Design and Analysis of Algorithms Lab Manual for Diploma in Karnataka

RJS Polytechnic

Vinayaka VM

Design and Analysis of Algorithms

RJS POLYTECHNIC, BENGALURU

Vinayaka VM | Computer Science | August 24, 2017

Prerequisites

Knowledge of Data Structures.

Course Objectives

1. Write sorting programs using Divide-and-Conquer techniques.

2. Implement to find the minimum cost spanning tree and shortest path using different Greedy techniques.

3. Construct DFS, BFS programs and topological ordering using Decrease-and-Conquer technique.

4. Implement knapsack, travelling salesperson.

Course Outcome

On successful completion of the course, the students will be able to attain CO:

	Course Outcome	**Experiment linked**	**CL**	**Teaching Hrs**
CO1	Demonstrate Quick sort and Merge sort and calculate the time required to sort the elements.	***1,2***	***U,A,AL***	**12**
CO2	Implement the topological ordering of vertices, travelling salesman problem and Knapsack problem.	***3 to 5***	***U,A***	**18**
CO3	Construct programs to check graph is connected or not using BFS and DFS methods	***6,7***	***U,A,AL***	**15**
CO4	Implement programs on divide and conquer, decrease and conquer	***8,9***	***U,A,AL***	**15**
CO5	Experiment finding the minimum cost of spanning tree using Prim's algorithms and shortest path using Dijkstra' algorithm.	***10,11***	***U,A,AL***	**18**
			Total sessions	**78**

RJS Polytechnic, KRJS Group of Institutions, Koramangala, Bengaluru

RJS Polytechnic was established in the year 1985 - 86 by Karnataka Reddy Jana Sangha under ED 60 MPI 84, dated : 06-10-1984. RJS Polytechnic started it's journey with humble beginning with Diploma in Commercial Practice. RJS Polytechnic has made great strides in the field of technical education right from it's inception in the year 1985-86. RJS Polytechnic is situated in a serene surrounding and located in koramangala, Bangalore-34. It is located in such a place that it is able to serve the cause of Students of both urban and rural areas of Bangalore in particular and the state of Karnataka and India in general.

RJS Polytechnic is run by Karnataka Reddy Jana Sangha(KRJS) , a non-profit registered body, under self financing scheme and it consists of a galaxy of personalities from all walks of life. RJS Polytechnic is affiliated to the Directorate of Technical education, recognized by the Government of Karnataka and approved by All India Council for Technical Education (AICTE), New Delhi. The affairs of RJS Polytechnic is managed by a constituted Governing Council By KRJS.

Vinayaka V.M. , Lecturer Computer Science Department, RJS Polytechnic. Qualified with M.Tech., GATE, and KSET. Work experience in Bapuji Instiute of Engineering and Technology, Davanagere.

Table of contents Page

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.	5-8
2 Sort a given set of elements using merge sort method and determine the time required to sort the elements. Repeat the experiment for different of values of n.	9-11
3 Write a program to obtain the topological ordering of vertices in a given digraph.	12-13
4 Implement travelling salesman problem.	14-16
5 Implement the knapsack problem (0/1).	17-19
6 Print all the nodes reachable from a given starting node in a digraph using BFS method.	20-23
7 Check whether a given graph is connected or not using DFS method.	24-26
8 Write a program to implement binary search using divide and conquer technique	27-28
9 Write a program to implement insertion sort using decrease and conquer technique	29-30
10 Find minimum cost spanning tree of a given undirected path using a Prim's algorithm.	31-33
11 From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	34-36

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.

ALGORITHM *Quicksort*(A[left..right])
//Sorts a subarray by quicksort
//Input: Subarray A[0..n-1]
//Output: Subarray A[left..right] sorted in increasing order
if left < right
    split← Partition (A[left.. right]) //split is split position bisecting array A
    Quicksort(A[left.. split-1])
    Quicksort(A[split+1..right])

ALGORITHM *Partition*(A[left.. right])
//Splits a subarray using first element as a pivot
//Input: Subarray of array A[0..n-1]
//Output: Partition of A[left..right], with split position returned
pivot←A[left]
i ← left; j ← right+1
repeat
    repeat i ← i+1 until A[i]>=pivot
    repeat j ← j-1 until A[j]<=pivot
    swap(A[i],A[j])
until i>=j
swap(A[i],A[j])
swap (A[left],A[right])
return j

C code:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<time.h>
#define SIZE 10
void Quick(int A[SIZE],int,int);
int partition(int A[SIZE],int,int);
void swap(int A[SIZE],int*,int*);
int n;
int main()
{
        int i;
        int A[SIZE];
        clock_t begin,end;
        double time_required;
        clrscr();
```

```
	printf("\n Sorting using Quick sort\n");
	printf("==================================\n");
	printf("\n Enter the total number of elements:");
	scanf("%d",&n);
	for(i=0;i<n;i++)
	{
		printf("Enter the number %d:",i+1);
		scanf("%d",&A[i]);
	}
	begin=clock();
	Quick(A,0,n-1);
	end=clock();
	printf("\n==================================\n");
	printf("\n Sorted elements are:\n");
	printf("==================================\n");
	for(i=0;i<n;i++)
	printf("\t %d",A[i]);
	time_required=1.0*((double)(end-begin));
	printf("\n The time required is :%f",time_required);
	getch();
	return 0;
}
int partition(int A[SIZE],int low,int high)
{
	int pivot=A[low],i=low,j=high;
	while(i<=j)
	{
		while(A[i]<=pivot)
			i++;
		while(A[j]>pivot)
			j--;
		if(i<j)
			swap(A,&i,&j);
	}
	swap(A,&low,&j);
	return j;
}
void swap(int A[SIZE],int *i,int *j)
{
	int temp;
	temp=A[*i];
	A[*i]=A[*j];
	A[*j]=temp;
}
void Quick(int A[SIZE],int low,int high)
{
	int m,i;
	if(low<high)
	{
		m=partition(A,low,high);
		Quick(A,low,m-1);
		Quick(A,m+1,high);
	}
}
```

OUTPUT:

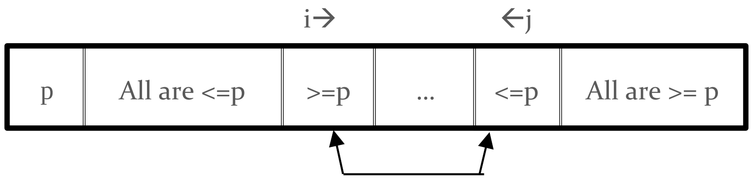
Sorting using Quick sort

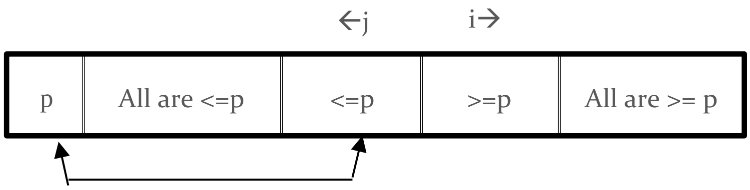
Enter the total number of elements: 5
Enter the number 1 : 2
Enter the number 2 : 6
Enter the number 3 : 4
Enter the number 4 : 9
Enter the number 5 : 1

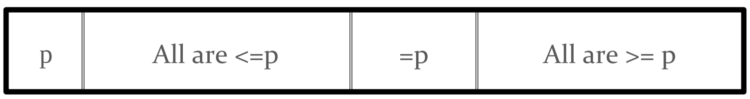
Sorted elements are :

1 3 4 6 9

The time required is : 0.000000







0	**1**	**2**	**3**	**4**
	i			j
2	6	4	9	1
		i	j	
2	1	4	9	6
		ij		
2	1	4	9	6
	j	i		
2	1	4	9	6
ij			i	j
1	2	**4**	9	6
			ij	
1	2	**4**	9	6
		j	i	
1	2	**4**	9	6
				ij
1	2	4	**9**	6
				ji
1	2	4	**9**	6
1	2	4	6	9
1	2	4	**6**	9
1	2	4	6	9

2 Sort a given set of elements using merge sort method and determine the time required to sort the elements. Repeat the experiment for different of values of n.

ALGORITHM *Mergesort*(A[0..n-1])

```
//Sorts array A[0..n-1] by recursive mergesort
//Input: An array A[0 ..n-1]
//Output: Array A[0..n-1] sorted in increasing order
if n>1
   copy A[0.. n/2 -1] to B[0..n/2 -1]
   copy A[n/2.. n-1] to C[0.. n/2 – 1]
   Mergesort(B[0 ..n/2 -1])
   Mergesort(C[0 .. n/2])
   Merge (B,C,A)
```

ALGORITHM *Merge*(B[0 ..v-1], C[0 ..m-1], A[0 ..v+m-1])

```
//Merges two sorted arrays into one sorted array
//Input: Arrays B[0..v -1] and C[0.. m-1] both sorted
//Output: Sorted array A[0..v+m-1]
i ← 0; j ← 0; k← 0
while i < v and j < m do
    if B[i]<=C[j]
        A[k] ← B[i]; i← i+1
    else A[k] ← C[j]; j← j+1
    k ← k+1
if i = v
    copy C[j..m-1] to A[k ..v+m-1]
else copy B[i..p-1] to A[k .. v+m-1]
```

C code:

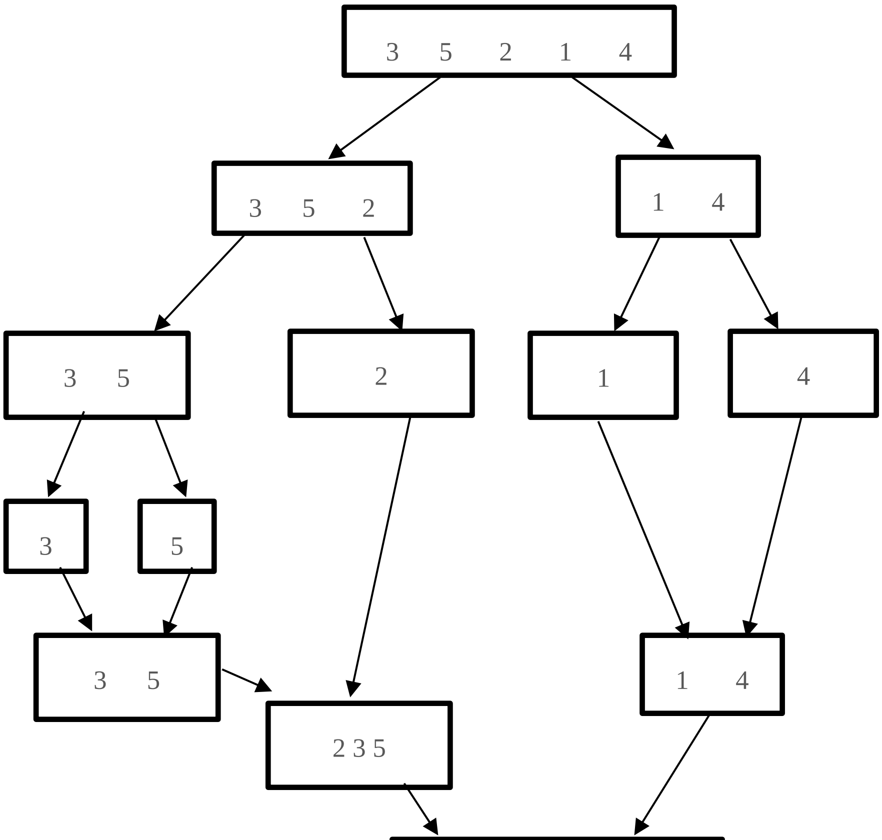
```
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int main()
{
        int i,low,high,n;
        int A[10];
        clock_t begin,end;
        double time_required;
        void MergeSort(int A[10],int low,int high);
        void Display(int A[10],int n);
        clrscr();
        printf("\n\n Sorting using merge sort\n");
        printf("\n How many elements are there?\n");
        scanf("%d",&n);
        printf("\n Enter the elements:\n");
        for(i=0;i<n;i++)
                scanf("%d",&A[i]);
```

```
	low=0;
	high=n-1;
	begin=clock();
	MergeSort(A,low,high);
	end=clock();
	Display(A,n);
	time_required=1.0*((double)(end-begin))/CLOCKS_PER_SEC;
	printf("\n The time required is:%2f",time_required);
	getch();
	return 0;
}
void MergeSort(int A[10],int low,int high)
{
	int mid;
	void combine(int A[10],int low,int mid,int high);
	if(low<high)
	{
		mid=(low+high)/2;
		MergeSort(A,low,mid);
		MergeSort(A,mid+1,high);
		combine(A,low,mid,high);
	}
}
void combine(int A[10],int low,int mid,int high)
{
	int i,j,k;
	int temp[10];
	k=low;
	i=low;
	j=mid+1;
	while(i<=mid&&j<=high)
	{
		if(A[i]<=A[j])
		{
			temp[k]=A[i];
			k++;
			i++;
		}
		else
		{
			temp[k]=A[j];
			k++;
			j++;
		}
	}
	while(i<=mid)
	{
		temp[k]=A[i];
		k++;
		i++;
	}
	while(j<=high)
	{
		temp[k]=A[j];
```

```
                k++;
                j++;
        }
        for(k=low;k<=high;k++)
                A[k]=temp[k];
}
void Display(int A[10],int n)
{
        int i;
        printf("\n\n the sorted list is....\n");
        for(i=0;i<n;i++)
                printf("%d\t",A[i]);
}
```

OUTPUT:

Sorting using merge sort
How many elements are there?
5
Enter the elements:
3 5 2 1 4
The sorted list is ….
1 2 3 4 5
The time required is : 0.000000



3 Write a program to obtain the topological ordering of vertices in a given digraph.

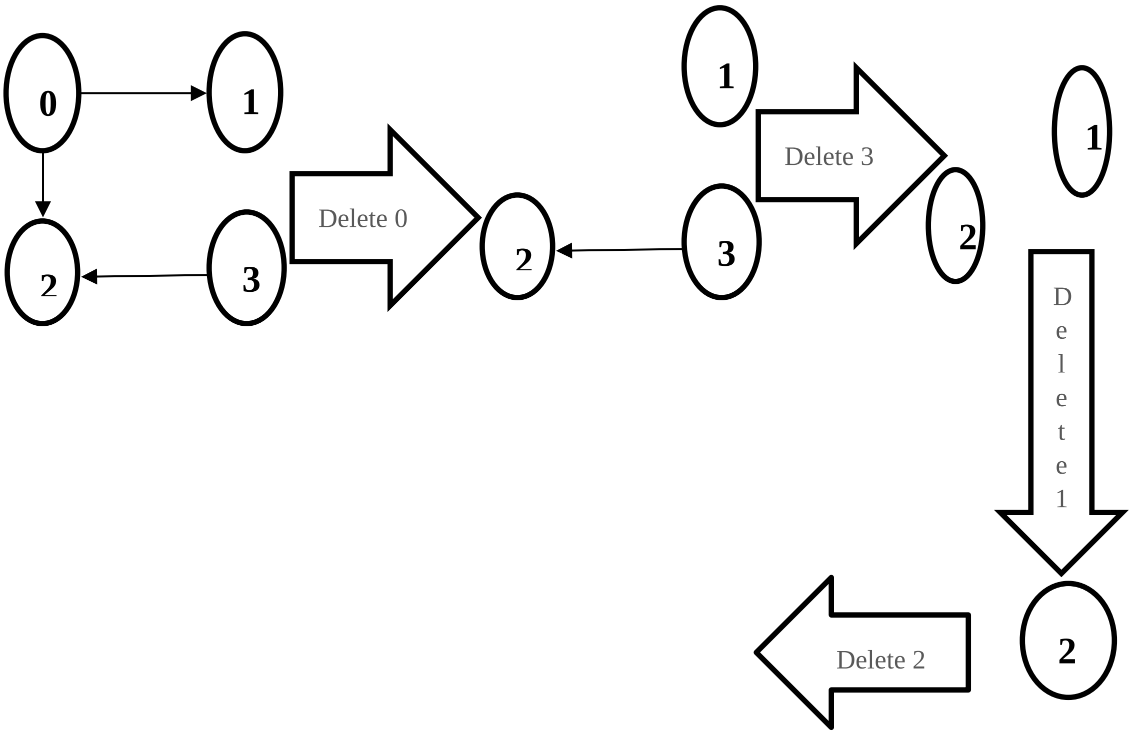
ALGORITHM Topological Sort
Step1: From a given graph find a vertex with no incoming edges.
Step2: Delete it along with all the edges outgoing from it.
Step3: Note the vertices that are deleted.
Step4: Print all these recorded vertices in the order they are deleted. This gives topologically sorted list.

```
#include<stdio.h>
#include<conio.h>
#define TRUE 1
#define FALSE 0
int main()
{
        int i,j,k;
        int n,a[10][10],in[10],visit[10];
        int count=0;
        clrscr();
        printf("\n\nTopological sorting");
        n=4;
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        a[i][j]=0;
                }
        }
        printf("\n Graph is created as follows:");
        printf("\n The node '0'and '1' are connected");
        a[0][1]=1;
        printf("\n The node '0' and '2' are connected");
        a[0][2]=1;
        printf("\n The node '3' and '2' are connected");
        a[3][2]=1;
        for(i=0;i<n;i++)
        {
                in[i]=0;
                visit[i]=FALSE;
        }
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        in[i]=in[i]+a[j][i];
        printf("\n\n The topological order is:");
        while(count<n)
        {
                   for(k=0;k<n;k++)
                   {
                        if((in [k]==0)&&(visit [k]==0))
```

```
                    {
                            printf("%d",k);
                            visit[k]=TRUE;
                    }
                    for(i=0;i<n;i++)
                    {
                            if(a[i][k]==1)
                                    in[k]--;
                    }
                }
            count++;
        }
        getch();
        return 0;
    }
```

OUTPUT:

Topological sorting
Graph is created as follows:
The node ‘0’ and ‘1’ are connected
The node ‘0’ and ‘2’ are connected
The node ‘3’ and ‘2’ are connected
The topological order is : 0312



4 Implement travelling salesman problem.

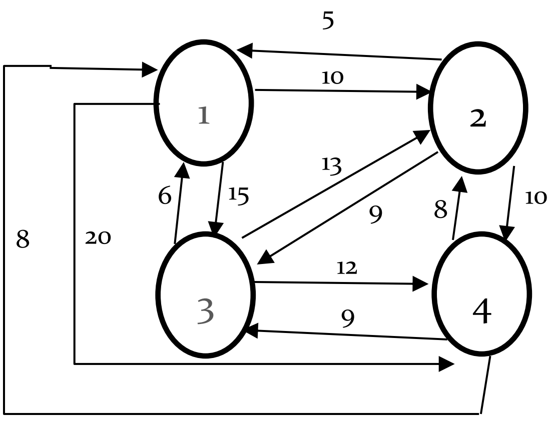
ALGORITHM Travelling Salesman
Step1: Generate a set of all the tours starting from and ending at city ‘1’.
Step2: Calculate length of all the tours generated in step1.
Step3: Select the minimum length tour as the optimal solution.
If more than one such tours exist, break the tie randomly.

```
#include<stdio.h>
#include<conio.h>
#define MAX 10
typedef struct
{
	int nodes[MAX];
	int vertex;
	int min;
}path_node;
path_node TSP(int source,path_node list,int Element[] [MAX],int max_no_cities)
{
	int i,j;
	path_node new_list,new_path,new_min;
	if(list.vertex==0)
	{
		new_min.min=Element[source][1];
		new_min.nodes[max_no_cities-1]=source;
		new_min.vertex=max_no_cities;
		return new_min;
	}
	for(i=0;i<list.vertex;i++)
	{
		new_list.vertex=0;
		for(j=0;j<list.vertex;j++)
			if(i!=j)
				new_list.nodes[new_list.vertex++]=list.nodes[j];
		new_path=TSP(list.nodes[i],new_list,Element,max_no_cities);
		new_path.min=Element[source][list.nodes[i]]+new_path.min;
		new_path.nodes[max_no_cities-list.vertex -1]=source;
		if(i==0)
			new_min=new_path;
		else if(new_path.min<new_min.min)
			new_min=new_path;
	}
	return new_min;
}
void display(path_node path)
{
	int i;
	printf("\n\n the minimum costis%d\n",path.min);
	printf("\n The path is.....\n");
	for(i=0;i<path.vertex;i++)
		printf("%d__",path.nodes[i]);
```

```
	printf("%d",path.nodes[0]);
}
main()
{
	int i,j,Element[MAX][MAX],max_no_cities;
	path_node graph,Path;
	clrscr();
	printf("\n How many numbers of cities are there?");
	scanf("%d",&max_no_cities);
	if(max_no_cities==0)
	{
		printf("error:There is no city for processing the TSP");
	}
	else
	{
		for(i=1;i<=max_no_cities;i++)
		{
			for(j=1;j<=max_no_cities;j++)
				if(i==j)
					Element[i][i]=0;
				else
				{
					printf("enter distance from city %d to %d?",i,j);
					scanf("%d",&Element[i][j]);
				}
			if(i>1)
				graph.nodes[i-2]=i;
		}
		graph.vertex=max_no_cities-1;
		Path=TSP(1,graph,Element,max_no_cities);
		display(Path);
	}
	getch();
	return 1;
}
```

OUTPUT:

How many number of cities are there? 4
enter the distance from city 1 to 2? 10
enter the distance from city 1 to 3? 15
enter the distance from city 1 to 4? 20
enter the distance from city 2 to 1? 5
enter the distance from city 2 to 3? 9
enter the distance from city 2 to 4? 10
enter the distance from city 3 to 1? 6
enter the distance from city 3 to 2? 13
enter the distance from city 3 to 4? 12
enter the distance from city 4 to 1? 8
enter the distance from city 4 to 2? 8
enter the distance from city 4 to 3? 9
the minimum cost is 35
The path is …..
1_2_4_3_1



TOUR	LENGTH
1→2→3→4→1	l = 10+9+12+8 = 39
1→3→2→4→1	l = 15+13+10+8 = 46
1→4→2→3→1	l = 20+8+9+6 = 43
1→4→3→2→1	l = 20+9+13+5 = 47
1→2→4→3→1	l = 10+10+9+6 = 35 optimal
1→3→4→2→1	l = 15+12+8+5 = 40

5 Implement the knapsack problem (0/1).

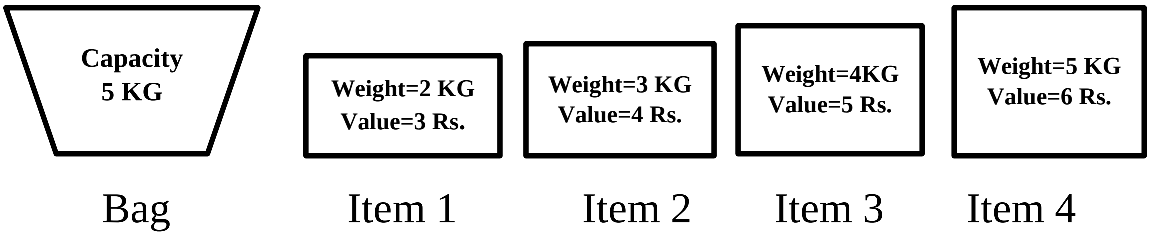
ALGORITHM Knapsack
Step1: Generate all the combinations of given items in a subset.
Step2: Calculate total weight of all the subsets generated in step1.
Step3: Eliminate the subsets whose total weight exceeds the capacity of knapsack.
Step4: Calculate total value of items in each subset that is remaining.
Step5: Select the subset which has highest total value as the feasible solution.
If there are more than one such subsets, break the tie randomly.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int table[5][6];

void main()
{
  int w[ ]={0,2,3,4,5};
  int v[ ]={0,3,4,5,6};
  int W=5;
  int n=4;
  int i,j;
  void DKP(int n, int W, int w[], int v[]);
  clrscr();
  printf("\n\t\t 0/1 Knapsack Problem");
  for (i=0;i<=n;i++)
  {
        for (j=0;j<=W;j++)
        {
                table[i][j]=0;
        }
  }
  DKP(n,W,w,v);
  getch();
}

void DKP(int n, int W,int w[5], int v[5])
{
  void find_item(int ,int, int[]);
  int i,j;
  int val1,val2;
  for(i=0;i<=n;i++)
  {
    for(j=0;j<=W;j++)
    {
      table[i][0]=0;
      table[0][j]=0;
    }
  }
  for(i=1;i<=n;i++)
  {
   for(j=1;j<=W;j++)
```

```
    {
     if(j<w[i])
          table[i][j]=table[i-1][j];
     else if (j>=w[i])
     {
           val1=table[i-1][j];
           val2=v[i]+table[i-1][j-w[i]];
           table[i][j]=(val1>val2?val1:val2);
    }
   }
  }
  printf("\n Table constructed \n");
  for (i=0;i<=n;i++)
  {
    for(j=0;j<=W;j++)
         printf("%d  ",table[i][j]);
    printf("\n");
  }
find_item(n,W,w);
}
void find_item(int i,int k,int w[5])
{
  printf("\nFor the Knapsack...");
  while(i>0 && k>0)
  {
    if (table[i][k]!=table[i-1][k])
    {
         printf("\nItem %d is selected",i);
         i=i-1;
         k=k-w[i];
    }
    else
         i=i-1;
  }
}
```

OUTPUT:

```
 0/1 Knapsack Problem
Table constructed
0 0 0 0 0 0
0 0 3 3 3 3
0 0 3 4 4 7
0 0 3 4 5 7
0 0 3 4 5 7
For the Knapsack…
Item 2 is selected
Item 1 is selected
```



Subset	Total weight	Total value
Null	**0**	**0**
{1}	**2**	**3**
{2}	**3**	**4**
{3}	**4**	**5**
{4}	**5**	**6**
{1,2}	**2+3=5**	**3+4=7 OPTIMAL**
{1,3}	2+4=6	3+5=8
{1,4}	2+5=7	3+6=9
{2,3}	3+4=7	4+5=9
{2,4}	3+5=8	4+6=10
{3,4}	4+5=9	5+6=11
{1,2,3}	2+3+4=9	3+4+5=12
{1,2,4}	2+3+5=10	3+4+6=13
{1,3,4}	2+4+5=11	3+5+6=14
{2,3,4}	3+4+5=12	4+5+6=15
{1,2,3,4}	2+3+4+5=14	3+4+5+6=18

CAPACITY OF BAG IS **5 KG**;

So in the subset of items less than or equal to 5 KG {1,2} has highest value. Hence we select item 1 and item 2.

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

ALGORITHM *BFS*(G)
// Implements a breadth first search traversal of a given graph
// Input: Graph G=<VERT,EDG>
//Output: Graph G with its verticies marked with consecutive integers in the order they are visited by the BFS traversal
mark each vertex VERT with 0 as a mark of being "not visited"
count← 0
for each vertex v in VERT do
  if v is marked with 0
    bfs(v)

bfs(v)
//visit all the unvisited vertices connected to vertex v by a path and numbers them in the order they are visited via global variable *count*
count← *count* +1; mark v with *count* and initialize a queue with v
while the queue is not empty do
  for each vertex w in VERT adjacent to the front vertex do
    if w is marked with 0
      count ← count +1; mark w with count
      add w to the queue
  remove the front vertex from the queue

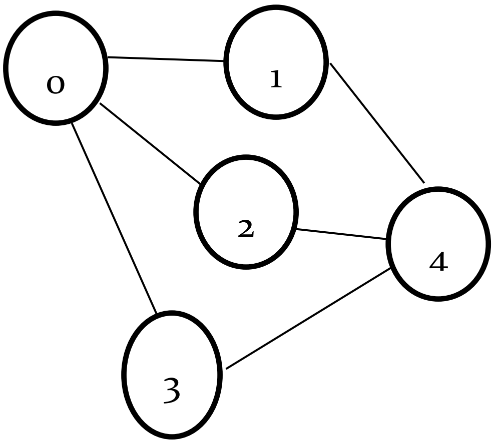
C code:

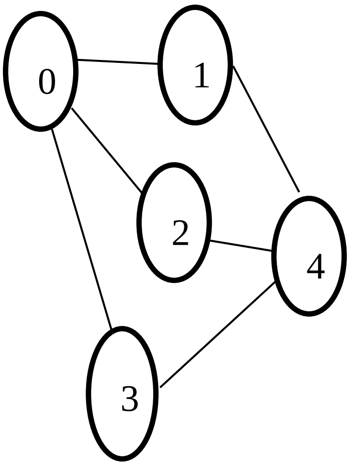
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size 20
#define TRUE 1
#define FALSE 0
int g[size][size];
int visit[size];
int q[size];
int front,rear;
int n;
void main()
{
	int v1,v2;
	char ans;
	void create(),bfs(int v1);
	ans='y';
	clrscr();
	create();
	clrscr();
	printf("The adjacency matrix for the graph is\n");
	for(v1=0;v1<n;v1++)
	{
		for(v2=0;v2<n;v2++)
```

```
                printf("%d",g[v1][v2]);
            printf("\n");
        }
        getch();
        do
        {
            for(v1=0;v1<n;v1++)
                visit[v1]=FALSE;
            clrscr();
            printf("Entre the vertex from which you want to traverse:");
            scanf("%d",&v1);
            if(v1>=n)
                printf("invalid vertex\n");
            else
            {
                printf("The breath first search of the graph is\n");
                bfs(v1);
                getch();
            }
            printf("\n Do you want to traverse from any other node?");
            ans=getche();
        }while(ans=='y');
        exit(0);
}
void create()
{
        int v1,v2;
        char ans;
        ans='y';
        printf("\n This is a program to creat a graph");
        printf("\n The display is in breadth first manner");
        printf("\n Entre number of nodes: ");
        scanf("%d",&n);
        for(v1=0;v1<n;v1++)
            for(v2=0;v2<n;v2++)
                g[v1][v2]=FALSE;
        printf("\n Enter the vertices no starting from 0;");
        do
        {
            printf("\n Enter the vertices v1 and v2:");
            scanf("%d%d",&v1,&v2);
            if(v1>=n||v2>=n)
                printf("invalid vertex value\n");
            else
            {
                g[v1][v2]=TRUE;
                g[v2][v1]=TRUE;
            }
        printf("\n\n Add more edges??(y/n)");
        ans=getche();
        }while(ans=='y');
}
void bfs(int v1)
{
```

```
        int v2;
        visit[v1]=TRUE;
        front=rear=-1;
        q[++rear]=v1;
        while(front!=rear)
        {
                v1=q[++front];
                printf("%d\n",v1);
                for(v2=0;v2<n;v2++)
                {
                        if(g[v1][v2]=TRUE&&visit[v2]==FALSE)
                        {
                                q[++rear]=v2;
                                visit[v2]=TRUE;
                        }
                }
}
```

OUTPUT:
This is a program to create a graph
The display is in breadth first manner
Enter no of nodes: 5
Enter the vertices no starting from 0:
Enter the vertices v1 and v2: 0 1
Add more edges??(y/n) y
Enter the vertices v1 and v2 : 0 2
Add more edges??(y/n) y
Enter the vertices v1 and v2 : 0 3
Add more edges??(y/n) y
Enter the vertices v1 and v2 : 1 4
Add more edges??(y/n) y
Enter the vertices v1 and v2 : 2 4
Add more edges??(y/n) y
Enter the vertices v1 and v2 : 3 4
Add more edges??(y/n) n
The adjacency matrix for the graph is
01110
10001
10001
10001
01110
Enter the vertex from which you want to traverse: 0
The breadth first search of the graph is
0
1
2
3
4

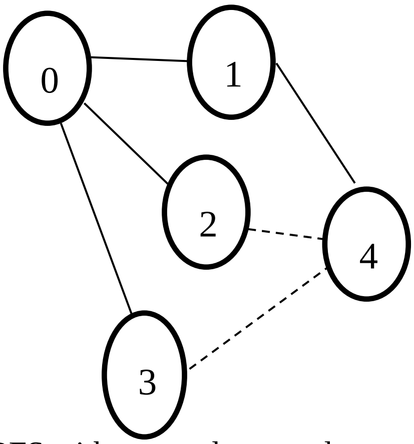




Graph

$0_1\ 1_2\ 2_3\ 3_4\ 4_5$

Traversal Sequence



BFS with tree and cross edges
Shown in solid and dotted line

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

ALGORITHM *DFS*(G)
//Implements a depth first search traversal of a given graph
// Input: Graph G = <VERT, EDG>
// Output: Graph G with its vertices marked with consecutive integers in the order they are first encountered by the DFS traversal
mark each vertex in VERT with 0 as a mark of being "not visited"
count ← 0
for each vertex v in VERT do
  if v is markded with 0
    dfs(v)

dfs(v)
//vistis recursively all the unvited vertices connected to vertex v by a path and numbers them in the order they are encountered via a global variable *count*
count ← *count* +1; mark v with *count*
for each vertex w in VERT adjacent to v do
  if w is marked with 0
    dfs(w)

C code:

```
#include        <stdio.h>
#include        <conio.h>
int dfs ( int source );
int ab = 0, i, j, n, m, a[10][10], vis[10],source, num = 1;
void main( )
{
        clrscr();
        printf ( "\nEnter the Number of Vertices: " );
        scanf ( "%d", &n );
        printf ( "\nEnter the Adjacency Matrix\n" );
        for ( i = 1; i <= n; i ++)
                for ( j = 1; j <= n; j ++ )
                        scanf("%d",&a[i][j]);
        printf ( "\nEnter the Starting Vertex: " );
        scanf ( "%d", &source );
        for ( i = 1; i <= n; i ++)
                vis[i] = 1;
        dfs ( source );
        if ( ab == ( n - 1 ) )
                printf ( "Connected Graph\n" );
        else
                printf ( "Unconnected Graph\n" );
        getch();
```

```
}

int dfs ( int source )
{
        vis[source] = 1;
        for ( i = 1; i <= n; i ++)
                if ( a[source][i] && vis[i] == 0 )
                {
                        ab ++;
                        dfs ( i );
                }
        return 0;
}
```

OUTPUT 1:

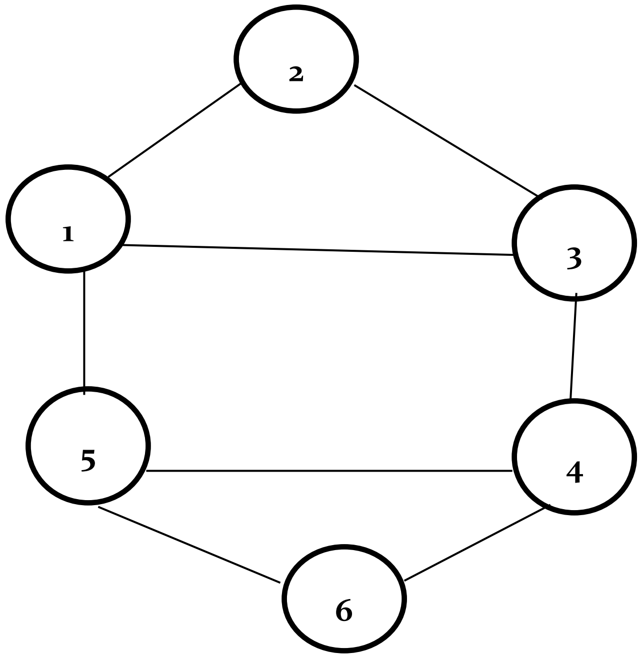
Enter the Number of Vertices:6

Enter the Adjacency Matrix

0 1 1 0 1 0
1 0 1 0 0 0
1 1 0 1 0 0
0 0 1 0 1 1
1 0 0 1 0 1
0 0 0 1 1 0

Enter the Starting Vertex:1

Connected Graph



OUTPUT 2:

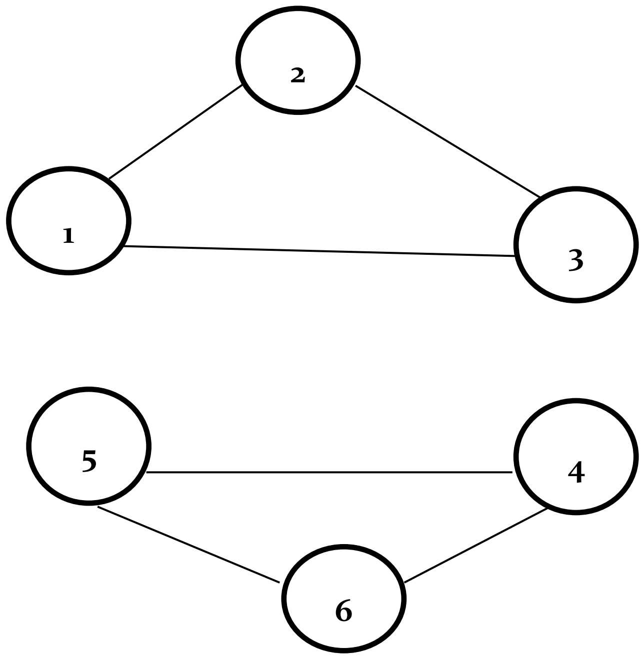
Enter the Number of Vertices:6

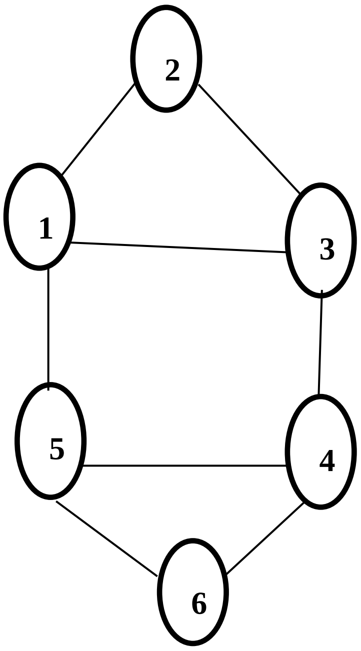
Enter the Adjacency Matrix

0 1 1 0 0 0
1 0 1 0 0 0
1 1 0 0 0 0
0 0 0 0 1 1
0 0 0 1 0 1
0 0 0 1 1 0

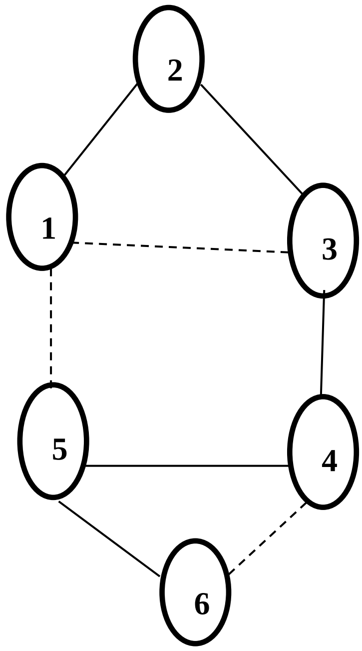
Enter the Starting Vertex:1

Unconnected Graph

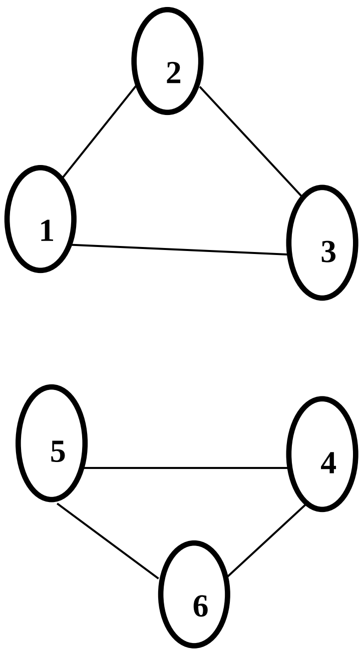




$1_1\ 2_2\ 3_3\ 4_4\ 5_5\ 6_6$

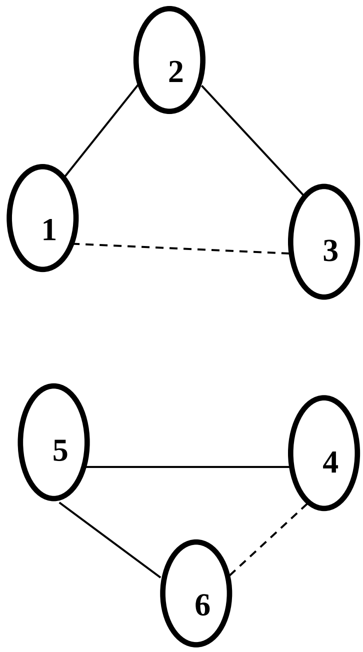


Graph | Traversal Sequence | DFS with tree and back edges shown in solid and dotted lines respectively



$1_1\ 2_2\ 3_3$

$4_1\ 5_2\ 6_3$



Graph | Traversal Sequence | DFS forest with tree and back edges Shown in solid and dotted lines respectively

8 Write a program to implement binary search using divide and conquer technique

ALGORTIHM *BinSearch*(A[0 .. n-1], KEY)
// Implements nonrecursive binay search
// Input: An array A[0 ..n-1] sorted in ascending order and a search key KEY
// Output: An index of the array's element that is equal to KEY or -1 if there is no such element
left ← 0; right ← n-1
while left <= right do
   middle ← (left + right) /2
   if KEY =A[middle] return middle
   else if KEY < A [middle] right ← middle -1
   else left ← middle +1
return -1

C Code:

```
#include <stdio.h>
#include <conio.h>
#define SIZE 10
int n;
void main()
{
	int A[SIZE],KEY,i,flag;
	int BinSearch(int A[SIZE],int KEY);
	clrscr();
	printf("\nHow many elements in an array?");
	scanf("%d",&n);
	printf("\nEnter the elements");
	for (i=0;i<n;i++)
		scanf("%d",&A[i]);
	printf("Enter the element which is to be searched:");
	scanf("%d",&KEY);
	flag=BinSearch(A,KEY);
	if (flag==-1)
		printf("\nThe Element is not present");
	else
		printf("\nThe element is at A[%d] location",flag);
	getch();
}
int BinSearch(int A[SIZE],int KEY)
{
	int low,high,m;
	low=0;
	high=n-1;
	while(low<=high)
	{
		m=(low+high)/2;
		if(KEY==A[m])
			return m;
		else if (KEY<A[m])
```

```
                    high=m-1;
            else
                    low=m+1;
        }
        return -1;
}
```

OUTPUT:
Case 1:
How many elements in an array? 13
Enter the elements
3 14 27 31 39 42 55 70 74 81 85 93 98
Enter the element which is to be searched: 70
The element is at A[7] location

Case 2:
How many elements in an array? 5
Enter the elements
10 20 30 40 50
Enter the element which is to be searched: 80
The element is not present

$$\underbrace{A[0]\dots A[m-1]}_{\substack{\text{search here if}\\ K<A[m]}} \quad \overset{\overset{K}{\updownarrow}}{A[m]} \quad \underbrace{A[m+1]\dots A[n-1]}_{\substack{\text{search here if}\\ K>A[m]}}.$$

As an example, let us apply binary search to searching for $K = 70$ in the array

3	14	27	31	39	42	55	70	74	81	85	93	98

The iterations of the algorithm are given in the following table:

index	0	1	2	3	4	5	6	7	8	9	10	11	12
value	3	14	27	31	39	42	55	70	74	81	85	93	98
iteration 1	*l*						*m*						*r*
iteration 2								*l*		*m*			*r*
iteration 3								*l,m*	*r*				

9 Write a program to implement insertion sort using decrease and conquer technique

ALGORITHM *InsertionSort*(A[0..n-1])

```
//Sorts a given array by insertion sort
//Input: An array A[0..n-1] of n orderable elements
//Output: Array A[0..n-1] sorted in increasing order
For i←1 to n-1 do
    v←A[i]
    j←i-1
    while j>=0 and A[j]> v do
            A[j+1] ← A[j]
            j ← j-1
   A[j+1] ← v
```

C code:

```
#include <stdio.h>
#include <conio.h>
void main()
{
        int A[10],n,i;
        void Insert_sort(int A[10],int n);
        clrscr();
        printf ("\nInsertion Sort");
        printf("\nHow many elements are there?");
        scanf("%d",&n);
        printf("\nEnter the elements\n");
        for (i=0;i<n;i++)
                scanf("%d",&A[i]);
        Insert_sort(A,n);
        getch();
}
void Insert_sort(int A[10],int n)
{
        int i,j,temp;
        for(i=1;i<=n-1;i++)
        {
                temp=A[i];
                j=i-1;
                while((j>=0)&&(A[j]>temp))
                {
                        A[j+1]=A[j];
                        j=j-1;
                }
                A[j+1]=temp;
        }
        printf("\nThe sorted list of elements is \n");
        for (i=0;i<n;i++)
                printf("\n%d",A[i]);
}
```

OUTPUT:
Insertion Sort
How many elements are there? 5
Enter the elements
30 20 10 40 50
The sorted list of elements is
10
20
30
40
50

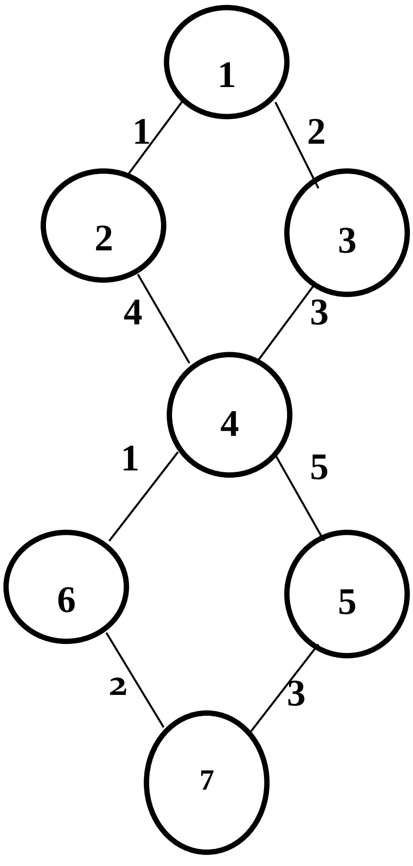
A [0] < = … < = A [j] < A[j+1] <= … <= A[i-1] | A[i] … A[n-1]
Smaller than or equal to A[i] greater than A[i]

Fig: Iteration of insertion sort:
A[i] is inserted in its proper position among the preceding elements previously sorted

30 | **20** 10 40 50
20 30 | **10** 40 50
10 20 30 | **40** 50
10 20 30 40 | **50**
10 20 30 40 50

10 Find minimum cost spanning tree of a given undirected path using a Prim's algorithm.

ALGORITHM *Prim*(G)
//Prim's algorithm for constructing a minimum cost spanning tree
//Input: A weighted connected graph G=<V,E>
//Output: E_T, the set of edges composing a minimum spannig tree of G
$V_T \leftarrow \{v_0\}$ // the set of tree vertices can be initialized with any vertex
$E_T \leftarrow$ NULL
for i$\leftarrow$ 1 to |V|-1 do
    find a minimum weight edge e*=(v*,u*) among all the edges (v,u)
    such that v is in V_T and u is in V - V_T
    $V_T \leftarrow V_T$ U {u*}
    $E_T \leftarrow E_T$ U {e*}
return E_T



GRAPH:

C code:

```
#include<stdio.h>
int s[10], min, ne=1, minCost=0, cost[10][10];
int i, j, a, b, u, v, n;
void main()
{
        clrscr();
        printf("Enter The Number of Vertices: ");
        scanf("%d",&n);
        printf("Enter 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;
                }
        for(i=2;i<=n;i++)
                s[i]=0;
        s[1]=1;
        while(ne<=n)
        {
                for(i=1,min=999;i<=n;i++)
                        for(j=1;j<=n;j++)
                        if(cost[i][j]<min)
                                if(s[i]==0)continue;
                        else
                        {
                                min=cost[i][j];
                                a=u=i;
                                b=v=j;
                        }
```

```
                if(s[u]==0 || s[v]==0)
                {
                        printf("\nEdge[%d %d] : %d",a,b,min);
                        minCost+=min;
                        s[b]=1;
                }
                ne++;
                cost[a][b]=cost[b][a]=999;
        }
        printf("\nMin Cost: %d",minCost);
getch();
}
```

OUTPUT:

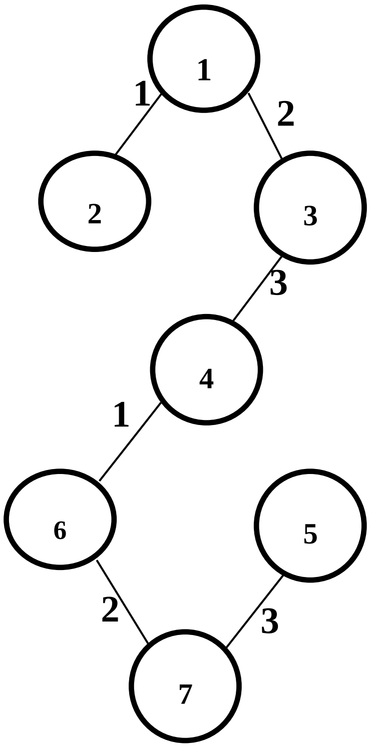
Enter Number of Vertices:7
Enter the Cost Adjacency Matrix :

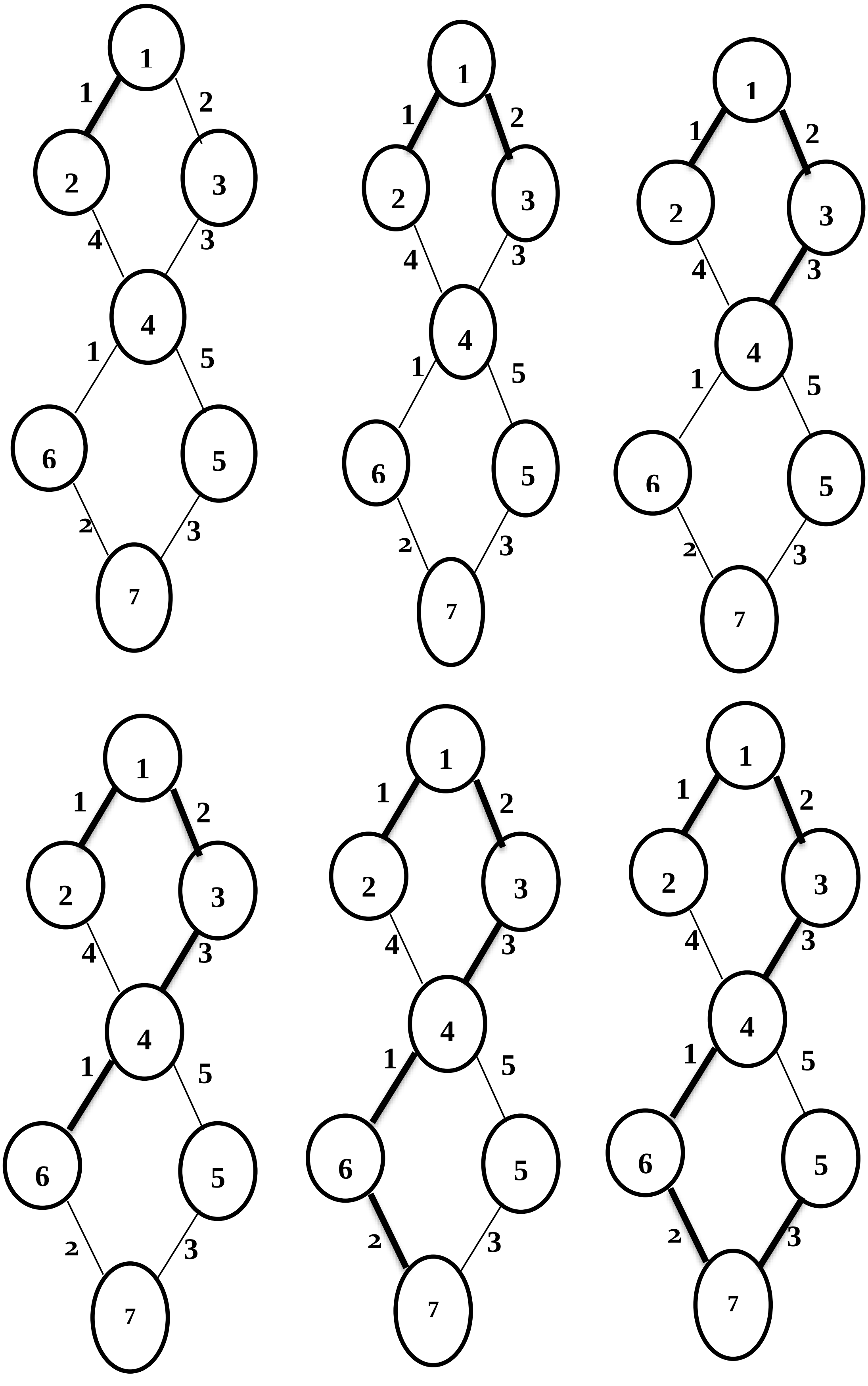
0	1	2	0	0	0	0
1	0	0	4	0	0	0
2	0	0	3	0	0	0
0	4	3	0	5	1	0
0	0	0	5	0	0	3
0	0	0	1	0	0	2
0	0	0	0	3	2	0

Edge(1 2) : 1
Edge(1 3) : 2
Edge(3 4) : 3
Edge(4 6) : 1
Edge(6 7) : 2
Edge(7 5) : 3

Min Cost: 12

Minimum Cost
Spanning Tree:





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

ALGORITHM *Dijkstra*(G,s)
//Dijkstra's algorithm for single source shortest paths
//Input: A weighted connected graph G = <V,E> with nonnegative weights and its vertex s
//Output: The length d_v of a shortest path from s to v and its penultimate vertex p_v for every vertex v in V
Initialize(Q) // initialize vertex priority queue to empty
For every vertex v in V
  d_v ← infinite; p_v ← null
  Insert (Q,v,d_v) // Initialize vertex priority in the priority queue
D_s ← 0; Decrease(Q,s,d_s) // update priority of s with d_s
V_T ← null
for i← 0 to |V|-1 do
  u* ← DeleteMin(Q) // delete the minimum priority element
  V_T ← V_T U {u*}
  for every vertex u in V – V_T that is adjacent to u* do
    if d_{u*}+w(u*,u); p_u ← u*
      Decrease (Q,u,d_u)

C code:

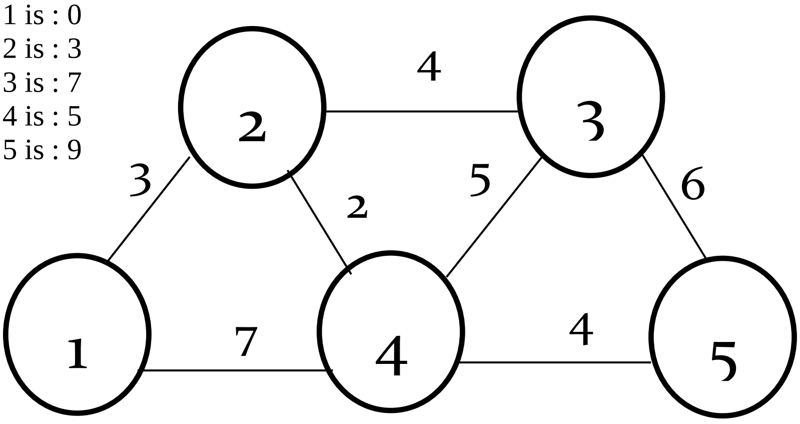
```
#include	<stdio.h>
#include	<conio.h>
int d[20], a[10][10], n;
void dijk(int ,int []);
void main()
{
	int i, j, s;
	clrscr();
	printf("Enter the Number of Vertices: ");
	scanf("%d",&n);
	printf("Enter Cost Adjacent Matrix:\n");
	for(i=1;i<=n;i++)
		for(j=1;j<=n;j++)
			scanf("%d",&a[i][j]);
	printf("Enter the Starting Vertex: ");
	scanf("%d",&s);
	dijk(s,d);
	for(i=1;i<=n;i++)
		printf("The Shortest path from %d to %d is: %d\n",s,i,d[i]);
	getch();
}

void dijk(int v, int d[20])
{
	int i, j, min, s[20], u, w;
	for(i=1;i<=n;i++)
```

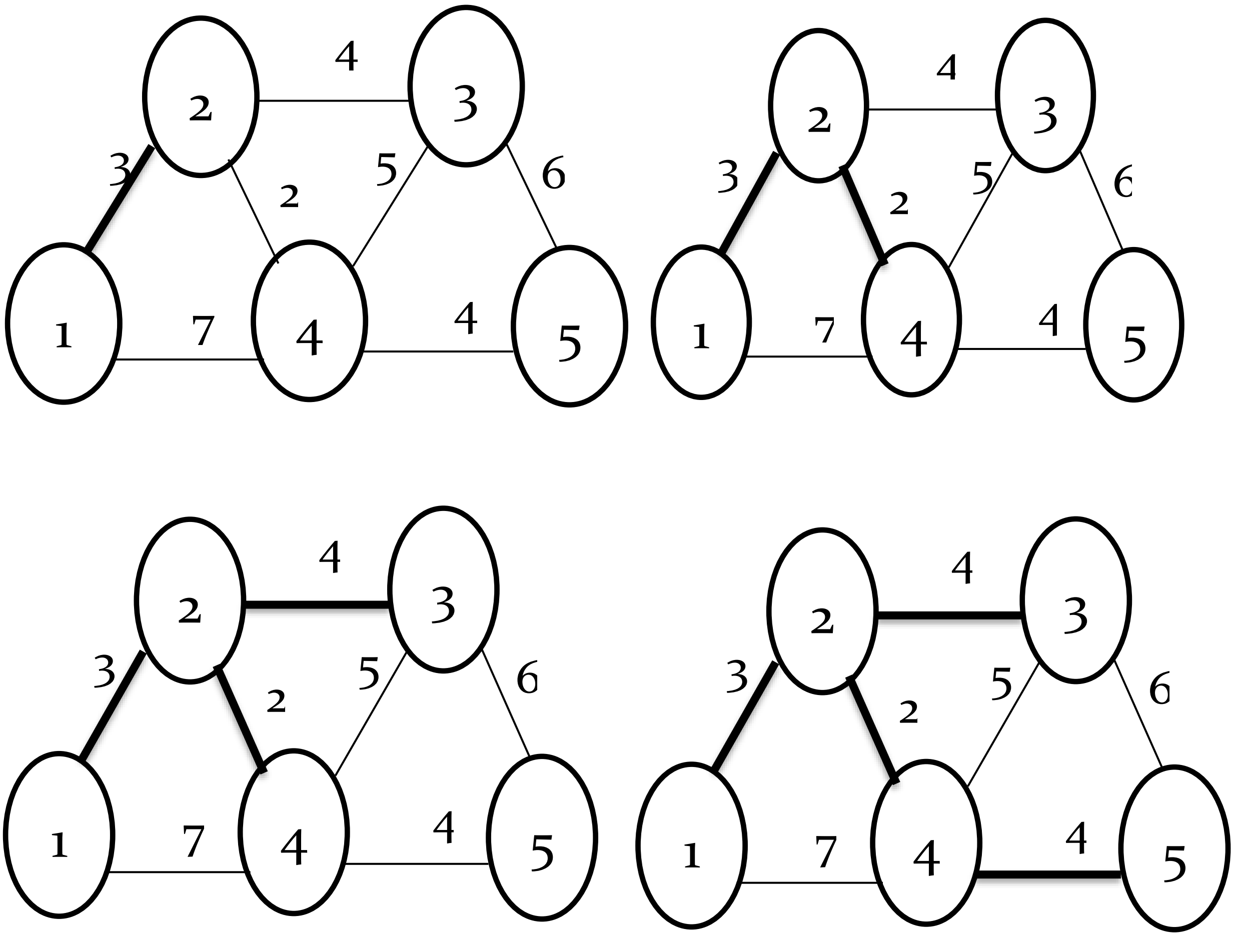
```
    {
        s[i]=0;
        d[i]=a[v][i];
    }
    s[v]=1;
    d[v]=0;
    for(j=2;j<=n;j++)
    {
        min=9999;
        for(i=1;i<=n;i++)
        {
            if(!s[i])
            {
                if(d[i]<min)
                {
                    min=d[i];
                    u=i;
                }
            }
        }
        s[u]=1;
        for(w=1;w<=n;w++)
        {
            if((a[u][w]!=9999) && (s[w]==0))
            {
                if(d[w]>(d[u]+a[u][w]))
                    d[w]=d[u]+a[u][w];
            }
        }
    }
}
```

Output:-
Enter the Number of Vertices: 5
Enter Cost Adjacent Matrix:

9999	3	9999	7	9999
3	9999	4	2	9999
9999	4	9999	5	6
7	2	5	9999	4
9999	9999	6	4	9999

Enter the Starting Vertex : 1
The Shortest path from 1 to 1 is : 0
The Shortest path from 1 to 2 is : 3
The Shortest path from 1 to 3 is : 7
The Shortest path from 1 to 4 is : 5
The Shortest path from 1 to 5 is : 9





2 4 3
3 2 5 6
1 7 4 4 5
2 4 3
3 2 5 6
1 7 4 4 5
2 4 3
3 2 5 6
1 7 4 4 5
2 4 3
3 2 5 6
1 7 4 4 5

RJS Polytechnic was established in the year 1985 - 86 by Karnataka Reddy Jana Sangha under ED 60 MPI 84, dated : 06-10-1984. RJS Polytechnic started it's journey with humble beginning. RJS Polytechnic has made great strides in the field of technical education right from it's inception in the year 1985-86. RJS Polytechnic is situated in a serene surrounding and located in koramangala, Bangalore-34. It is located in such a place that it is able to serve the cause of Students of both urban and rural areas of Bangalore in particular and the state of Karnataka and India in general.

RJS Polytechnic is run by Karnataka Reddy Jana Sangha(KRJS) , a non-profit registered body, under self financing scheme and it consists of a galaxy of personalities from all walks of life. RJS Polytechnic is affiliated to the Directorate of Technical education, recognized by the Government of Karnataka and approved by All India Council for Technical Education (AICTE), New Delhi. The affairs of RJS Polytechnic is managed by a constituted Governing Council By KRJS.

Vinayaka V M, Lecturer of Computer Science Department in RJS Polytechnic. Qualified with M.Tech., GATE and KSET. Work experience in Bapuji Institute of Engineering and Technology, Davanagere.