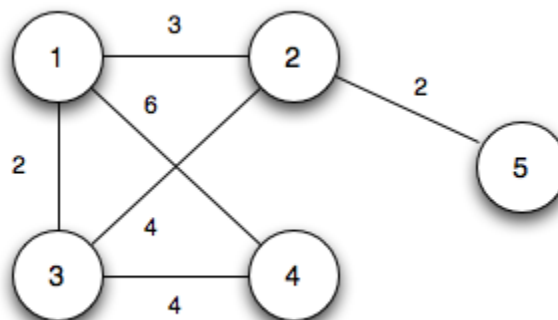
Find a subset of a given set $S = \{s_1, s_2, \dots, s_n\}$ of n positive integers whose sum is equal to a given positive integer d . For example, if $S = \{1, 2, 5, 6, 8\}$ and $d = 9$ there are two solutions $\{1, 2, 6\}$ and $\{1, 8\}$. A suitable message is to be displayed if the given problem instance 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.

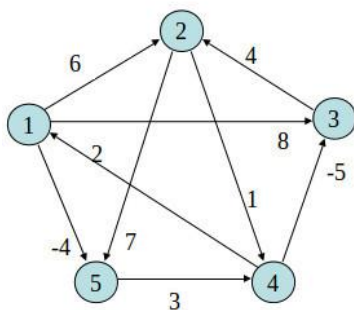
WEEK-11 MINIMUM COST SPANNING TREE

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 Wiley 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>

SOFTWARE AND HARDWARE REQUIREMENTS FOR A BATCH OF 36 STUDENTS:**HARDWARE:**

Desktop Computer Systems: 36 nos

SOFTWARE:

Application Software: C Programming Compiler

6. INDEX:

S. No	Experiment	Page No
1	QUICK SORT	14
2	MERGE SORT	17
3	WARSHALL'S ALGORITHM	20
4	KNAPSACK PROBLEM	24
5	SHORTEST PATHS ALGORITHM	26
6	MINIMUM COST SPANNING TREE	29
7	TREE TRAVERSALS	32
8	GRAPH TRAVERSALS	38
9	SUM OF SUB SETS PROBLEM	42
10	TRAVELLING SALES PERSON PROBLEM	45
11	MINIMUM COST SPANNING TREE	48
12	ALL PAIRS SHORTEST PATHS	51
13	N QUEENS PROBLEM	55

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:

Dev C++

1.3 PROGRAM LOGIC:

Quick Sort 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 Quick Sort 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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

1.5 SOURCE CODE:

```
include <stdio.h>
include <time.h>
void Exch(int *p, int *q) {
    int temp = *p;
    *p = *q;
    *q = temp;
}
void QuickSort(int a[], int low, int high) {
    int i, j, key, k;
    if(low >= high)
        return;
    key = low;
```

```

        i=low+1;
        j=high;
        while(i<=j) {
            while ( a[i] <= a[key] )
                i=i+1;
            while ( a[j] > a[key] )
                j=j -1;
            if(i<j)
                Exch(&a[i], &a[j]);
        }
        Exch(&a[j], &a[key]);
        QuickSort(a, low, j-1);
        QuickSort(a, j+1, high);
    }
}

void main() {
    int n, a[1000],k;
    clock_tst,et; double ts; clrscr();
    printf("\n Enter How many Numbers: ");
    scanf("%d", &n);
    printf("\nThe Random Numbers are:\n");
    for(k=1; k<=n; k++) {
        a[k]=rand();
        printf("%d\t",a[k]);
    }
    st=clock();
    QuickSort(a, 1, n);
    et=clock();
    ts=(double)(et-st)/CLOCKS_PER_SEC;
    printf("\nSorted Numbers are: \n ");
    for(k=1; k<=n; k++)
        printf("%d\t", a[k]);
    printf("\nThe time taken is %e",ts);
}

```

1.6 INPUT/ OUTPUT

```

D:\suresh\DAAB LAB PROGRAMS\quicksort.exe
Enter How many Numbers: 30
The Random Numbers are:
41 18467 6334 26500 19169 15724 11478 29358 26962 24464
5705 28145 23281 16827 9961 491 2995 11942 4827 5436
32391 14604 3902 153 292 12382 17421 18716 19718 19895

st=5136
et=5136
CLOCKS_PER_SEC=1000
Sorted Numbers are:
41 153 292 491 2995 3902 4827 5436 5705 6334
9961 11478 11942 12382 14604 15724 16827 17421 18467 18716
19169 19718 19895 23281 24464 26500 26962 28145 29358 32391

It took me 0 clicks (0.000000 seconds).

-----
Process exited after 5.323 seconds with return value 41
Press any key to continue . . .

```

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:

Dev C++

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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

2.5 SOURCE CODE:

```
#include <stdio.h>
#include<time.h>
int b[50000];
void Merge(int a[], int low, int mid, int high){
    int i, j, k;
    i=low; j=mid+1; k=low;
    while ( i<=mid && j<=high )    {
        if( a[i] <= a[j] )
            b[k++] = a[i++] ;
        else
            b[k++] = a[j++] ;
    }
    while (i<=mid)
        b[k++] = a[i++] ;
    while (j<=high)
        b[k++] = a[j++] ;
    for(k=low; k<=high; k++)
        a[k] = b[k];
}
```

```

void MergeSort(int a[], int low, int high) {
    int mid;
    if(low >= high)
        return;
    mid = (low+high)/2 ;
    MergeSort(a, low, mid);
    MergeSort(a, mid+1, high);
    Merge(a, low, mid, high);
}

void main() {
    int n, a[50000], k;
    clock_t st, et;
    double ts;
    printf("\n Enter How many Numbers:");
    scanf("%d", &n);
    printf("\n The Random Numbers are:\n");
    for(k=1; k<=n; k++) {
        a[k]=rand();
        printf("%d\t", a[k]);
    }
    st=clock();
    MergeSort(a, 1, n);
    et=clock();
    ts=(double)(et-st)/CLOCKS_PER_SEC;
    printf("\n Sorted Numbers are : \n ");
    for(k=1; k<=n; k++)
        printf("%d\t", a[k]);
    printf("\n The time taken is %e", ts);
}

```

2.6 INPUT/ OUTPUT

```

D:\suresh\DAALAB PROGRAMS\mergesort.exe

Enter How many Numbers:20

The Random Numbers are:
41      18467   6334    26500    19169    15724    11478    29358    26962    24464
5705    28145   23281    16827    9961     491      2995     11942    4827     5436

Sorted Numbers are :
41      491      2995     4827     5436     5705     6334     9961     11478    11942
15724    16827    18467    19169    23281    24464    26500    26962    28145    29358

The time taken is 0.000000e+000

-----
Process exited after 4.915 seconds with return value 32
Press any key to continue . . .

```

2.7 LAB VIVA QUESTIONS:

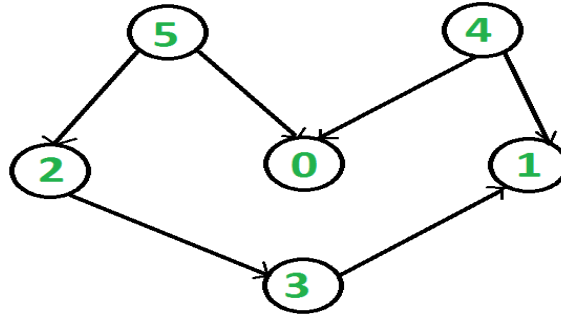
1. What is the running time of merge sort?
2. What technique is used to sort elements in merge sort?
3. Is merge sort in place sorting algorithm?
4. Define stable sort algorithm.

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:

Dev C++

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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

3.5 SOURCE CODE:

```
// Topological ordering
#include<stdio.h>

int a[10][10],n,indegre[10];
void find_indegre () {
    int j,i,sum;
    for(j=0;j<n;j++) {
```

```

        sum=0;
        for(i=0;i<n;i++)
            sum+=a[i][j];
            indegre[j]=sum;
        }
    }
    void topology() {
        inti,u,v,t[10],s[10],top= -1,k=0;
        find_indegre();
        for(i=0;i<n;i++) {
            if(indegre[i]==0)
                s[++top]=i;
        }
        while(top!= -1) {
            u=s[top--];
            t[k++]=u; //top element of stack is stored in temporary array
            for(v=0;v<n;v++) {
                if(a[u][v]==1) {
                    indegre[v]--;
                    if(indegre[v]==0)
                        s[++top]=v;    //Pushing adjacent vertex to stack
                }
            }
        }
        printf ("The topological Sequence is:\n");
        for(i=0;i<n;i++)
            printf ("%d ",t[i]);
    }
    void main() {
        inti,j;
        printf("Enter number of jobs:");
        scanf("%d",&n);
        printf("\nEnter the adjacency matrix:\n");
        for(i=0;i<n;i++) {
            for(j=0;j<n;j++)
                scanf("%d",&a[i][j]);
        }
        topology();
    }

```

//Transitive closure of a graph using Warshall's algorithm

```

#include <stdio.h>
intn,a[10][10],p[10][10];
void path() {
    inti,j,k;
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            p[i][j]=a[i][j];
    for(k=0;k<n;k++)
        for(i=0;i<n;i++)

```

```

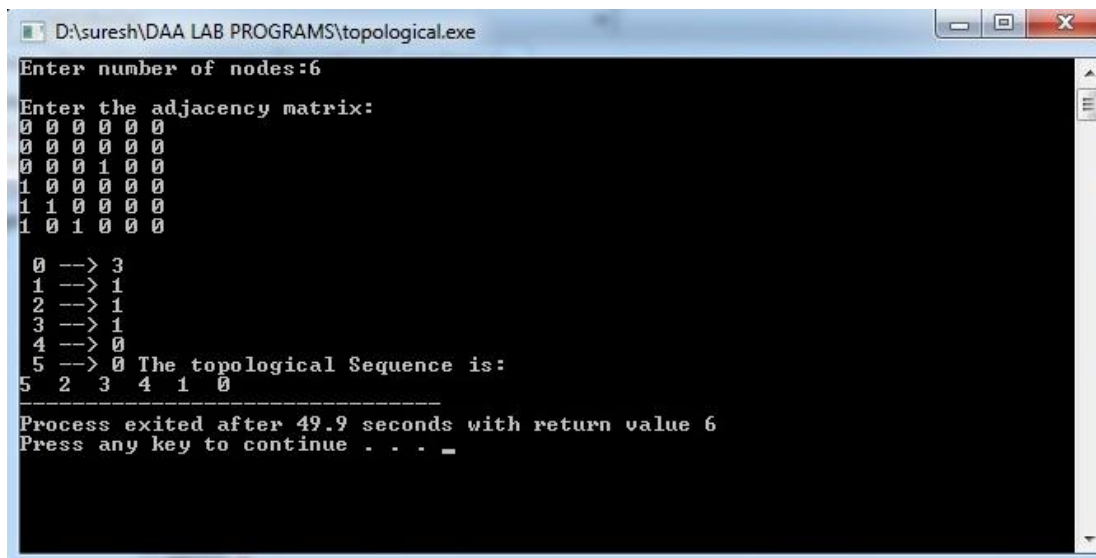
        for(j=0;j<n;j++)
            if(p[i][k]==1&& p[k][j]==1)
                p[i][j]=1;
    }
void main() {
    inti,j;
    printf("Enter the number of nodes:");
    scanf("%d",&n);
    printf("\nEnter the adjacency matrix:\n");
    for(i=0;i<n;i++)
        for(j=0;j<n;j++)
            scanf("%d",&a[i][j]);

    path();
    printf("\nThe path matrix is shown below\n");
    for(i=0;i<n;i++) {
        for(j=0;j<n;j++)
            printf("%d ",p[i][j]);
        printf("\n");
    }
}

```

3.6 INPUT/ OUTPUT

Topological ordering



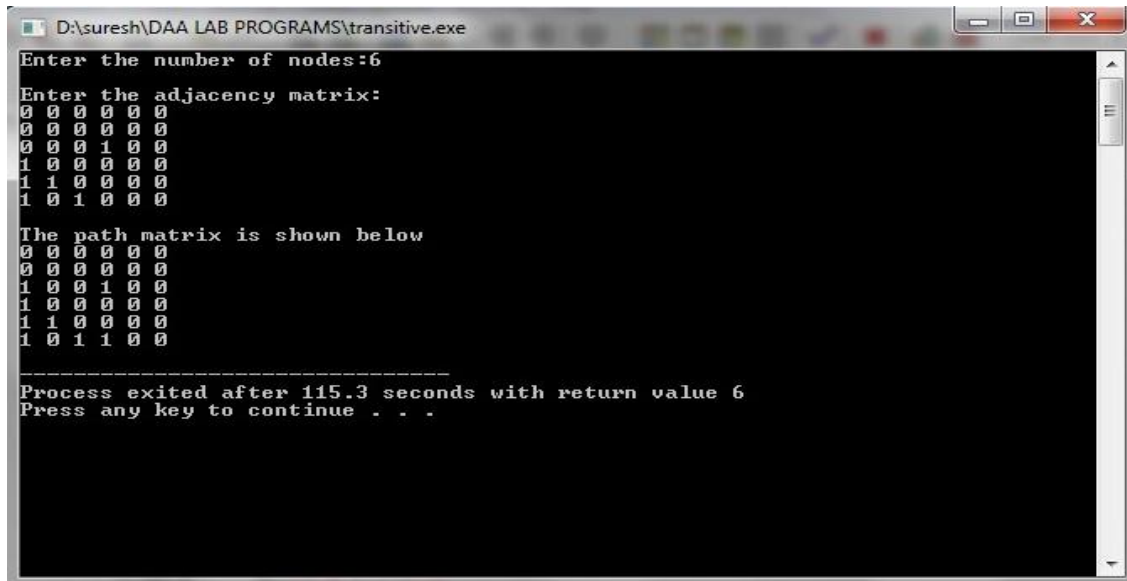
```

D:\suresh\DAA LAB PROGRAMS\topological.exe
Enter number of nodes:6
Enter the adjacency matrix:
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 0
1 1 0 0 0 0
1 0 1 0 0 0

0 --> 3
1 --> 1
2 --> 1
3 --> 1
4 --> 0
5 --> 0 The topological Sequence is:
5 2 3 4 1 0
-----
Process exited after 49.9 seconds with return value 6
Press any key to continue . . . _

```

Transitive closure of a graph using Warshall's algorithm



```
D:\suresh\DAA LAB PROGRAMS\transitive.exe
Enter the number of nodes:6
Enter the adjacency matrix:
0 0 0 0 0 0
0 0 0 0 0 0
0 0 0 1 0 0
1 0 0 0 0 0
1 1 0 0 0 0
1 0 1 0 0 0

The path matrix is shown below
0 0 0 0 0 0
0 0 0 0 0 0
1 0 0 1 0 0
1 0 0 0 0 0
1 1 0 0 0 0
1 0 1 1 0 0

-----
Process exited after 115.3 seconds with return value 6
Press any key to continue . . .
```

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:

Dev C++

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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

4.5 SOURCE CODE:

```
#include<stdio.h>
int w[10],p[10],v[10][10],n,i,j,cap,x[10]={0};
int max(int i,int j){
    return ((i>j)?i:j);
}
int knap(int i,int j) {
    int value;
    if(v[i][j]<0) {
        if(j<w[i])
            value=knap(i-1,j);
        else
            value=max(knap(i-1,j),p[i]+knap(i-1,j-w[i]));
        v[i][j]=value;
    }
    return(v[i][j]);
}
int main() {
    int profit,count=0;
    printf("\nEnter the number of objects ");
    scanf("%d",&n);
    printf("Enter the profit and weights of the elements \n ");
    for(i=1;i<=n;i++) {
        printf("\nEnter profit and weight For object no %d :",i);
        scanf("%d%d",&p[i],&w[i]);
    }
    printf("\nEnter the capacity ");
```



```

scanf("%d",&cap);
for(i=0;i<=n;i++)
    for(j=0;j<=cap;j++)
        if((i==0)||(j==0))
            v[i][j]=0;
        else
            v[i][j]=-1;

profit=knap(n,cap);
i=n;
j=cap;
while(j!=0&& i!=0) {
    if(v[i][j]!=v[i-1][j]) {
        x[i]=1;
        j=j-w[i];
        i--;
    }
    else
        i--;
}
printf("object included are \n ");
printf("Sl.no\tweight\tprofit\n");
for(i=1;i<=n;i++)
    if(x[i])
        printf("%d\t%d\t%d\n",++count,w[i],p[i]);
printf("Total profit = %d\n",profit);
}

```

4.6 INPUT/ OUTPUT

```

C:\Users\Administrator\Desktop\suresh\DAAB LAB PROGRAMS\KNAPSACK .exe
Enter the number of objects 3
Enter the profit and weights of the elements
Enter profit and weight For object no 1 :1 2
Enter profit and weight For object no 2 :2 3
Enter profit and weight For object no 3 :5 4
Enter the capacity 6
object included are
Sl.no  weight  profit
1       2       1
2       4       5
Total profit = 6

-----
Process exited after 24.4 seconds with return value 0
Press any key to continue . . .

```

4.7 LAB VIVA QUESTIONS:

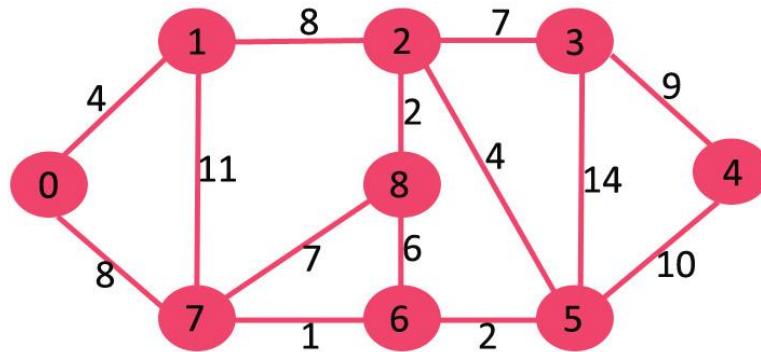
1. Define knapsack problem.
2. Define principle of optimality.
3. What is the optimal solution for knapsack problem?
4. What is the time complexity of knapsack problem?

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:

Dev C++

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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

5.5 SOURCE CODE:

```
#include<stdio.h>
#define infinity 999
void dij(int n, int v, int cost[20][20], int dist[]) {
```

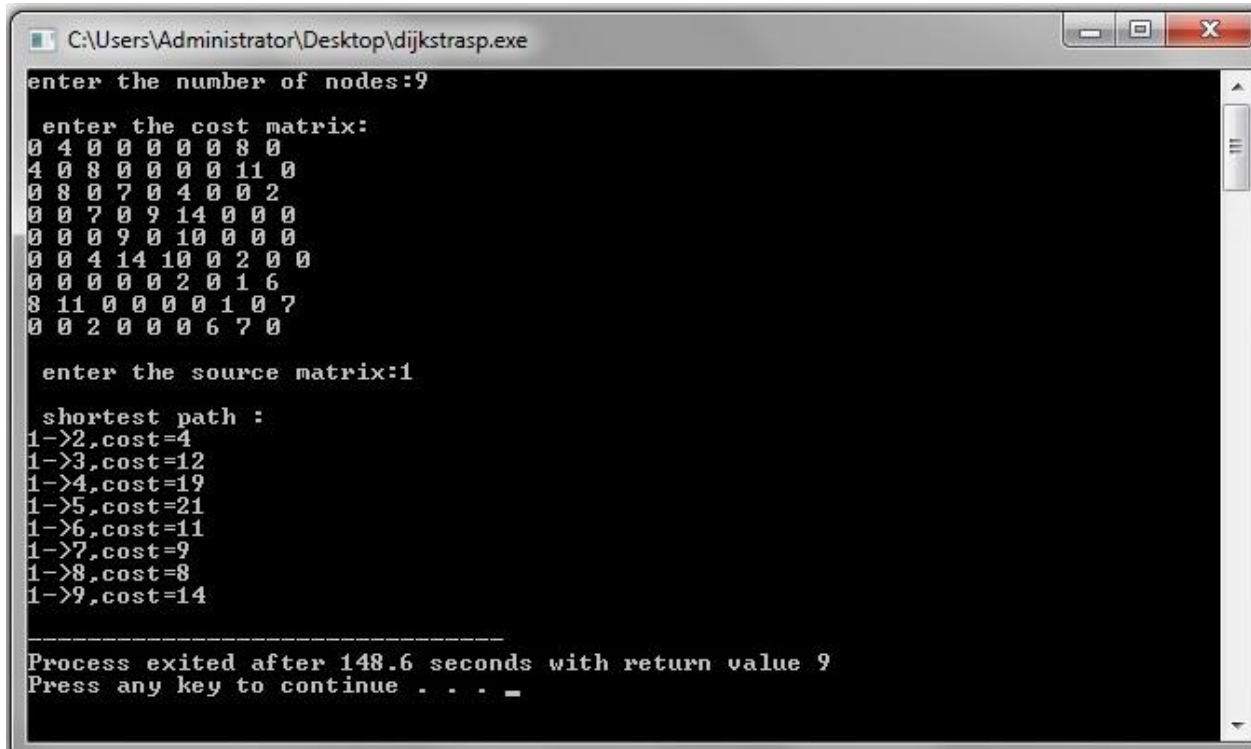
```

int i,u,count,w,flag[20],min;
for(i=1;i<=n;i++)
    flag[i]=0, dist[i]=cost[v][i];
count=2;
while(count<=n) {
    min=99;
    for(w=1;w<=n;w++)
        if(dist[w]<min && !flag[w]) {
            min=dist[w];
            u=w;
        }
    flag[u]=1;
    count++;
    for(w=1;w<=n;w++)
        if((dist[u]+cost[u][w]<dist[w]) && !flag[w])
            dist[w]=dist[u]+cost[u][w];
}
}

int main() {
    int n,v,i,j,cost[20][20],dist[20];
    printf("enter the number of nodes:");
    scanf("%d",&n);
    printf("\n enter the cost matrix:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) {
            scanf("%d",&cost[i][j]);
            if(cost[i][j] == 0)
                cost[i][j]=infinity;
        }
    printf("\n enter the source matrix:");
    scanf("%d",&v);
    dij(n,v,cost,dist);
    printf("\n shortest path : \n");
    for(i=1;i<=n;i++)
        if(i!=v)
            printf("%d->%d,cost=%d\n",v,i,dist[i]);
}

```

5.6 INPUT/ OUTPUT



```
C:\Users\Administrator\Desktop\dijkstrasp.exe
enter the number of nodes:9

enter the cost matrix:
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

enter the source matrix:1

shortest path :
1->2, cost=4
1->3, cost=12
1->4, cost=19
1->5, cost=21
1->6, cost=11
1->7, cost=9
1->8, cost=8
1->9, cost=14

-----
Process exited after 148.6 seconds with return value 9
Press any key to continue . . . _
```

5.7 LAB VIVA QUESTIONS:

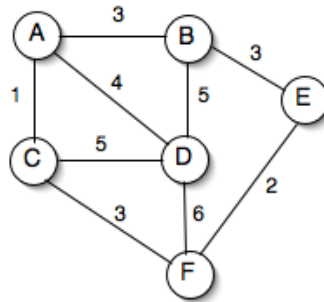
1. What is the time complexity of Dijkstra's algorithm?
2. Define cost matrix.
3. Define directed graph.
4. Define connected graph.

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:

Dev C++

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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

6.5 SOURCE CODE:

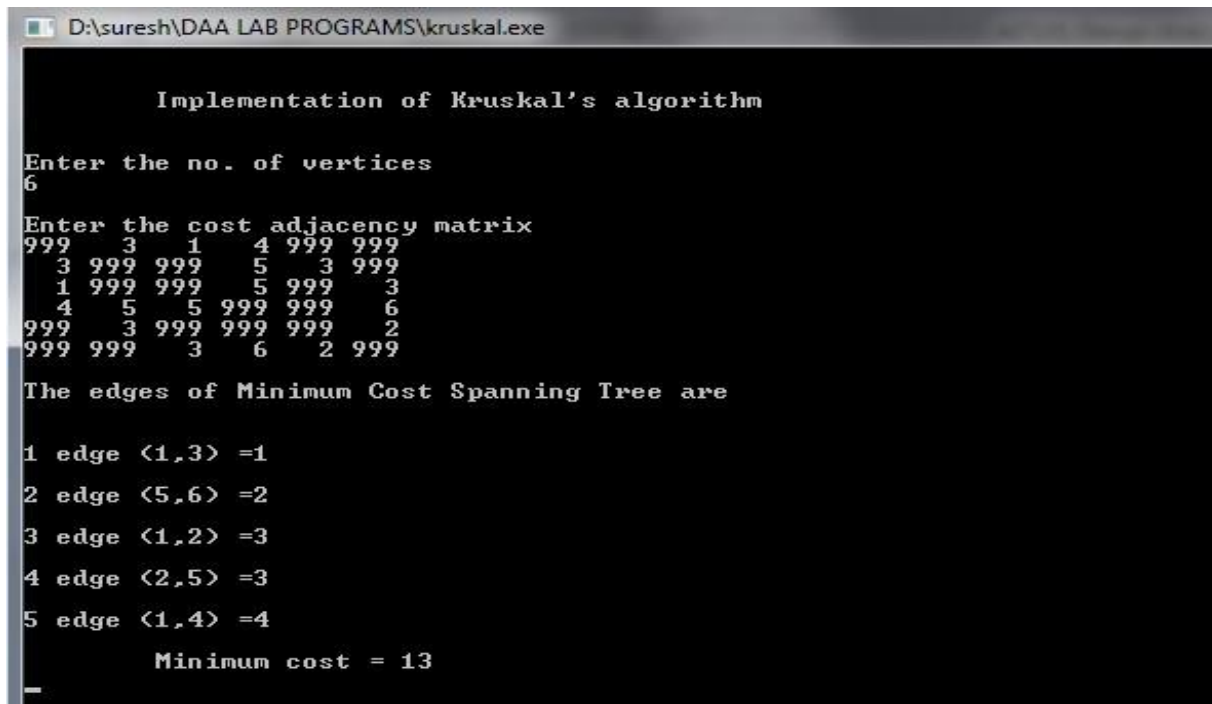
```
#include<stdio.h>
#include<stdlib.h>
inti,j,k,a,b,u,v,n,ne=1;
intmin,mincost=0,cost[9][9],parent[9];
int find(int);
intuni(int,int);
void main() {
    printf("\n Implementation of Kruskal's algorithm\n\n");
    printf("\nEnter the no. of vertices\n");
    scanf("%d",&n);
    printf("\nEnter the cost adjacency matrix\n");
    for(i=1;i<=n;i++) {
        for(j=1;j<=n;j++) {
            scanf("%d",&cost[i][j]);
```

```

        if(cost[i][j]==0)
            cost[i][j]=999;
    }
}
printf("\nThe edges of Minimum Cost Spanning Tree are\n\n");
while(ne<n) {
    for(i=1,min=999;i<=n;i++) {
        for(j=1;j<=n;j++) {
            if(cost[i][j]<min) {
                min=cost[i][j];
                a=u=i;
                b=v=j;
            }
        }
    }
    u=find(u);
    v=find(v);
    if(uni(u,v)) {
        printf("\n%d edge (%d,%d) =%d\n",ne++,a,b,min);
        mincost +=min;
    }
    cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
}
int find(int i) {
    while(parent[i])
        i=parent[i];
    return i;
}
int uni(inti,int j){
    if(i!=j) {
        parent[j]=i;
        return 1;
    }
    return 0;
}

```

6.6 INPUT/ OUTPUT



```
D:\suresh\DAA LAB PROGRAMS\kruskal.exe

Implementation of Kruskal's algorithm

Enter the no. of vertices
6

Enter the cost adjacency matrix
999 3 1 4 999 999
3 999 999 5 3 999
1 999 999 5 999 3
4 5 5 999 999 6
999 3 999 999 999 2
999 999 3 6 2 999

The edges of Minimum Cost Spanning Tree are

1 edge <1,3> =1
2 edge <5,6> =2
3 edge <1,2> =3
4 edge <2,5> =3
5 edge <1,4> =4

Minimum cost = 13
```

6.7 LAB VIVA QUESTIONS:

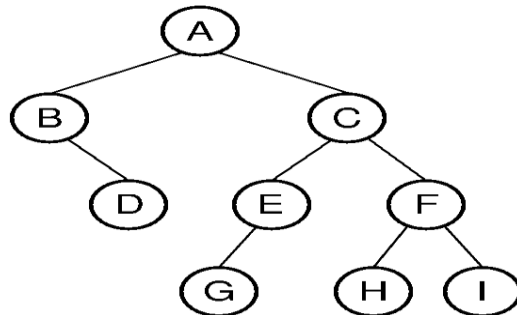
1. What is the time complexity of Kruskal's algorithm.
2. Define spanning tree.
3. Define minimum cost spanning tree.

WEEK-7

TREE TRAVERSALS

7.1 OBJECTIVE:

Perform various tree traversal algorithms for a given tree.



7.2 RESOURCES:

Dev C++

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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

7.5 SOURCE CODE:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct treeNode {
    int data;
    struct treeNode *left;
    struct treeNode *right;
}treeNode;

treeNode* FindMin(treeNode *node) {
    if(node==NULL) { /* There is no element in the tree */
        return NULL;
    }
    if(node->left) /* Go to the left sub tree to find the min element */
        return FindMin(node->left);
    else
        return node;
}

treeNode * insert(treeNode *node,int data) {
    if(node==NULL) {
        treeNode *temp;
        temp = (treeNode *)malloc(sizeof(treeNode));
        temp -> data = data;
        temp -> left = temp -> right = NULL;
        return temp;
    }
    if(data >(node->data)) {
        node->right = insert(node->right,data);
    }
    else if(data < (node->data)) {
        node->left = insert(node->left,data);
    }
    /* Else there is nothing to do as the data is already in the tree. */
    return node;
}

treeNode * deletion(treeNode *node, int data) {
    treeNode *temp;
    if(node==NULL) {
        printf("Element Not Found");
    }
    else if(data < node->data) {
        node->left = deletion(node->left, data);
    }
    else if(data > node->data) {
        node->right = deletion(node->right, data);
    }
    else {
        /* Now We can delete this node and replace with either minimum element in the right sub
        tree or maximum element in the left subtree */
    }
}
```

```

        if(node->right && node->left) {
            /* Here we will replace with minimum element in the right sub tree */
            temp = FindMin(node->right);
            node->data = temp->data;
            /* As we replaced it with some other node, we have to delete that node */
            node->right = deletion(node->right,temp->data);
        }
        else {
            /* If there is only one or zero children then we can directly remove it
            from the tree and connect its parent to its child */
            temp = node;
            if(node->left == NULL)
                node = node->right;
            else if(node->right == NULL)
                node = node->left;
            free(temp); /* temp is longer required */
        }
    }
    return node;
}

treeNode * search(treeNode *node, int data) {
    if(node==NULL) { /* Element is not found */
        return NULL;
    }
    if(data > node->data) { /* Search in the right sub tree. */
        return search(node->right,data);
    }
    else if(data < node->data) { /* Search in the left sub tree. */
        return search(node->left,data);
    }
    else { /* Element Found */
        return node;
    }
}

void inorder(treeNode *node) {
    if(node!=NULL) {
        inorder(node->left);
        printf("%d ",node->data);
        inorder(node->right);
    }
    else return;
}

void preorder(treeNode *node) {
    if(node!=NULL) {
        printf("%d ",node->data);
        preorder(node->left);
        preorder(node->right);
    }
}

```

```

        else return;
    }

void postorder(treeNode *node) {
    if(node!=NULL) {
        postorder(node->left);
        postorder(node->right);
        printf("%d ",node->data);
    }
    else return;
}

void main() {
    treeNode *t,*root = NULL;
    int ch, elt;
    do {
        printf("\n ### Binary Search Tree Operations ###");
        printf("\n Press 1-Creation of BST");
        printf("\n    2-deleting ");
        printf("\n    3-searching ");
        printf("\n    4-Traversal in Inorder");
        printf("\n    5-Traversal in Preorder");
        printf("\n    6-Traversal in Postorder");
        printf("\n    7-Exit\n");
        printf("\n    enter your choice ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("enter element to be inserted");
                scanf("%d", &elt);
                root = insert(root, elt);
                break;

            case 2:
                printf("enter element to be deleted");
                scanf("%d",&elt);
                deletion(root,elt);
                break;

            case 3:
                printf("enter element to be search");
                scanf("%d",&elt);
                t=search(root,elt);
                if(t==NULL)
                    printf("element NOT found");
                break;

            case 4:
                printf("\n BST Traversal in INORDER \n");
                inorder(root);
                break;

            case 5:
                printf("\n BST Traversal in PREORDER \n");

```

```

        preorder(root);
        break;
    case 6:
        printf("\n BST Traversal in POSTORDER \n");
        postorder(root);
        break;
    case 7:
        printf("\n\n Terminating \n\n");
        break;
    default:
        printf("\n\nInvalid Option !!! Try Again !! \n\n");
        break;
    }
} while (ch!= 7);
}

```

7.6 INPUT/ OUTPUT

```

### Binary Search Tree Operations ###
Press 1-Creation of BST
2-deleting
3-searching
4-Traversal in Inorder
5-Traversal in Preorder
6-Traversal in Postorder
7-Exit

    enter yor choice 1
enter element to be inserted20

### Binary Search Tree Operations ###
Press 1-Creation of BST
2-deleting
3-searching
4-Traversal in Inorder
5-Traversal in Preorder
6-Traversal in Postorder
7-Exit

    enter yor choice 1
enter element to be inserted10

### Binary Search Tree Operations ###
Press 1-Creation of BST
2-deleting
3-searching
4-Traversal in Inorder
5-Traversal in Preorder
6-Traversal in Postorder
7-Exit

    enter yor choice 1
enter element to be inserted30

### Binary Search Tree Operations ###
Press 1-Creation of BST
2-deleting
3-searching
4-Traversal in Inorder
5-Traversal in Preorder
6-Traversal in Postorder
7-Exit

    enter yor choice 4

BST Traversal in INORDER
10 20 30
### Binary Search Tree Operations ###
Press 1-Creation of BST
2-deleting
3-searching
4-Traversal in Inorder
5-Traversal in Preorder
6-Traversal in Postorder

```

7.7 LAB VIVA QUESTIONS:

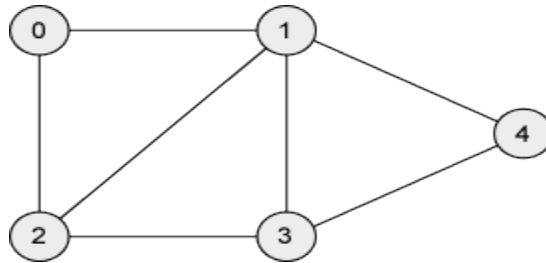
1. Define binary tree.
2. List different tree traversals.
3. Explain inorder travels with example.
4. Explain preorder travels with example.
5. Explain postorder travels with example.

WEEK-8

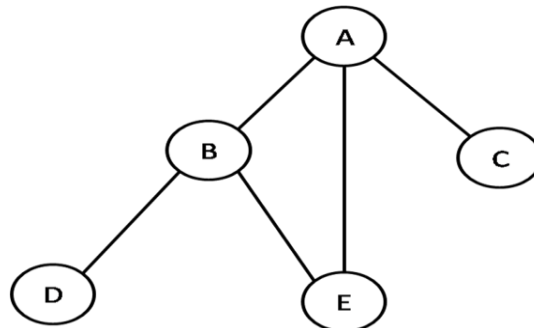
GRAPH TRAVERSALS

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:

Dev C++

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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

8.5 SOURCE CODE:

//Breadth first traversal

```
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=-1,r=0;
void bfs(int v) {
    q[++r]=v;
    visited[v]=1;
    while(f<=r) {
        for(i=1;i<=n;i++)
            if(a[v][i] && !visited[i]) {
                visited[i]=1;
                q[++r]=i;
            }
        f++;
        v=q[f];
    }
}
void main() {
    int v;
    printf("\n Enter the number of vertices:");
    scanf("%d",&n);
    for(i=1;i<=n;i++) {
        q[i]=0;
        visited[i]=0;
    }
    printf("\n Enter graph data in matrix form:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            scanf("%d",&a[i][j]);
    printf("\n Enter the starting vertex:");
    scanf("%d",&v);
    bfs(v);
}
```

```

printf("\n The node which are reachable are:\n");
for(i=1;i<=n;i++)
    if(visited[i])
        printf("%d\t",q[i]);
    else
        printf("\n Bfs is not possible");
}

```

//Checking whether a given graph is connected or not using DFS method

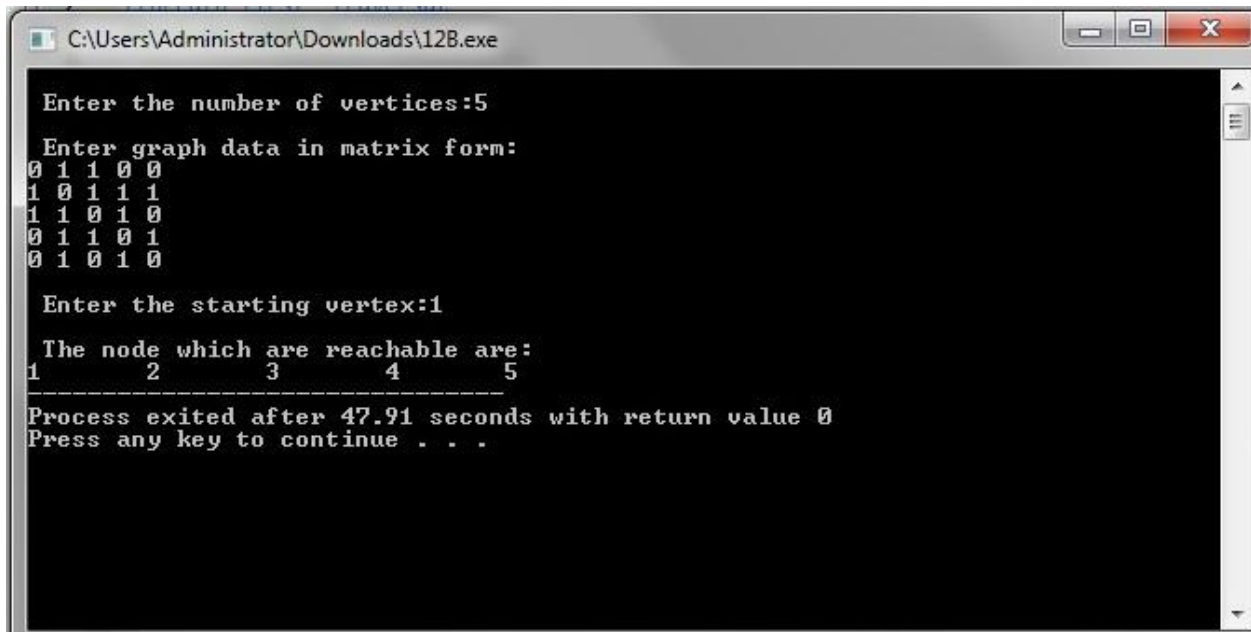
```

#include<stdio.h>
#include<conio.h>
int a[20][20],reach[20],n;
void dfs(int v) {
    int i; reach[v]=1;
    for(i=1;i<=n;i++)
        if(a[v][i] && !reach[i]) {
            printf("\n %d->%d",v,i);
            dfs(i);
        }
}
void main() {
    int i,j,count=0;
    printf("\n Enter number of vertices:");
    scanf("%d",&n);
    for(i=1;i<=n;i++) {
        reach[i]=0;
        for(j=1;j<=n;j++)
            a[i][j]=0;
    }
    printf("\n Enter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            scanf("%d",&a[i][j]);
    dfs(1);
    printf("\n");
    for(i=1;i<=n;i++) {
        if(reach[i])
            count++;
    }
    if(count==n)
        printf("\n Graph is connected");
    else
        printf("\n Graph is not connected");
}

```


8.6 INPUT/ OUTPUT

Breadth first traversal

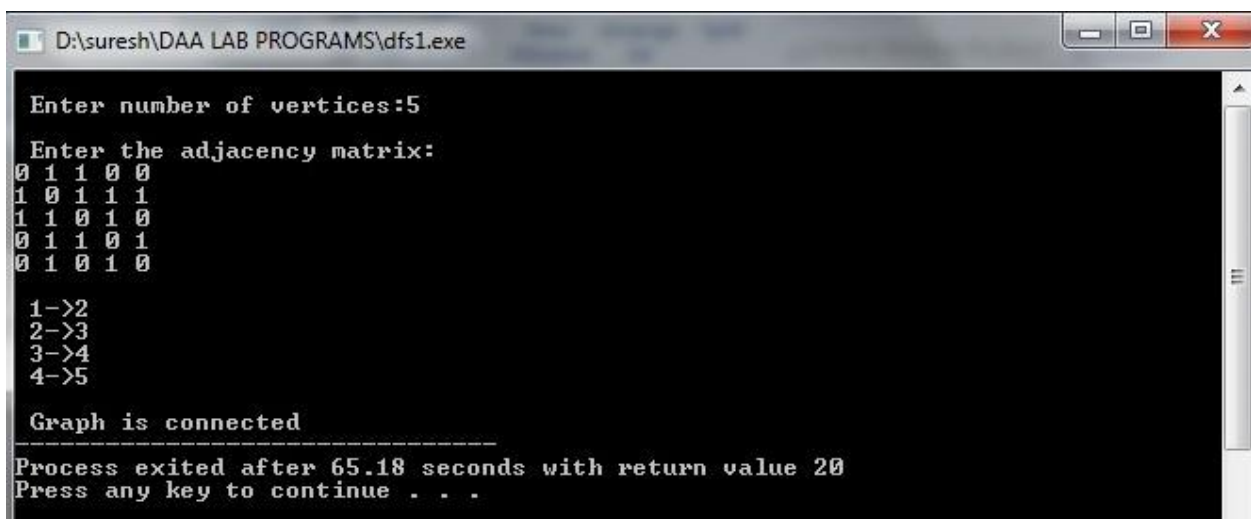


```
C:\Users\Administrator\Downloads\12B.exe

Enter the number of vertices:5
Enter graph data in matrix form:
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0

Enter the starting vertex:1
The node which are reachable are:
1      2      3      4      5
-----
Process exited after 47.91 seconds with return value 0
Press any key to continue . . .
```

Checking whether a given graph is connected or not using DFS method



```
D:\suresh\DAA LAB PROGRAMS\dfs1.exe

Enter number of vertices:5
Enter the adjacency matrix:
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0

1->2
2->3
3->4
4->5

Graph is connected
-----
Process exited after 65.18 seconds with return value 20
Press any key to continue . . .
```

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 $S = \{s_1, s_2, \dots, s_n\}$ 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:

Dev C++

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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

9.5 SOURCE CODE:

```
#include<stdio.h>
#define TRUE 1
#define FALSE 0
int inc[50],w[50],sum,n;
void sumset(int ,int ,int);
int promising(inti,intwt,int total) {
    return (((wt+total)>=sum)&&((wt==sum)||((wt+w[i+1]<=sum))));
}
void main() {
    inti,j,n,temp,total=0;
    printf("\n Enter how many numbers: ");
    scanf("%d",&n);
    printf("\n Enter %d numbers : ",n);
    for (i=0;i<n;i++) {
        scanf("%d",&w[i]);
        total+=w[i];
    }
    printf("\n Input the sum value to create sub set: ");
    scanf("%d",&sum);
    for (i=0;i<=n;i++)
        for (j=0;j<n-1;j++)
```

```

        if(w[j]>w[j+1]) {
            temp=w[j];
            w[j]=w[j+1];
            w[j+1]=temp;
        }
printf("\n The given %d numbers in ascending order: ",n);
for (i=0;i<n;i++)
    printf("%3d",w[i]);
if((total<sum))
    printf("\n Subset construction is not possible");
else {
    for (i=0;i<n;i++)
        inc[i]=0;
    printf("\n The solution using backtracking is:\n");
    sumset(-1,0,total);
}

}

void sumset(int i,int wt, int total) {
    int j;
    if(promising(i,wt,total)) {
        if(wt==sum) {
            printf("\n{ ");
            for (j=0;j<=i;j++)
                if(inc[j])
                    printf("%3d",w[j]);

            printf(" }\n");
        } else {
            inc[i+1]=TRUE;
            sumset(i+1,wt+w[i+1],total-w[i+1]);
            inc[i+1]=FALSE;
            sumset(i+1,wt,total-w[i+1]);
        }
    }
}
}

```

9.6 INPUT/ OUTPUT

```
Enter how many numbers: 5
Enter 5 numbers : 1 2 5 6 8
Input the sum value to create sub set: 9
The given 5 numbers in ascending order: 1 2 5 6 8
The solution using backtracking is:
< 1 2 6 >
< 1 8 >
```

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:

Dev C++

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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

10.5 SOURCE CODE:

```
#include<stdio.h>
int s,c[100][100],ver;
float optimum=999,sum;
/* function to swap array elements */
void swap(int v[], int i, int j) {
    int t;
    t = v[i];
    v[i] = v[j];
    v[j] = t;
}
/* recursive function to generate permutations */
void brute_force(int v[], int n, int i) {
    // this function generates the permutations of the array from element i to element n-1
    int j,sum1,k;
    //if we are at the end of the array, we have one permutation
    if (i == n) {
        if(v[0]==s) {
            for (j=0; j<n; j++)
                printf ("%d ", v[j]);
            sum1=0;
        }
    }
}
```

```

        for( k=0;k<n-1;k++) {
            sum1=sum1+c[v[k]][v[k+1]];
        }
        sum1=sum1+c[v[n-1]][s];
        printf("sum = %d\n",sum1);
        if (sum1<optimum)
            optimum=sum1;
    }
}
else
// recursively explore the permutations starting at index i going through index n-1*/
    for (j=i; j<n; j++) { /* try the array with i and j switched */
        swap (v, i, j);
        brute_force (v, n, i+1);
        /* swap them back the way they were */
        swap (v, i, j);
    }
}

void nearest_neighbour(intver) {
    int min,p,i,j,vis[20],from;
    for(i=1;i<=ver;i++)
        vis[i]=0;
    vis[s]=1;
    from=s;
    sum=0;
    for(j=1;j<ver;j++) {
        min=999;
        for(i=1;i<=ver;i++)
            if(vis[i] !=1 && c[from][i]<min && c[from][i] !=0 ) {
                min= c[from][i];
                p=i;
            }
        vis[p]=1;
        from=p;
        sum=sum+min;
    }
    sum=sum+c[from][s];
}

void main () {
    int ver,v[100],i,j;
    printf("Enter n : ");
    scanf("%d",&ver);
    for (i=0; i<ver; i++)
        v[i] = i+1;
    printf("Enter cost matrix\n");
    for(i=1;i<=ver;i++)
        for(j=1;j<=ver;j++)
            scanf("%d",&c[i][j]);
    printf("\nEnter source : ");
    scanf("%d",&s);
}

```

```

    brute_force(v, ver, 0);
    printf("\nOptimum solution with brute force technique is=%f\n", optimum);
    nearest_neighbour(ver);
    printf("\nSolution with nearest neighbour technique is=%f\n", sum);
    printf("The approximation val is=%f", ((sum/optimum)-1)*100);
    printf(" % ");
}

```

10.6 INPUT/ OUTPUT

```

D:\suresh\DAALAB PROGRAMS\traveling.exe
Enter n : 4
Enter cost matrix
0 10 15 20
5 0 9 10
6 13 0 12
8 8 9 0

Enter source : 1
1 2 3 4 sum = 39
1 2 4 3 sum = 35
1 3 2 4 sum = 46
1 3 4 2 sum = 40
1 4 3 2 sum = 47
1 4 2 3 sum = 43

Optimum solution with brute force technique is=35.000000
Solution with nearest neighbour technique is=39.000000
The approximation val is=11.428571
-----
Process exited after 59.19 seconds with return value 1
Press any key to continue . . . _

```

10.7 LAB VIVA QUESTIONS:

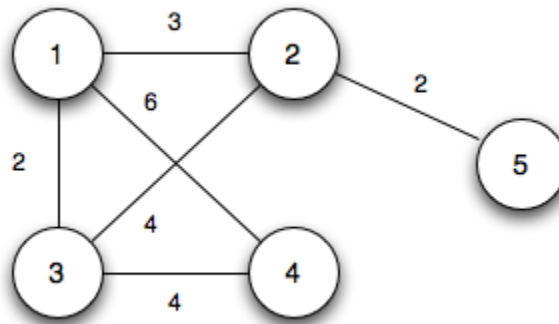
1. Define Optimal Solution.
2. Explain Travelling Sales Person Problem.
3. What is the time complexity of Travelling Sales Person Problem?

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:

Dev C++

11.3 PROGRAM LOGIC:

- 1) Create a set S that 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 S and 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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

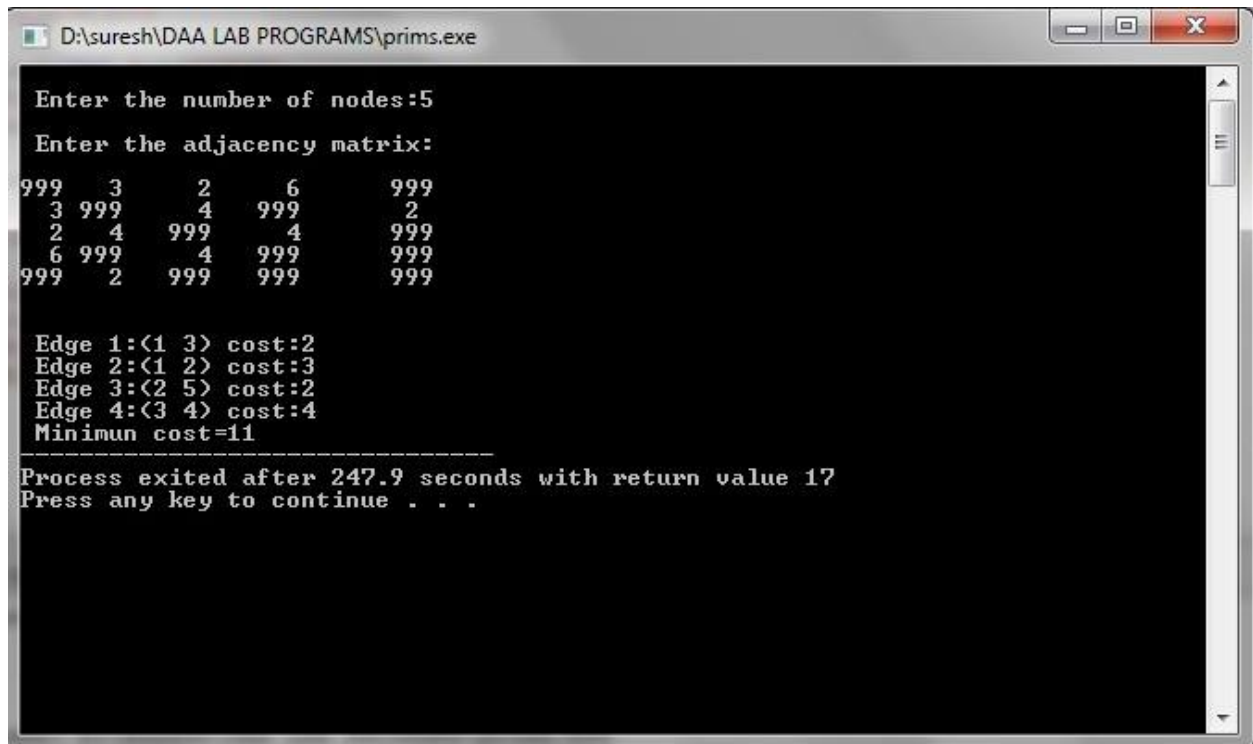
11.5 SOURCE CODE.


```

#include<stdio.h>
inta,b,u,v,n,i,j,ne=1;
int visited[10]={0},min,mincost=0,cost[10][10];
void main()
{
    printf("\n Enter the number of
nodes:"); scanf("%d",&n);
    printf("\n Enter the adjacency matrix:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++) {
            scanf("%d",&cost[i][j]);
            if(cost[i][j]==0)
                cost[i][j]=999;
        }
    visited[1]=1;
    printf("\n");
    while(ne<n)
    {
        for(i=1,min=999;i<=n;i++)
            for(j=1;j<=n;j++)
                if(cost[i][j]<min)
                    if(visited[i]!=0)
                    {
                        min=cost[i][j];
                        a=u=i;
                        b=v=j;
                    }
        if(visited[u]==0 || visited[v]==0)
        {
            printf("\n Edge %d:(%d %d)
cost:%d",ne++,a,b,min); mincost+=min;
            visited[b]=1;
        }
        cost[a][b]=cost[b][a]=999;
    }
    printf("\n Minimun cost=%d",mincost);
}

```

11.6 INPUT/ OUTPUT



```
D:\suresh\DAA LAB PROGRAMS\prims.exe

Enter the number of nodes:5
Enter the adjacency matrix:
999  3   2   6   999
 3 999   4  999   2
 2  4  999   4  999
 6 999   4  999  999
999  2  999  999  999

Edge 1:(1 3) cost:2
Edge 2:(1 2) cost:3
Edge 3:(2 5) cost:2
Edge 4:(3 4) cost:4
Minimun cost=11
-----
Process exited after 247.9 seconds with return value 17
Press any key to continue . . .
```

11.7 LAB VIVA QUESTIONS:

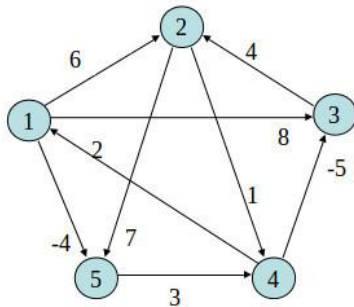
1. What is Minimum Cost spanning Tree.
2. Explain Prim's ALGORITHM.
3. What is time complexity of Prim's algorithm.

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:

Dev C++

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 idea 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 $\{0, 1, 2, \dots, k-1\}$ 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 $\text{dist}[i][j]$ as it is.
- 2) k is an intermediate vertex in shortest path from i to j . We update the value of $\text{dist}[i][j]$ as $\text{dist}[i][k] + \text{dist}[k][j]$.

12.4 PROCEDURE:

1. Create: Open Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

12.5 SOURCE CODE.

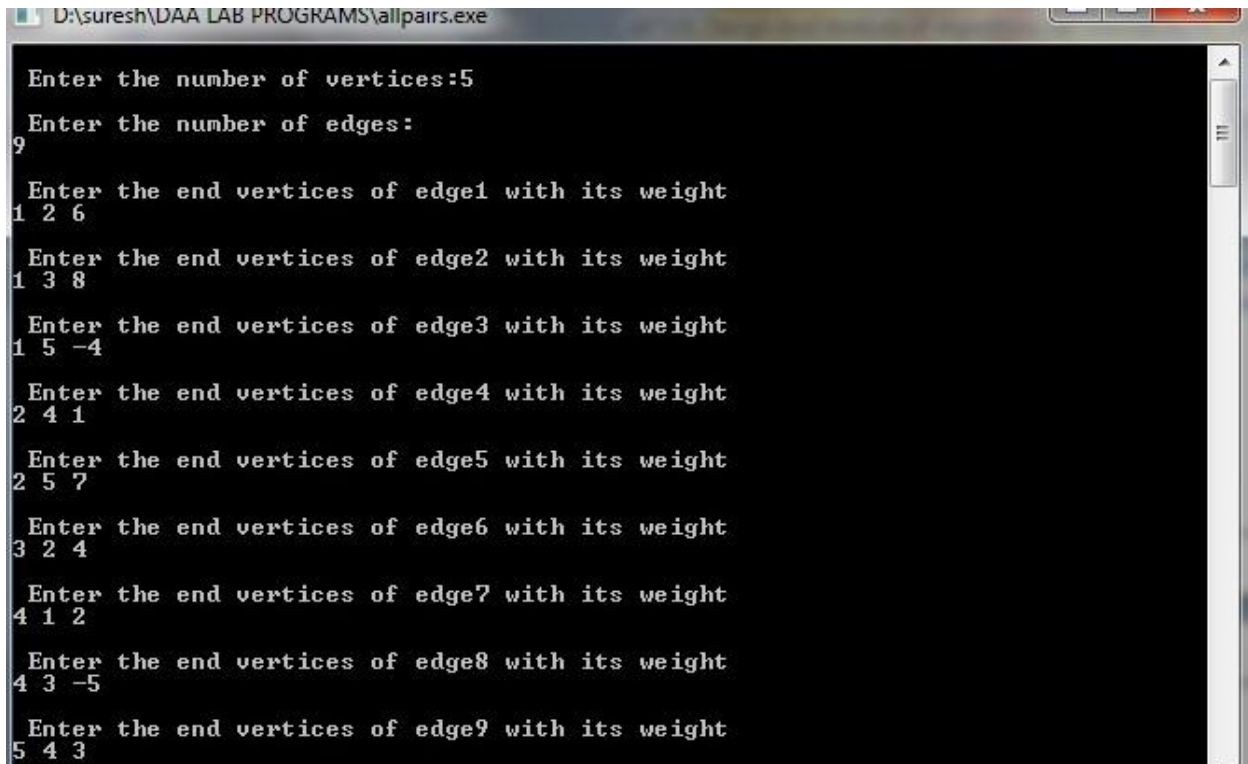
```
#include<stdio.h>
int min(int,int);
void floyds(int p[10][10],int n) {
    inti,j,k;
    for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
            for(j=1;j<=n;j++)
                if(i==j)
                    p[i][j]=0;
                else
                    p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}
int min(int a,int b) {
    if(a<b)
        return(a);
    else
        return(b);
}
main() {
    int p[10][10],w,n,e,u,v,i,j;
    printf("\n Enter the number of vertices:");
    scanf("%d",&n);
    printf("\n Enter the number of edges:\n");
    scanf("%d",&e);
    for(i=1;i<=n;i++){
        for(j=1;j<=n;j++)
            p[i][j]=999;
    }
    for(i=1;i<=e;i++) {
        printf("\n Enter the end vertices of edge%d with its weight \n",i);
        scanf("%d%d%d",&u,&v,&w);
        p[u][v]=w;
    }
    printf("\n Matrix of input data:\n");
    for(i=1;i<=n;i++) {
        for(j=1;j<=n;j++)
            printf("%d\t",p[i][j]);
        printf("\n");
    }
    floyds(p,n);
    printf("\n Transitive closure:\n");
    for(i=1;i<=n;i++) {
        for(j=1;j<=n;j++)
            printf("%d\t",p[i][j]);
        printf("\n");
    }
    printf("\n The shortest paths are:\n");
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++){
```

```

        if(i!=j)
            printf("\n <%d,%d>=%d",i,j,p[i][j]);
    }
}

```

12.6 INPUT/ OUTPUT



```

D:\suresh\DAALAB PROGRAMS\allpairs.exe
Enter the number of vertices:5
Enter the number of edges:
9
Enter the end vertices of edge1 with its weight
1 2 6
Enter the end vertices of edge2 with its weight
1 3 8
Enter the end vertices of edge3 with its weight
1 5 -4
Enter the end vertices of edge4 with its weight
2 4 1
Enter the end vertices of edge5 with its weight
2 5 7
Enter the end vertices of edge6 with its weight
3 2 4
Enter the end vertices of edge7 with its weight
4 1 2
Enter the end vertices of edge8 with its weight
4 3 -5
Enter the end vertices of edge9 with its weight
5 4 3

```

```
Matrix of input data:
999    6    8    999    -4
999   999   999    1    7
999    4    999   999   999
2     999   -5    999   999
999   999   999    3    999

Transitive closure:
0     -2    -6    -1    -4
3     0     -4    1    -1
7     4     0    5     3
2     -1    -5    0    -2
5     2     -2    3     0

The shortest paths are:
<1,2>=-2
<1,3>=-6
<1,4>=-1
<1,5>=-4
<2,1>=3
<2,3>=-4
<2,4>=1
<2,5>=-1
<3,1>=7
<3,2>=4
<3,4>=5
<3,5>=3
<4,1>=2
<4,2>=-1
<4,3>=-5
<4,5>=-2
<5,1>=5
<5,2>=2
<5,3>=-2
<5,4>=3

-----
Process exited after 91.57 seconds with return value 5
Press any key to continue . . .
```

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:

Dev C++

13.3 PROGRAM LOGIC:

- 1) Start in the leftmost column
- 2) If all queens are placed return 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 Dev C++, write a program after that save the program with .c extension.
2. Compile: Alt + F9
3. Execute: Ctrl + F10

13.5 SOURCE CODE:

```
#include<stdio.h>
#include<math.h>
int a[30],count=0;
int place(intpos) {
    int i;
    for(i=1;i<pos;i++) {
        if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
            return 0;
    }
    return 1;
}
void print_sol(int n) {
    inti,j; count++;
    printf("\n\nSolution #d:\n",count);
    for(i=1;i<=n;i++) {
        for(j=1;j<=n;j++) {
            if(a[i]==j)
```

```

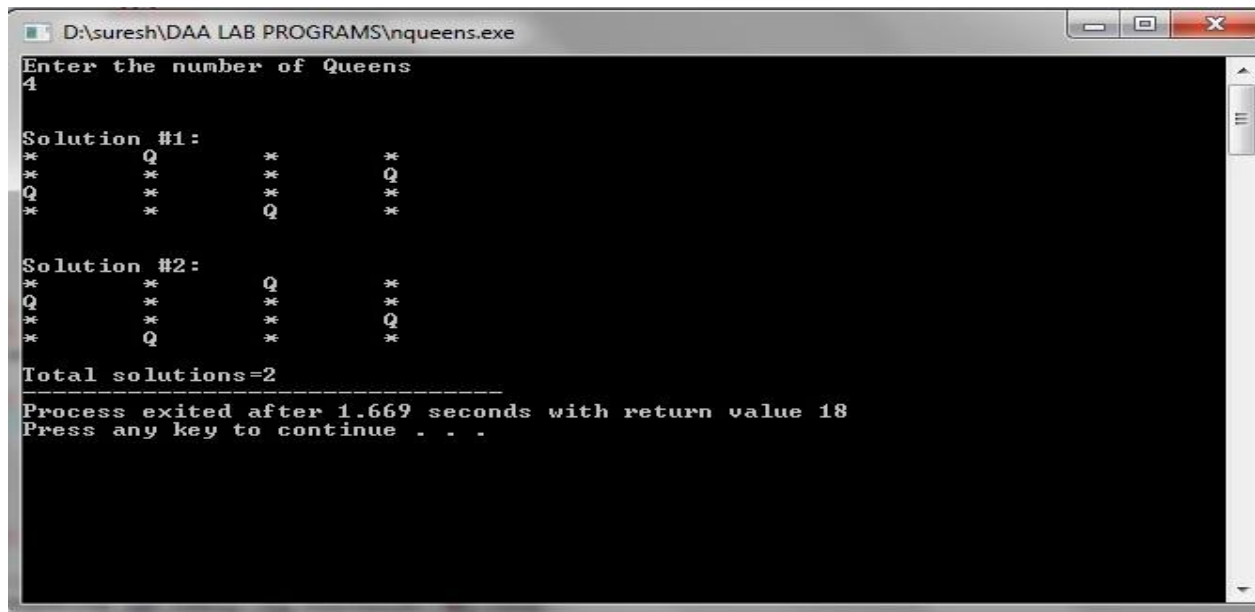
                printf("Q\t");
            else
                printf("*\t");
        }
        printf("\n");
    }
}

void queen(int n) {
    int k=1;
    a[k]=0;
    while(k!=0) {
        a[k]=a[k]+1;
        while((a[k]<=n)&&!place(k))
            a[k]++;
        if(a[k]<=n) {
            if(k==n)
                print_sol(n);
            else {
                k++;
                a[k]=0;
            }
        }
        else
            k--;
    }
}

void main() {
    inti,n;
    printf("Enter the number of Queens\n");
    scanf("%d",&n);
    queen(n);
    printf("\nTotal solutions=%d",count);
}

```


13.6 INPUT/ OUTPUT



```
D:\suresh\DAA LAB PROGRAMS\nqueens.exe
Enter the number of Queens
4

Solution #1:
*   Q   *   *
*   *   *   Q
Q   *   *   *
*   *   Q   *

Solution #2:
*   *   Q   *
Q   *   *   *
*   *   *   Q
*   Q   *   *

Total solutions=2
-----
Process exited after 1.669 seconds with return value 18
Press any key to continue . . .
```

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.