CS 213L

DESIGN AND ANALYSIS OF ALGORITHMS (DAA)

RECORD FILE (DAA), 2021

B. Tech (4th Semester)

Submitted to: Dr. Sanjay Kumar Sharma

Submitted by:

Name	**Riyanshi Verma**
Exam Roll Number	**1913075**
ID	BTBTC19047
Branch	Computer Science (Section B)
Email ID	btbtc19047_riyanshi@banasthali.in

Total no of pages: 85

INDEX

P1. Program to implement Max Heap.

```
//MAX HEAP PROGRAM
#include <stdio.h>
void max_heapify(int a[20],int i,int n)
{
        int l,r,large,temp;
        l=2*i; r=2*i+1;
        if(l<=n && a[l]>a[i])
                large=l;
        else large=i;
        if(r<=n && a[r]>a[large])
           large=r;
        if(i!=large)
        {
                temp=a[i];
                a[i]=a[large];
                a[large]=temp;
                max_heapify(a,large,n);
        }
}//create max heap
void create_maxheap(int a[20],int n)
{
        int i;
        for(i=n/2;i>=1;i--)
        max_heapify(a,i,n);
}
void insertintomaxheap(int a[20],int n)
{
        int i,item;
        i=n; item =a[n];
        while(i>1 && a[i/2]<item)
        {
                a[i]=a[i/2];
                i=i/2;
        }
        a[i]=item;
}
int delmax(int a[20],int *n)
{
```

```
        int x=0;
        if(*n==0)
        printf("\nHeap is empty");
        else
        {
                x=a[1];
                a[1]=a[*n];
                *n=*n-1;
                max_heapify(a,1, *n);
        }
        return x;
}
void main()
{
        int n,i,a[20];
        printf("\nEnter the size of array\n");
        scanf("%d",&n);
        printf("\nEnter the elements\n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\nArray elements are--- \n");
        for(i=1;i<=n;i++)
        printf("%d\t",a[i]);
        create_maxheap(a,n);
        printf("\nMaxheap elements are--- \n");
        for(i=1;i<=n;i++)
        printf("%d\t",a[i]);
        int x;
        printf("\nEnter element to insert: ");
        scanf("%d",&x);
        n=n+1;
        a[n]=x;
        insertintomaxheap(a,n);
        printf("\nHeap after insertion of %d is  : \n",x);
        for(i=1;i<=n;i++)
        printf("%d\t",a[i]);
        x=delmax(a,&n);
        if(x!=0)
        printf("\nDeleted element is = %d",x);
        printf("\nHeap after delection of %d is  : \n",x);
```

```
        for(i=1;i<=n;i++)
        printf("%d\t",a[i]);
}
```

P2. Program to implement Min Heap.

```
//MIN HEAP PROGRAM
#include <stdio.h>
void min_heapify(int a[20],int i,int n)
{
        int l,r,small,temp;
        l=2*i; r=2*i+1;
        if(l<=n && a[l]<a[i])
                small=l;
        else small=i;
        if(r<=n && a[r]<a[small])
           small=r;
        if(i!=small)
        {
                temp=a[i];
                a[i]=a[small];
                a[small]=temp;
                min_heapify(a,small,n);
        }
}
void create_minheap(int a[20],int n)
{
        int i;
        for(i=n/2;i>=1;i--)
        min_heapify(a,i,n);
}
void insertintominheap(int a[20],int n)
{
        int i,item;
        i=n; item =a[n];
        while(i>1 && a[i/2]>item)
        {
                a[i]=a[i/2];
                i=i/2;
```

```
        }
        a[i]=item;
}
int delmax(int a[20],int *n)
{
        int x=0;
        if(*n==0)
        printf("\nHeap is empty");
        else
        {
                x=a[1];
                a[1]=a[*n];
                *n=*n-1;
                min_heapify(a,1, *n);
        }
        return x;
}
void main()
{
        int n,i,a[20];
        printf("\nEnter the size of array\n");
        scanf("%d",&n);
        printf("\nEnter the elements\n");
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\nArray elements are--- \n");
        for(i=1;i<=n;i++)
        printf("%d\t",a[i]);
        create_minheap(a,n);
        printf("\nSorted Array elements are--- \n");
        for(i=1;i<=n;i++)
        printf("%d\t",a[i]);
        int x;
        printf("\nEnter element to insert: ");
        scanf("%d",&x);
        n=n+1;
        a[n]=x;
        insertintominheap(a,n);
        printf("\nHeap after insertion of %d is  : \n",x);
        for(i=1;i<=n;i++)
```

```
        printf("%d\t",a[i]);
        x=delmax(a,&n);
        if(x!=0)
        printf("\nDeleted element is = %d",x);
        printf("\nHeap after delection of %d is  : \n",x);
        for(i=1;i<=n;i++)
        printf("%d\t",a[i]);
}
```

P3. Program to implement HeapSort.

```
//HEAP SORT
#include <stdio.h>
void max_heapify(int a[20],int i,int n)
{
        int l,r,large,temp;
        l=2*i; r=2*i+1;
        if(l<=n && a[l]>a[i])
                large=l;
        else large=i;
        if(r<=n && a[r]>a[large])
           large=r;
        if(i!=large)
        {
                temp=a[i];
                a[i]=a[large];
                a[large]=temp;
                max_heapify(a,large,n);
        }
}//create max heap
void create_maxheap(int a[20],int n)
{
        int i;
        for(i=n/2;i>=1;i--)
        max_heapify(a,i,n);
}
//heapsort function
void heap_sort(int a[20],int n)
{
```

```
        int i,temp;
        create_maxheap( a, n);
        for(i=n;i>1;i--)
        {
                temp=a[i];
                a[i]=a[1];
                a[1]=temp;
                max_heapify(a,1,i-1);
        }
}
void main()
{
        int n,i,a[20];
        printf("\nENTER THE SIZE OF ARRAY\n");
        scanf("%d",&n);
        printf("\nENTER THE ELEMENTS\n");
        for(i=1;i<=n;i++)
           scanf("%d",&a[i]);
        printf("\nARRAY ELEMENTS ARE--- \n");
        for(i=1;i<=n;i++)
           printf("%d\t",a[i]);
        heap_sort(a,n);
        printf("\nSORTED ARRAY ELEMENTS ARE--- \n");
        for(i=1;i<=n;i++)
           printf("%d\t",a[i]);
}
```

P4. Program to implement Menu Driven Max Heap

```
//MAX HEAP PROGRAM IN MENU DRIVEN FORM
#include <stdio.h>
void max_heapify(int a[20],int i,int n)
{
        int l,r,large,temp;
        l=2*i; r=2*i+1;
        if(l<=n && a[l]>a[i])
                large=l;
        else large=i;
        if(r<=n && a[r]>a[large])
```

```
            large=r;
        if(i!=large)
        {
                temp=a[i];
                a[i]=a[large];
                a[large]=temp;
                max_heapify(a,large,n);
        }
}//create max heap
void create_maxheap(int a[20],int n)
{
        int i;
        for(i=n/2;i>=1;i--)
        max_heapify(a,i,n);
}
void insertintomaxheap(int a[20],int n)
{
        int i,item;
        i=n; item =a[n];
        while(i>1 && a[i/2]<item)
        {
                a[i]=a[i/2];
                i=i/2;
        }
        a[i]=item;
}
int delmax(int a[20],int *n)
{
        int x=0;
        if(*n==0)
        printf("\nHeap is empty");
        else
        {
                x=a[1];
                a[1]=a[*n];
                *n=*n-1;
                max_heapify(a,1, *n);
        }
        return x;
}
```

```
void main()
{
	int n,i,ch,x,a[20];
	do
	{
		printf("\n1.CREATE MAXHEAP\n2.INSERT AN
ELEMENT\n3.DELETE AN ELEMENT\n4.EXIT");
		printf("\nEnter choice:\n");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1: printf("\nEnter the size of array\n");
				scanf("%d",&n);
				printf("\nEnter the elements\n");
				for(i=1;i<=n;i++)
				    scanf("%d",&a[i]);
				create_maxheap(a,n);
				printf("\nMaxheap elements are--- \n");
				for(i=1;i<=n;i++)
				    printf("%d\t",a[i]);
				 break;
			case 2: printf("\nEnter element to insert: ");
				scanf("%d",&x);
				n=n+1;
				a[n]=x;
				insertintomaxheap(a,n);
				printf("\nHeap after insertion of %d is  : \n",x);
				for(i=1;i<=n;i++)
				   printf("%d\t",a[i]);
				break;
			case 3: x=delmax(a,&n);
				if(x!=0)
				printf("\nDeleted element is = %d",x);
				printf("\nHeap after delection of %d is  : \n",x);
				for(i=1;i<=n;i++)
				    printf("%d\t",a[i]);
				break;
			case 4: printf("\nthankyou");
				break;
		}
```

```
        }while(ch!=4);
}
```

P5. Program to implement Linear Search

```
//LINEAR SEARCH
#include <stdio.h>
#include <stdlib.h>
int linsearch(int a[10],int n,int x)
{
        int j=1;
        while(j<n && a[j]!=x)
        {
                j=j+1;
        }
        if(a[j]==x)
        return j;
        else
        return 0;
}
void main()
{
        int n;
        printf("Enter the size of array(1-10)\n");
        scanf("%d",&n);
        if(n<0 || n>10)
        {
                printf("Enter correct value of n\n ");
                scanf("%d",&n);
        }
        if(n<0 || n>10)
        {
                exit(0);
        }
        int arr[10];
        int i;
        printf("Enter your elements\n");
        for(i=0;i<n;i++)
        {
```

```
            scanf("%d",&arr[i]);
        }
        printf("You entered--  ");
        for(i=0;i<n;i++)
        {
            printf("%d    ",arr[i]);
        }
        printf("\nEnter the value to be searched\n");
        int value;
        scanf("%d",&value);
        int pos;
        pos=linsearch(arr,n,value);
        if(pos!=0)
        printf("%d is found at position %d",value,pos+1);
        else
        printf("%d is not found in the given array",value);
}
```

P6. Program to implement Linear Search using Recursion

```
// IMPLEMENT LINEAR SEARCH USING RECURSION
#include <stdio.h>
#include <stdlib.h>
int rec_lin_search(int a[20], int l, int r, int x)
{
    if (r < l)
       return -1;
    if (a[l] == x)
       return l;
    if (a[r] == x)
       return r;
    return rec_lin_search(a, l+1, r-1, x);
}
void main()
{
   int n;
        printf("Enter the size of array(1-10)\n");
        scanf("%d",&n);
        if(n<0 || n>10)
```

```
	{
		printf("Enter correct value of n\n ");
		scanf("%d",&n);
	}
	if(n<0 || n>10)
	{
		exit(0);
	}
	int arr[10];
	int i;
	printf("Enter your elements\n");
	for(i=0;i<n;i++)
	{
		scanf("%d",&arr[i]);
	}
	printf("You entered--  ");
	for(i=0;i<n;i++)
	{
		printf("%d    ",arr[i]);
	}
	printf("\nEnter the value to be searched\n");
	int value;
	scanf("%d",&value);
	int pos;
	pos=rec_lin_search(arr,1,n,value);
	if(pos!=-1)
	printf("%d is found at position %d",value,pos+1);
	else
	printf("%d is not found in the given array",value);
}
```

P7. Program to implement Binary Search using Recursion

```
//BINARY SEARCH RECURSIVE
#include <stdio.h>
#include <stdlib.h>
int Binsearch(int a[20],int low,int high,int x)
{
	int mid;
```

```
        if(low==high)
        {
                if(x==a[low])
                return low;
                else
                return 0;
        }
        else
        {
                mid=(low+high)/2;
                if(x==a[mid])
                return mid;
                else if(x<a[mid])
                Binsearch(a,low,mid-1,x);
                else
                Binsearch(a,mid+1,high,x);
        }
}
void main()
{
        int n,i,x,a[20],pos;
        printf("\nENTER THE SIZE OF ARRAY\n");
        scanf("%d",&n);
        printf("\nENTER THE %d ELEMENTS IN ASCENDING
ORDER\n",n);
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\nARRAY ELEMENTS ARE--- \n");
        for(i=1;i<=n;i++)
        printf("%d\t",a[i]);
        for(i=0;i<n-1;i++)
        {
           if(a[i]<a[i+1])
               continue;
           else break;
        }
        if(i!=n-1)
        {
            printf("\n\nSince array is not sorted binary search cannot be
 used!");
```

```
        exit(0);
        }
        printf("\nENTER THE ELEMENT TO BE SEARCHED IN
  ARRAY\n");
        scanf("%d",&x);
        pos=Binsearch(a,1,n,x);
        if(pos!=0)
        printf("\nELEMENT PRESENT AT POSITION = %d",pos);
        else
        printf("\nELEMENT IS NOT PRESENT");
}
```

P8. Program to implement Bubble Sort

```
//PROGRAM FOR BUBBLE SORT
#include<stdio.h>
void bubble(int a[20],int N)
{
        int i,j,k,temp,flag=1;
        for(i=1;(i<=N-1) && (flag==1);i++)
        {
           flag=0;
               for(j=1;j<=N-i;j++)
               {
                       if(a[j]>a[j+1])
                       {
                          temp=a[j];
                      a[j]=a[j+1];
                      a[j+1]=temp;
                      flag=1;
                       }
               }
        }
        printf("\n\nThe sorted list of elements is--  ");
        for(i=1;i<=N;i++)
        {
               printf("%d    ",a[i]);
        }
```

```
}
int main()
{
	int n;
	printf("Enter the size of array(1-20)\n");
	scanf("%d",&n);

	int arr[20];
	int i;
	printf("Enter your elements\n");
	for(i=1;i<=n;i++)
	{
		scanf("%d",&arr[i]);
	}
	printf("You entered--  ");
	for(i=1;i<=n;i++)
	{
		printf("%d    ",arr[i]);
	}
	bubble(arr,n);
	return 0;
}
```

P9. Program to implement Selection Sort

```
//PROGRAM FOR SELECTION SORT
#include<stdio.h>
void selection(int a[20],int N)
{
	int i,j,min,temp;
	for(i=0;i<N-1;i++)
	{
	   min=i;
		for(j=i+1;j<N;j++)
		{
			if(a[j]<a[min])
			{
				min=j;
```

```
                }
            }
                temp=a[i];
            a[i]=a[min];
            a[min]=temp;
        }
}
int main()
{
        int n;
        printf("Enter the size of array(1-20)\n");
        scanf("%d",&n);
        int a[20];
        int i;
        printf("Enter your elements\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("You entered--  ");
        for(i=0;i<n;i++)
        {
                printf("%d    ",a[i]);
        }
        selection(a,n);
        printf("\n\nThe sorted list of elements is--  ");
        for(i=0;i<n;i++)
        {
                printf("%d    ",a[i]);
        }
        return 0;
}
```

P10. Program to implement Insertion Sort

```
//PROGRAM TO IMPLEMENT INSERTION SORT
#include<stdio.h>
#include<stdlib.h>
void insertion(int arr[20], int n)
{
	int j,x,i;
	for(i=1;i<n;i++)
	{
		j=i;x=arr[i];
		while(arr[j-1]>x && j>0)
		{
			arr[j]=arr[j-1];j=j-1;
		}
		if (j!=i)
		arr[j]=x;
	}
}
void main()
{
	int n;
	printf("Enter the size of array(1-10)\n");
	scanf("%d",&n);
	if(n<0 || n>10)
	{
		printf("Enter correct value of n\n ");
		scanf("%d",&n);
	}
	if(n<0 || n>10)
	{
		exit(0);
	}
	int arr[10];
	int i;
	printf("Enter your elements\n");
	for(i=0;i<n;i++)
	{
		scanf("%d",&arr[i]);
	}
	printf("You entered--  ");
```

```
        for(i=0;i<n;i++)
        {
                printf("%d    ",arr[i]);
        }
        insertion(arr,n);
        printf("\n\nThe sorted list of elements is--  ");
        for(i=0;i<n;i++)
        {
                printf("%d    ",arr[i]);
        }
}
```

P11. Program to implement Tower of Hanoi

```
//PROGRAM OF TOWER OF HANOI
#include <stdio.h>
int cnt=0;// count of no. of disk movement
void ToH(int n, char TA, char TB, char TC)
{
   if (n >= 1)
   {
      ToH(n-1,TA,TC,TB);
      printf("\nMove disk-%d from %c to %c",n,TA,TB);
      ToH(n-1,TC,TB,TA);
              cnt++;
   }
}
int main()
{
   int n;
   printf("Enter the size of n\n");
   scanf("%d",&n);
       printf("The sequence of moves involved in the Tower of Hanoi are
:\n");
   ToH(n, 'A', 'B', 'C');
   printf("\nNo of moves = %d",cnt);
       return 0;
}
```

P12. Program to implement Merge Sort

```
//PROGRAM TO IMPLEMENT MERGE SORT
#include <stdio.h>
int b[20]; //global array
//merge function
void merge(int a[20],int low,int mid,int high)
{
	int i=low,h=low,j=mid+1,k;
	while(h<=mid && j<=high)
	{
		if(a[h]<a[j])
		{
			b[i]=a[h];
			h=h+1;
		}
		else
		{
			b[i]=a[j];
			j=j+1;
		}
		i=i+1;
	}
	if(h>mid)
	for(k=j;k<=high;k++)
	{
		b[i]=a[k];
		i++;
	}
	if(j>high)
	for(k=h;k<=mid;k++)
	{
		b[i]=a[k];
		i++;
	}
	for(k=low;k<=high;k++)
	{
		a[k]=b[k];
	}
```

```
}
//mergesort function
void mergesort(int a[20],int low,int high)
{
	int mid;
	if(low<high)
	{
		mid=(low+high)/2;
		mergesort(a,low,mid);
		mergesort(a,mid+1,high);
		merge(a,low,mid,high);
	}
}
//main function
int main()
{
	int a[20],n,i;
	printf("\nEnter the size of array: ");
	scanf("%d",&n);
	printf("Enter %d elements\n",n);
	for(i=0;i<n;i++)
	{
		scanf("%d",&a[i]);
	}
	printf("You entered--  ");
	for(i=0;i<n;i++)
	{
		printf("%d    ",a[i]);
	}
	mergesort(a,0,n-1);
	printf("\n\nThe sorted list of elements is--  ");
	for(i=0;i<n;i++)
	{
		printf("%d    ",a[i]);
	}
	return 0;
}
```

P13. Program to find maximum and minimum element.

```
//PROGRAM FOR MAXMIN
#include <stdio.h>
void maxmin(int a[20],int low,int high,int *max,int *min)
{
	int max1,min1,mid;
	if(low==high)
	{
		*max=*min=a[low];
	}
	else if(low ==(high-1))
	{
		if(a[low]>a[high])
		{
			*max=a[low];
			*min=a[high];
		}
		else
		{
			*max=a[high];
			*min=a[low];
		}
	}
	else
	{
		mid=(low+high)/2;
		maxmin(a,low,mid,max,min);
		maxmin(a,mid+1,high,&max1,&min1);
		if(min1<*min)
		*min=min1;
		if(max1>*max)
		*max=max1;
	}
}
void main()
{
	int n,i,max,min,low,high,a[20];
	printf("\nENTER THE SIZE OF ARRAY\n");
	scanf("%d",&n);
```

```
        printf("\nENTER THE %d ELEMENTS\n",n);
        for(i=1;i<=n;i++)
        scanf("%d",&a[i]);
        printf("\nARRAY ELEMENTS ARE--- \n");
        for(i=1;i<=n;i++)
        printf("%d\t",a[i]);
        low=1;
        high=n;
        maxmin(a,low,high,&max,&min);
        printf("\nTHE MAX ELEMENT IS %d AND MIN ELEMENT IS
%d",max,min);

}
```

P14. Program to implement Quick Sort using Hoare partition.

```
//PROGRAM FOR QUICK SORT USING HOARE PARTITION
#include <stdio.h>
//Partition function
int Partition(int a[20], int low, int high)
{
        int temp,i,j,pivot;
   pivot = a[low];
   i = low - 1;
   j = high + 1;
   while (1)
   {
      do
              {
         i=i+1;
      } while (a[i] < pivot);

      do
              {
         j=j-1;
      } while (a[j] > pivot);
      if (i<j)
      {
```

```
        temp=a[i]; a[i]=a[j]; a[j]=temp;
                }
                else
           return j;
    }
}
// Quicksort function
void QuickSort(int a[20], int low, int high)
{
        int pivot;
    if (low < high)
    {
        pivot = Partition(a, low, high);
       QuickSort(a, low, pivot);
       QuickSort(a, pivot + 1, high);
        }
}
int main()
{
   int n,i,a[20];
        printf("Enter the size of array(1-20)\n");
        scanf("%d",&n);
        printf("Enter %d elements\n",n);
        for(i=1;i<=n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("You entered--  ");
        for(i=1;i<=n;i++)
        {
                printf("%d    ",a[i]);
        }
        QuickSort(a,1,n);
        printf("\n\nThe sorted list of elements is--  ");
        for(i=1;i<=n;i++)
        {
                printf("%d    ",a[i]);
        }
        return 0;
}
```

P15. Program to implement Quick Sort Using Hoare partition having first element as Pivot.

```
//PROGRAM FOR QUICK SORT HAVING FIRST ELEMENT AS PIVOT
#include <stdio.h>
//function for partition
int Partition(int a[20], int low, int high)
{
        int i,j,x,temp;
        x=a[low];
        i=low;
        for(j=low+1;j<=high;j++)
        {
                if(a[j]<=x)
                {
                        i=i+1;
                        if(i!=j)
                        {
                                temp=a[i]; a[i]=a[j]; a[j]=temp;
                        }
                }
        }
                a[low]=a[i]; a[i]=x;
                return i;
}
// Quicksort function
void QuickSort(int a[20], int low, int high)
{
        int pivot;
   if (low < high)
   {
        pivot = Partition(a, low, high);
       QuickSort(a, low, pivot-1);
       QuickSort(a, pivot + 1, high);
        }
}
//main function
int main()
```

```
{
  int n,i,a[20];
        printf("Enter the size of array(1-20)\n");
        scanf("%d",&n);
        printf("Enter %d elements\n",n);
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("You entered--  ");
        for(i=0;i<n;i++)
        {
                printf("%d    ",a[i]);
        }
        QuickSort(a,0,n-1);
        printf("\n\nThe sorted list of elements is--  ");
        for(i=0;i<n;i++)
        {
                printf("%d    ",a[i]);
        }
        return 0;
}
```

P16. Program to implement Quick Sort Using Hoare partition having last element as Pivot.

```
//PROGRAM FOR QUICK SORT HAVING LAST ELEMENT AS
PIVOT
#include <stdio.h>
//function for partition
int Partition(int a[20], int low, int high)
{
        int i,j,x,temp;
        x=a[high];
        i=low-1;
        for(j=low;j<=high-1;j++)
        {
                if(a[j]<=x)
```

```
            {
                i=i+1;
                if(i!=j)
                {
                    temp=a[i]; a[i]=a[j]; a[j]=temp;
                }
            }
        }
            a[high]=a[i+1]; a[i+1]=x;
            return (i+1);
}
// Quicksort function
void QuickSort(int a[20], int low, int high)
{
    int pivot;
   if (low < high)
   {
      pivot = Partition(a, low, high);
      QuickSort(a, low, pivot-1);
      QuickSort(a, pivot + 1, high);
   }
}
//main function
int main()
{
    int n,i,a[20];
    printf("Enter the size of array(1-20)\n");
    scanf("%d",&n);
    printf("Enter %d elements\n",n);
    for(i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("You entered--  ");
    for(i=0;i<n;i++)
    {
        printf("%d    ",a[i]);
    }
    QuickSort(a,0,n-1);
    printf("\n\nThe sorted list of elements is--  ");
```

```
        for(i=0;i<n;i++)
        {
                printf("%d    ",a[i]);
        }
        return 0;
}
```

P17. Program to find Kth Smallest element in an array.

```
//PROGRAM FOR FINDING K^TH SMALLEST ELEMENT IN AN
ARRAY
#include <stdio.h>
//Partition function
int partition(int a[20], int low, int high)
{
   int j,x,i,temp;
   x = a[high];//pivot as last element
        i = low;
   for (j = low; j <= high - 1; j++)
        {
       if (a[j] <= x)
                {
          temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
          i=i+1;
       }
   }
   temp=a[i];
        a[i]=a[high];
        a[high]=temp;
   return i;
}
//function to find kth smallest element
int kthSmallest(int a[20], int low, int high, int k)
{
        int pos;
   if (k>0 && k<=high-low+1)
```

```
        {
       pos = partition(a,low,high);
              if (pos-low == k-1)
          return a[pos];
       if (pos-low > k-1)
          return kthSmallest(a, low, pos-1, k);
       else
        return kthSmallest(a, pos+1,high,k-pos+low-1);
    }
   return -1;
}
//main function
int main()
{
        int n,i,a[20],k,ans;
        printf("Enter the size of array(1-20)\n");
        scanf("%d",&n);

        printf("Enter %d elements\n",n);
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }

        printf("You entered--  ");
        for(i=0;i<n;i++)
        {
                printf("%d    ",a[i]);
        }
        printf("\nEnter the value of k\n");
        scanf("%d",&k);
        ans=kthSmallest(a,0,n-1,k);
        if(ans==-1)
        printf("\nValue of k is NOT in range of array!!");
        else
        printf("%d'th smallest element is %d",k,ans);
        printf("final array--  ");
        for(i=0;i<n;i++)
        {
                printf("%d    ",a[i]);
```

```
        }
        return 0;
}
```

P18. Program to implement Simple Union and Find.

```
//SIMPLE UNION AND SIMPLE FIND PROGRAM
#include <stdio.h>
int n,p[20];
void setunion(int i,int j)
{//i and j are the roots of tree that represent the set
        p[i]=j;
}
int sfind(int i)
{
        int j;
        j=i;
        while(p[j]>0)
        j=p[j];
        return j;
}
int main()
{
        int i,j,r1,r2,ch;
        printf("\nEnter the value of n:");
        scanf("%d",&n);//set each element in their own p[i]
        for(i=1;i<=n;i++)
        p[i]=-1;
        printf("\n  P[]= ");
        for(i=1;i<=n;i++)
        printf("%d\t",p[i]);
        printf("\nele[]= ");
        for(i=1;i<=n;i++)
        printf("%d\t",i);
        do
        {
                printf("\n1. FIND SIMPLE UNION\n2. DO SIMPLE
FIND\n3. EXIT\n");
```

```
            printf("\nEnter your choice : ");
            scanf("%d",&ch);
            switch(ch)
            {
                case 1: printf("\nEnter the roots of two sets\n");
                        scanf("%d%d",&r1,&r2);
                        if((p[r1]==-1 && (p[r2]==-1)))
                        setunion(r1,r2);
                        else
                        printf("Invalid roots");
                        break;
                case 2: printf("\nEnter the element to find : ");
                        scanf("%d",&i);
                        j=sfind(i);
                        printf("\nThe element %d is in the set whose
root is %d\n",i,j);
                        break;
                case 3: printf("\nTHANKYOU!!");break;

            }
            printf("\n  P[]= ");
            for(i=1;i<=n;i++)
            printf("%d\t",p[i]);
            printf("\nele[]= ");
            for(i=1;i<=n;i++)
            printf("%d\t",i);
        }while(ch!=3);
        return 0;
}
```

P19. Program to implement Weighted Union and Collapsing Find.

```
//PROGRAM FOR WEIGHTED UNION AND COLLAPSING FIND
#include <stdio.h>
int n,p[20];
//weighted union function
void weightedunion(int i, int j)
```

```
{//i and j are the roots of tree of different sets
        int temp;
        if((p[i]>0) || (p[j]>0))
        printf("\nInvalid roots");
        else
        {
                temp=p[i]+p[j];
                if(p[i]>p[j])
                {
                        p[i]=j;
                        p[i]=temp;
                }
                else
                {
                        p[j]=i;
                        p[i]=temp;
                }
        }
}
//collaping find algorithm or path compression algorithm
int collapsingfind(int i)
{
        int r,s;
        r=i;
        while(p[r]>0)
        r=p[r];
        while(i!=r)
        {
                s=p[i];
                p[i]=r;
                i=s;
        }
        return r;
}
int main()
{
        int i,j,r1,r2,ch;
        printf("\nEnter the value of n:");
        scanf("%d",&n);//set each element in their own p[i]
        for(i=1;i<=n;i++)
```

```
        p[i]=-1;
        printf("\n  P[]= ");
        for(i=1;i<=n;i++)
        printf("%d\t",p[i]);
        printf("\nele[]= ");
        for(i=1;i<=n;i++)
        printf("%d\t",i);
        do
        {
                printf("\n1. FIND WEIGHTED UNION\n2. DO
COLLAPSING FIND\n3. EXIT\n");
                printf("\nEnter your choice : ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("\nEnter the roots of two sets\n");
                                        scanf("%d%d",&r1,&r2);
                                        if((p[r1]==-1 && (p[r2]==-1)))
                                        weightedunion(r1,r2);
                                        else
                                        printf("Invalid roots");
                                        break;
                        case 2: printf("\nEnter the element to find : ");
                                        scanf("%d",&i);
                                        j=collapsingfind(i);
                                        printf("\nThe element %d is in the set whose
root is %d\n",i,j);
                                        break;
                        case 3: printf("\nTHANKYOU!!");break;
                }
                printf("\n  P[]= ");
                for(i=1;i<=n;i++)
                printf("%d\t",p[i]);
                printf("\nele[]= ");
                for(i=1;i<=n;i++)
                printf("%d\t",i);
        }while(ch!=3);
        return 0;
}
```

P20. Program to implement Adjacency Representation of Graph

```
//ADJCENCY MATRIX REPRESENTATION
#include<stdio.h>
void printmatrix(int a[10][10],int r,int c)
{
	int i,j;
	printf("\nAdjacency matrix size %d X %d\n",r,c);
	for(i=1;i<=r;i++)
	{
		for(j=1;j<=c;j++)
		printf("%d ",a[i][j]);
		printf("\n");
	}
}
int main()
{
	int r,c,nv,ne,mat[10][10],v1,v2,i,j;
	printf("\nEnter the no. of vertices and edges in graph : \n");
	scanf("%d %d",&nv,&ne);
	for(i=1;i<=nv;i++)
		for(j=1;j<=nv;j++)
			mat[i][j]=0;
	for(i=1;i<=ne;i++)
	{
		printf("\nEnter the two end vertices of edge no. %d:\n",i);
		scanf("%d %d",&v1,&v2);
		mat[v1][v2]=1;
		mat[v2][v1]=1;
	}
	printmatrix(mat,nv,nv);
	return 0;
}
```

P21. Program to implement Breadth First Search (BFS)

```
//BREADTH FIRST SEARCH UNDIRECTED GRAPH- USING
MATRIX REPRESENTATION OF GRAPH
#include <stdio.h>
#define max 10
int g[10][10],q[max],vis[10];
int n,front=-1,rear=-1;
//empty queue funtion
int emptyq()
{
        if(front==-1 && rear==-1)
        return 1;
        else
        return 0;
}
//insert into queue
void insertq(int x)
{
        if((front==0) && (rear==max-1))
        printf("\nOVERFLOW");
        else
        {
                rear++;
                q[rear]=x;
                if(front==-1)
                front =0;
        }
}
//delete from queue
int deleteq()
{
        int d;
        if(emptyq()==1)
        d=0;
        else
        {
                d=q[front];
                if(front==rear)
                {
```

```
                front=-1;
                rear=-1;
            }
            else
                front=front+1;
        }
        return d;
}
//BFS function
bfs (int v)
{
        int i,w;
        for(i=1;i<=n;i++)
        vis[i]=0;
        printf("\nTraversal from vertex %d\n",v);
        printf("\t%d",v);
        vis[v]=1;
        insertq(v);
        while(!emptyq())
        {
            v=deleteq();
            for(w=1;w<=n;w++)
            {
                if(g[v][w]==1 && vis[w]==0)
                {
                    printf("\t%d",w);
                    vis[w]=1;
                    insertq(w);
                }
            }
        }
}
int main()
{
        int i,j,v,e,k,v1,v2;
        printf("\nEnter total no.of vertices in Graph : ");
        scanf("%d",&n);
        printf("\nEnter total no.of edges in Graph : ");
        scanf("%d",&e);
        for(i=1;i<=n;i++)
```

```
        {
                for(j=1;j<=n;j++)
                {
                        g[i][j]=0;
                }
        }
        for(k=1;k<=e;k++)
        {
                printf("\nEnter the two end vertices of edge no. %d:\n",k);
                scanf("%d %d",&v1,&v2);
                g[v1][v2]=1;
                g[v2][v1]=1;
        }
        //printing adjacency matrix of graph
        printf("\nAdjacency matrix size %d X %d\n",v,e);
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                printf("%d ",g[i][j]);
                printf("\n");
        }
        printf("\nEnter starting vertex : ");
        scanf("%d",&v);
        bfs(v);
        return 0;
}
```

P22. Program to implement Breadth First Traversal (BFT)

```
//BREADTH FIRST TRAVERSAL UNDIRECTED GRAPH- USING MATRIX REPRESENTATION OF GRAPH
#include <stdio.h>
#define max 10
int g[10][10],q[max],vis[10];
int n,front=-1,rear=-1;
//empty queue funtion
int emptyq()
{
        if(front==-1 && rear==-1)
        return 1;
```

```
        else
        return 0;
}
//insert into queue
void insertq(int x)
{
        if((front==0) && (rear==max-1))
        printf("\nOVERFLOW");
        else
        {
                rear++;
                q[rear]=x;
                if(front==-1)
                front =0;
        }
}
//delete from queue
int deleteq()
{
        int d;
        if(emptyq()==1)
        d=0;
        else
        {
                d=q[front];
                if(front==rear)
                {
                        front=-1;
                        rear=-1;
                }
                else
                        front=front+1;
        }
        return d;
}
//BFS function
void bfs (int v)
{
        int i,w;
        printf("\t%d",v);
```

```
        vis[v]=1;
        insertq(v);
        while(!emptyq())
        {
                v=deleteq();
                for(w=1;w<=n;w++)
                {
                        if(g[v][w]==1 && vis[w]==0)
                        {
                                printf("\t%d",w);
                                vis[w]=1;
                                insertq(w);
                        }
                }
        }
}
void bft()
{
        int i;
        for(i=1;i<=n;i++)
        vis[i]=0;//initialize visisted vertices
        for(i=1;i<=n;i++)
        {
                if(vis[i]==0)
                bfs(i);
        }
}
int main()
{
        int i,j,v,e,k,v1,v2;
        printf("\nEnter total no.of vertices in Graph : ");
        scanf("%d",&n);
        printf("\nEnter total no.of edges in Graph : ");
        scanf("%d",&e);
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        g[i][j]=0;
                }
```

```
        }
        for(k=1;k<=e;k++)
        {
                printf("\nEnter the two end vertices of edge no. %d:\n",k);
                scanf("%d %d",&v1,&v2);
                g[v1][v2]=1;
                g[v2][v1]=1;
        }
        //printing adjacency matrix of graph
        printf("\nAdjacency matrix size %d X %d\n",n,e);
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                printf("%d ",g[i][j]);
                printf("\n");
        }

        printf("\nBreadth First Traversal of graph: \n");
        bft();
        return 0;
}
```

P23. Program to implement Depth First Search (DFS)

```
//DEPTH FIRST SEARCH UNDIRECTED GRAPH- USING MATRIX REPRESENTATION OF GRAPH
#include <stdio.h>
#define max 10
int g[10][10],vis[10];
int n;
//DFS function
dfs (int v)
{
        int i,w;
        vis[v]=1;
        printf("%d \t ",v);
                for(w=1;w<=n;w++)
                {
```

```
                if(g[v][w]==1 && vis[w]==0)
                {
                        dfs(w);
                }
        }
}
int main()
{
        int i,j,v,e,k,v1,v2;
        printf("\nEnter total no.of vertices in Graph : ");
        scanf("%d",&n);
        printf("\nEnter total no.of edges in Graph : ");
        scanf("%d",&e);
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        g[i][j]=0;
                }
        }
        for(k=1;k<=e;k++)
        {
                printf("\nEnter the two end vertices of edge no. %d:\n",k);
                scanf("%d %d",&v1,&v2);
                g[v1][v2]=1;
                g[v2][v1]=1;
        }
        //printing adjacency matrix of graph
        printf("\nAdjacency matrix size %d X %d\n",n,e);
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                printf("%d ",g[i][j]);
                printf("\n");
        }

        printf("\nFor DFS Enter starting vertex : ");
        scanf("%d",&v);
        for(i=1;i<=n;i++)
        vis[i]=0;
```

```
        printf("\nTraversal from vertex %d\n",v);
        dfs(v);
        return 0;
}
```

P24. Program to implement Depth First Traversal (DFT)

```
//DEPTH FIRST TRAVERSAL UNDIRECTED GRAPH- USING
MATRIX REPRESENTATION OF GRAPH
#include <stdio.h>
#define max 10
int g[10][10],vis[10];
int n;
//DFS function
dfs (int v)
{
        int i,w;
        vis[v]=1;
        printf("%d \t ",v);
                for(w=1;w<=n;w++)
                {
                        if(g[v][w]==1 && vis[w]==0)
                        {
                                dfs(w);
                        }
                }
        }

void dft(int n)
{
        int i;
        for(i=1;i<=n;i++)
        vis[i]=0;//initialize visisted vertices
        for(i=1;i<=n;i++)
        {
                if(vis[i]==0)
                dfs(i);
```

```
        }
}
int main()
{
        int i,j,v,e,k,v1,v2;
        printf("\nEnter total no.of vertices in Graph : ");
        scanf("%d",&n);
        printf("\nEnter total no.of edges in Graph : ");
        scanf("%d",&e);
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        g[i][j]=0;
                }
        }
        for(k=1;k<=e;k++)
        {
                printf("\nEnter the two end vertices of edge no. %d:\n",k);
                scanf("%d %d",&v1,&v2);
                g[v1][v2]=1;
                g[v2][v1]=1;
        }
        //printing adjacency matrix of graph
        printf("\nAdjacency matrix size %d X %d\n",n,e);
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                printf("%d ",g[i][j]);
                printf("\n");
        }

        printf("\nDepth First Traversal from vertex %d\n",v);
        dft(n);
        return 0;
}
```

P25. Program to implement BFS, BFT, DFS using linked list representation of graph.

```
//BFS, BFT, DFS USING LINKED LIST REPRESENTATION OF GRAPH
#include <stdio.h>
#include <stdlib.h>
typedef struct notetype
{
        int info;
        struct nodetype *next;
}node;
node *getnode()
{
        node *p;
        p=(node *)malloc(sizeof(node));
        return p;
}
node *a[10];
int n,visited[10],q[10],rear=-1,front=0;
void dfs(int );
void bfs(int );
int empty()
{
        if(rear<front)
        return 1;
        return 0;
}
int deleteq()
{
        int x;
        if(rear<front)
        return 0;
        x=q[front];
        front++;
        return x;
}
void insertq(int x)
{
        if(rear==10)
```

```
        {
                printf("\nQueue is FULL");
                return;
        }
        rear++;
        q[rear]=x;
}
int main()
{
        int i,k,index,ch,v;
        node *p,*r;
        printf("\nEnter the number of vertices in graph : ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
                a[i]=getnode();
                p=a[i];
                p->info=i;
                p->next=NULL;
                printf("\nEnter the number of vertices adjacent from vertex
%d : ",i);
                scanf("%d",&k);
                printf("\nEnter those %d vertices : ",k);
                while(k>=1)
                {
                        scanf("%d",&index);
                        r=getnode();
                        r->info=index;
                        r->next=NULL;
                        p->next=r;
                        p=r;
                        k--;
                }
        }
        printf("\nLinked List representation :\n");
        for(i=1;i<=n;i++)
        {
                printf("HEAD->");
                p=a[i];
                while(p!=NULL)
```

```
            {
                printf("%d->",p->info);
                p=p->next;
            }
            printf("NULL");
            printf("\n");
        }
        do
        {
            for(i=1;i<=n;i++)
            visited[i]=0;
            printf("\n1. BFS\n2. BFT\n3. DFS\n4. EXIT\n");
            printf("\nEnter your choice : ");
            scanf("%d",&ch);
            switch(ch)
            {
                case 1: printf("\nEnter the BFS index : ");
                        scanf("%d",&v);
                        bfs(v);
                        break;
                case 2: for(i=1;i<=n;i++)
                        {
                            if(visited[i]==0)
                            bfs(i);
                        }
                        break;
                case 3: printf("\nEnter the DFS index : ");
                        scanf("%d",&v);
                        dfs(v);
                        break;
                case 4: printf("\nTHANKYOU!!");
            }
        }while(ch!=4);
        return 0;
}
void bfs(int v)
{
        int x;
        node *p;
        visited[v]=1;
```

```
        printf("%d\t",v);
        insertq(v);
        while(!empty())
        {
                x=deleteq();
                p=a[x];
                while(p!=NULL)
                {
                        if(visited[p->info]==0)
                        {
                                printf("%d\t",p->info);
                                insertq(p->info);
                                visited[p->info]=1;
                        }
                        p=p->next;
                }
        }
}
void dfs(int v)
{
        node *p;
        visited[v]=1;
        printf("%d\t",v);
        p=a[v];
        while(p!=NULL)
        {
                if(visited[p->info]==0)
                {
                        dfs(p->info);
                }
                p=p->next;
        }
}
```

P26. Program to implement Fractional Knapsack (Greedy Method)

```
//FRACTIONAL KNAPSACK (GREEDY METHOD)
#include <stdio.h>
struct knaps
{
	int id;
	float p;
	float w;
};
int n;
void knapsack(struct knaps *kn,float m)
{
	float x[10],u,profit=0.0,weight=0.0;
	int i,j;
	for(i=1;i<=n;i++)
		x[i]=0;
	u=m;
	for(i=1;i<=n;i++)
	{
		if(kn[i].w > u)
			break;
		x[i]=1;
		u=u-kn[i].w;
		profit=profit + kn[i].p;
	}
	if(i<=n)
	{
		x[i]=u/kn[i].w;
	}
	for(i=1;i<=n;i++)
	{
		profit= profit + kn[i].p * x[i];
		weight= weight + kn[i].w * x[i];
	}
	printf("\nThe optimal solution vector : \n");
	for(i=1;i<=n;i++)
	printf("x[%d]=%.2f\t",kn[i].id,x[i]);
```

```
        printf("\nThe profit =%.2f and Total weight =%.2f \n", profit,
weight);
}

void sort(struct knaps *ob,int n)
{
        int i,j;
        struct knaps temp;
        //sorting in decreasing order of profit/weight
        for(i=1;i<=n-1;i++)
        {
                for(j=1;j<=n-i;j++)
                {
                        if((ob[j].p/ob[j].w) < (ob[j+1].p/ ob[j+1].w))
                        {
                                temp=ob[j];
                                ob[j]=ob[j+1];
                                ob[j+1]=temp;
                        }
                }
        }
}
int main()
{
        int i,j;
        struct knaps obj[10],temp;
        float m;
        printf("\nEnter the number of objects : ");
        scanf("%d",&n);
        printf("\nEnter object_ID, profit and weight of %d object \n",n);
        for(i=1;i<=n;i++)
        {
                scanf("%d%f%f",&obj[i].id,&obj[i].p,&obj[i].w);
        }
        printf("\nEnter the capacity of the knapscak : ");
        scanf("%f",&m);
        printf("\nObject_ID\tProfit\tWeight\tprofit/weight\n");
        for(i=1;i<=n;i++)
        {
```

```
            printf("%d\t%f\t%f\t%.2f\n",obj[i].id,obj[i].p, obj[i].w,
obj[i].p/obj[i].w);
        }
        sort (obj,n);
        knapsack(obj,m);
        return 0;
}
```

P27. Program to implement Job Sequencing Problem

```
//JOB SEQUENCING PROGRAM
#include<stdio.h>
struct job
{
        int id;
        int p;
        int d;
};
void jobseq(struct job *,int n);
int main()
{
        int n,i,j;
        struct job jb[10],temp;
        printf("\nEnter the no of jobs : ");
        scanf("%d",&n);
        printf("\nEnter the job_no, profit and deadline of %d jobs : \n",n);
    for(i=1;i<=n;i++)
            scanf("%d%d%d",&jb[i].id,&jb[i].p,&jb[i].d);
        printf("\nJob_No.\tProfit\tDeadline\n");
        for(i=1;i<=n;i++)
            printf("%d  \t %d  \t %d\n",jb[i].id,jb[i].p,jb[i].d);
        //sorting in jobs decreaing order of profit
        for(i=1;i<=n-1;i++)
        {
            for(j=1;j<=n-i;j++)
            {
                if(jb[j].p < jb[j+1].p)
```

```
                {
                        temp=jb[j];
                        jb[j]=jb[j+1];
                        jb[j+1]=temp;
                }
        }
    }
    printf("\nAfter sorting : ");
    printf("\nJob_No.\tProfit\tDeadline\n");
    for(i=1;i<=n;i++)
        printf("%d  \t %d  \t %d\n",jb[i].id,jb[i].p,jb[i].d);
    jobseq(jb,n);
    return 0;
}
void jobseq(struct job *jb,int n)
{
    int j[10],k,i,r,tprofit=0,q;
    jb[0].d=0;
    j[0]=0;
    j[1]=1;
    k=1;
    //tprofit+=jb[1].p;

    for(i=2;i<=n;i++)
    {
        r=k;
        while(jb[j[r]].d > jb[i].d && jb[j[r]].d != r)
            r=r-1;
        if(jb[j[r]].d <= jb[i].d && jb[i].d > r)
        {
            for(q=k;q>=r+1;q--)
                j[q+1]=j[q];
            j[r+1]=i;
            k=k+1;
            }
    }
    for(i=1;i<=k;i++)
    tprofit+=jb[i].p;//total profit
    printf("\nSolution Subsets of jobs are : \n{");
    for(i=1;i<=k;i++)
```

```
        printf("%d,",jb[j[i]].id);
        printf("}\nTotal profit = %d",tprofit);
}
```

P28. Program to implement Prism program for Minimum Spanning Tree Graph Representation

```
//PRISM PROGRAM FOR MINIMUM SPANNING TREE GRAPH REPRESENTATION
#include <stdio.h>
void prims(int g[10][10],int n);
int near[10];
int main()
{
        int i,j,n,e,ch,v1,v2,g[10][10],w;
        printf("\nEnter the no of vertices in the graph: ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        g[i][j]=999;
        printf("\nGraph is 1. Directed 2. Undirected\nEnter your choice : ");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: printf("\nEnter the number of edges in th directed graph : ");
                                scanf("%d",&e);
                                printf("\nEnter the pair of vertices v1--->v2 and weight\n");
                                for(i=1;i<=e;i++)
                                {
                                        scanf("%d%d%d",&v1,&v2,&w);
                                        g[v1][v2]=w;
                                }
                                break;
```

```
            case 2: printf("\nEnter the number of edges in undirected
graph : ");
                    scanf("%d",&e);
                    printf("\nEnter the pair of vertices v1--->v2 and
weight\n");
                    for(i=1;i<=e;i++)
                    {
                        scanf("%d%d%d",&v1,&v2,&w);
                        g[v1][v2]=w;
                        g[v2][v1]=w;
                    }
                    break;
            default : printf("\nEnter correct choice");
        }
        printf("\nMatrix Representation : \n");
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                printf("%d\t",g[i][j]);
            }
            printf("\n");
        }
        prims(g,n);
        return 0;
}

void prims(int cost[10][10], int n)
{
        int i,v1,v2,j,v,costmst=0,min,t[10][10],q,k;
        //finding the minimum code edge
        min=99;
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                if(min>cost[i][j])
                {
                    min=cost[i][j];
                    v1=i;
```

```
                        v2=j;
                    }
                }
            }
            t[1][1]=v1;
            t[1][2]=v2;
            costmst+=cost[v1][v2];
            //initialize the near[] value of vertices
            for(i=1;i<=n;i++)
            {
                if(cost[v1][i]<cost[v2][i])
                near[i]=v1;
                else
                near[i]=v2;
            }
            near[v1]=near[v2]=0;//selecting remaining n-1 edges
            for(i=2;i<=n-1;i++)
            {
                min=99;
                //select a vertex j such that near[j]!=0 and a[j][near[j]]is
minimum
                for(q=1;q<=n;q++)
                {
                    if(cost[q][near[q]]<min && near[q]!=0)
                    {
                        min=cost[q][near[q]];
                        j=q;
                    }
                }
                //update cost of MST
                costmst+=min;
                t[i][1]=j;
                t[i][2]=near[j];
                near[j]=0;
                //update near[] value of other vertices
                for(k=1;k<=n;k++)
                {
                    if(cost[k][near[k]] > cost[k][j] && near[k]!=0)
                    near[k]=j;
                }
```

```
        }
        printf("\nThe minimum spamming tree is as follows :\n");
        for(i=1;i<=n-1;i++)
        {
                printf("(%d , %d)",t[i][1],t[i][2]);
                printf("\n");
        }
        printf("Minimum cost of MST is %d",costmst);
}
```

P29. Program to implement Kruskal Program for Minimum Spanning Tree.

```
//KRUSKAL PROGRAM FOR MINIMUM SPANNING TREE
#include <stdio.h>
void minheapify(int i);
void createminheap();
void kruskals();
int edge[10],g[10][10],n,e,p[10];
//main function
int main()
{
        int i,j,ch,v1,v2,w;
        printf("\nEnter the number of vertices in the graph : ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        g[i][j]=999;
        printf("\nGraph is 1. Directed 2. Undirected\nEnter your choice :
");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: printf("\nEnter the number of edges in th directed
graph : ");
                                scanf("%d",&e);
```

```
                        printf("\nEnter the pair of vertices v1--->v2 and
weight\n");
                        for(i=1;i<=e;i++)
                        {
                                scanf("%d%d%d",&v1,&v2,&w);
                                g[v1][v2]=w;
                                edge[i]=w;
                        }
                        break;
                case 2: printf("\nEnter the number of edges in undirected
graph : ");
                        scanf("%d",&e);
                        printf("\nEnter the pair of vertices v1--->v2 and
weight\n");
                        for(i=1;i<=e;i++)
                        {
                                scanf("%d%d%d",&v1,&v2,&w);
                                g[v1][v2]=w;
                                g[v2][v1]=w;
                                edge[i]=w;
                        }
                        break;
                default : printf("\nEnter correct choice");
        }
        printf("\nMatrix Representation : \n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        printf("%d\t",g[i][j]);
                }
                printf("\n");
        }
        kruskals();
        return 0;
}
//find function to find the set that contains element x
int find(int x)
{
        int r=x;
```

```
        while(p[r]> 0)
                r=p[r];
        return r;
}
//dunion function to find the union of two sets whose roots are j and k
void dunion(int j,int k)
{
        p[j]=k;
}
//delete a mincost edge from min heap
int mindelete()
{
        int x;
        x=edge[1];
        edge[1]=edge[e];
        e--;
        minheapify(1);
        return x;
}
//kruskals function for MST
void kruskals()
{
        int i=1,t[10][2],j,x,q,w,v1,v2,costmst=0,s,r;
        //create min heap
        createminheap();
        printf("\nArray after min heap :\n\n");
        for(i=1;i<=n;i++)
         p[i]=-1;
        i=0;
        while(i<n-1 && e>0)
        {
                x=mindelete();//delete a mincost edge
                for(q=1;q<=n;q++)
                {
                        for(w=1;w<=n;w++)
                        {
                                if(g[q][w]==x)
                                {
                                        v1=q;v2=w;
                                }
```

```
                }
            }
            s=find(v1);
            r=find(v2);
            if(s!=r)
            {
                i++;
                t[i][1]=v1;
                t[i][2]=v2;
                dunion(v1,v2);
                costmst +=g[v1][v2];
            }
        }
        if(i<n-1)
        printf("\nNo minimum spamming tree.");
        else
        {
            printf("\nThe spamming tree : \n");
            for(i=1;i<=n-1;i++)
            {
                printf("Edge( %d , %d ) cost = %d",t[i][1],t[i][2],
g[t[i][1]][t[i][2]]);
                printf("\n");
            }
            printf("\nCost of minimum spamming tree = %d",costmst);
        }
}
void createminheap()
{
        int i;
        for(i=e/2;i>=1;i--)
        minheapify(i);
}
void minheapify(int i)
{
        int small=i,l,r,temp;
        l=2*i;r=2*i+1;
        if(l<=e && (edge[l]<edge[small]))
        small=l;
        if(r<=e && (edge[r]<edge[small]))
```

```
        small=r;
        if(i!=small)
        {
                temp=edge[i];
                edge[i]=edge[small];
                edge[small]=temp;
                minheapify(small);
        }
}
```

P30. Program to implement Dijkstra- Source Shortest Path (SSP)

```
//PROGRAM FOR SOURCE SHORTEST PATH (SSP)
#include<stdio.h>
#include<conio.h>
//gr is adjacency matrix, c[i][j] cost matrix
//dist[i] 1<=i<=n distance from source to other vertex i
int main()
{
        int i,e,n,c[20][20],j,k,gr[20][20],min,w;
        int v,u,dist[20],s[20];
        printf("\nEnter the total no. of vertices in graph ");
        scanf("%d",&n);
        //adjecency matrix
        for(i=1;i<=n;i++)
         for(j=1;j<=n;j++)
                gr[i][j]=0;
        printf("\nEnter total no.of edges ");
        scanf("%d",&e);
        //fill the cost matrix diagonal 0 and all other 1000 if no edge
        for(i=1;i<=e;i++)
        for(j=1;j<=e;j++)
        {
                if(i==j)
                c[i][j]=0;
                else
                c[i][j]=1000;
```

```
}
for(k=1;k<=e;k++)
{
        printf("\nEnter two vertices for edge - %d and its cost : ",k);
        scanf("%d",&i);
        scanf("%d",&j);
        scanf("%d",&c[i][j]);
        gr[i][j]=1;
//      gr[j][i]=1;
}
for(i=1;i<=n;i++)//print adjacency matrix
{
        printf("\n");
        for(j=1;j<=n;j++)
        printf("%d   ",gr[i][j]);
}
printf("\nCost matrix is ");//print cost adjacency matrix
for(i=1;i<=n;i++)
{
        printf("\n");
        for(j=1;j<=n;j++)
        printf("%d  ",c[i][j]);
}
printf("\nEnter the source vertex: ");
scanf("%d",&v);
for(i=1;i<=n;i++)
{
        dist[i]=c[v][i];//initialize the dist[] value and set s
        s[i]=0;
}
s[v]=1;//add v to set s
dist[v]=0;//distance from source to source =0
u=v;
for(k=2;k<=n;k++)
{
        min=10000;//find a vertex u not in s and dist[u] is minimum
        for(i=1;i<=n;i++)
        {
                if((s[i]!=1)&&(dist[i]<min))
                {
```

```
                        min=dist[i];
                        u=i;
                    }
                }
                s[u]=1;
                printf("\nDistance : %d--->%d is : %d",v,u,dist[u]);
                for(w=1;w<=n;w++)
                {
                    if((gr[u][w]==1)&&(s[w]!=1))
                    {
                        if(dist[w]>dist[u]+c[u][w])
                        dist[w]=dist[u]+c[u][w];
                    }
                }
        }
        return 0;
}
```

P31. Program to implement Dijkstra- Source Shortest Path (SSP) and printing paths along with it.

```
//PROGRAM FOR SOURCE SHORTEST PATH (SSP) PRINTING PATHS ALONG WITH IT.
#include<stdio.h>
void dijkshtra(int g[20][20], int cost[20][20], int vis[20], int dis[20],int parent[20],int v,int src)
{
        int min,i,j,u,k,x,path[20];
        vis[src]=1;
        dis[src]=0;
        u=src;
        for(i=2;i<=v;i++)
        {
                min=1000;// find a vertex u which is not visited and distance less than minimum
            for(j=1;j<=v;j++)
                {
                if(vis[j]==0 && dis[j]<min)
```

```
                {
                        min=dis[j];
                        u=j;
        }
        }
        vis[u]=1;
for(j=1;j<=v;j++)
        {
        if((g[u][j]==1)&&(vis[j]!=1))
                {
                        if(dis[j]>(dis[u]+cost[u][j]))
                        {
                                parent[j]=u;
                                dis[j]=dis[u]+cost[u][j];
                        }
        }
}
int a,k;
k=1;
printf("\nShortest distance from %d ---> %d is %d",src,u,dis[u]);
a=u;
                do
                {
                        a=parent[a];
                        path[k]=a;
                        k++;

                }while(a!=src);
                if(dis[a]!=1000)
                {
                        printf(" following the path : " );
                        for(x=k-1;x>=1;x--)
                        {
                                printf("%d->",path[x]);
                        }
                        printf("%d",u);
                }
                else
                 printf(" = infinity i.e. NO path");
        }
```

```
	}
int main()
{
	int
g[20][20],cost[20][20],dis[20],u,v,vis[20],e,i,j,v1,v2,n,w,src,parent[10];
	printf("**PROGRAM TO FIND SINGLE SOURCE SHORTEST
PATH IN AN GRAPH ALONG WITH ITS PATH!**");
	printf("\n****************************************
**************************************");
	printf("\nEnter the no. of vertices : ");
	scanf("%d",&v);
	printf("\nEnter the no. of edges : ");
	scanf("%d",&e);
	for(i=1;i<=v;i++)
	{
		for(j=1;j<=v;j++)
		{
			g[i][j]=0;
		}
	}
	for(i=1;i<=v;i++)
	{
		for(j=1;j<=v;j++)
		{
			if(i==j)
				cost[i][j]=0;
			else
				cost[i][j]=1000;
		}
	}
	for(i=1;i<=e;i++)
	{
		printf("\nEnter the two vertices for edge no. %d and its
weight : ",i);
		scanf("%d%d%d",&v1,&v2,&w);
		g[v1][v2]=1;
		//g[v2][v1]=1;
		cost[v1][v2]=w;
		//cost[v2][v1]=w;
	}
```

```
        printf("\nAdjacency matrix is:\n");
        for(i=1;i<=v;i++)//print adjacency matrix
        {
                for(j=1;j<=v;j++)
                {
                        printf("%d\t",g[i][j]);
                }
                printf("\n");
        }
        printf("\nCost adjacency matrix is:\n");//print cost adjacency matrix
        for(i=1;i<=v;i++)
        {
                for(j=1;j<=v;j++)
                {
                        printf("%d\t",cost[i][j]);
                }
                printf("\n");
        }
        printf("\nEnter the source vertex : ");
        scanf("%d",&src);
        for(i=1;i<=v;i++)
        {
                vis[i]=0;//making all vertices unvisited
                dis[i]=cost[src][i];
           parent[i]=src;
        }
        dijkshtra(g,cost,vis,dis,parent,v,src);
        return 0;
}
```

P32. Program to implement Threaded Binary Search Tree

```
//IMPLEMENTATION OF THREADED BINARY SEARCH TREE
#include<stdio.h>
#include<stdlib.h>
struct node
{
        struct node *left;
```

```
    struct node *right;
    int info;
    int rthread;
        int lthread;
};
struct node *root=NULL;
void maketree(int x);
void createtree(int n);
void insert(struct node *,int x);
struct node * search(struct node *root,int x);
struct node * insuccessor(struct node *x);
struct node * inpredecessor(struct node *x);
void inorder(struct node *x);
void preorder(struct node *x);
void postorder(struct node *x);
void setleft(struct node *,int );
void setright(struct node*,int );
void main()
{
        int a,n,x,c,b,ch,e;
        struct node *p;
        do
        {
                printf("\n\n\n1.Create a Tree  \t2.Insert a node  \t3.Search a
node");
                printf("\n4.Inorder Traversal \t5.Preorder Traversal
\t6.Postorder Traversal");
                printf("\n7.Inorder Predecessor \t8.Inorder Successor");
                printf("\n9.To Exit \nEnter your choice:");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1: printf("\nEnter how many nodes you want in
tree : ");
                                        scanf("%d",&n);
                                        createtree(n);
                                        printf("\nInorder traversal is : ");
                                        inorder(root);
                                        break;
                        case 2: printf("\nEnter new node value to insert : ");
```

```
                        scanf("%d",&x);
                        insert(root,x);
                        printf("\nInorder traversal is : ");
                        inorder(root);
                        break;
                case 3: printf("\nEnter to search : ");
                        scanf("%d",&x);
                        p=search(root,x);
                        if(p==NULL)
                                printf("\nNode is not present!!");
                        else
                                printf("\nNode is present!!");
                        break;
                case 4: printf("\nInorder traversal is : ");
                        inorder(root);
                        break;
                case 5: printf("\nPreorder traversal is : ");
                        preorder(root);
                        break;
                case 6: printf("\nPostorder traversal is : ");
                        postorder(root);
                        break;
                case 7: printf("\nEnter element whose predecessor you
want : ");
                        scanf("%d",&x);
                        p=search(root,x);
                        if(p==NULL)
                        printf("\nExisting node is not present whose
predecessor you want!");
                        else
                        {
                                p=inpredecessor(p);
                        if(p==NULL)
                                printf("\nNo predecessor exist!");
                        else
                                printf("\nThe predecessor of %d is
%d",x,p->info);
                        }
                        break;
```

```
                case 8: printf("\nEnter element whose successor you
want : ");
                        scanf("%d",&x);
                        p=search(root,x);
                        if(p==NULL)
                        printf("\nExisting node is not present whode
successor you want!!");
                        else
                        {
                                p=insuccessor(p);
                                if(p==NULL)
                                        printf("\nNo successor exist!");
                                else
                                        printf("\nThe successor of %d is
%d",x,p->info);
                         }
                        break;
                case 9: printf("\nThank you!!");
                        break;
                default : printf("\nWrong input!!");
        }
    }while(ch>=1 && ch<=8);
}
struct node* search(struct node *root,int x)
{
    struct node *p,*q;
    p=root;
    if(root==NULL || root->info==x) return p;
    else if(x<root->info)
    {
        if(root->lthread==1)
                return NULL;
        else
                return search(root->left,x);
    }
    else if(x>root->info)
    {
        if(root->rthread==1)
                return NULL;
        else
```

```
                return search(root->right,x);
        }
}
void maketree(int x)
{
        struct node *p;
        p=(struct node *)malloc(sizeof(struct node));
        p->info=x;
        p->left=NULL;
        p->right=NULL;
        p->rthread=1;
        p->lthread=1;
        root=p;
}
void createtree(int n)
{
        int i,x;
        printf("\nEnter the %d values : \n",n);
        for(i=0;i<n;i++)
        {
                scanf("%d",&x);
                if(i==0)
                        maketree(x);
                else
                insert(root,x);
        }
}
void insert(struct node *root,int x)
{
        if(root==NULL)
                maketree(x);
        else if(x < root->info)
        {
                if(root->lthread==1)
                        setleft(root,x);
                else
                        insert(root->left,x);

        }
        else if(x>=root->info)
```

```
        {
                if(root->rthread==1)
                        setright(root,x);
                else
                        insert(root->right,x);
        }
}
void setleft(struct node *p,int x)
{
        struct node *q,*r;
        q=(struct node *)malloc(sizeof(struct node));
        q->info=x;
        r=p->left;
        p->left=q;
        p->lthread=0;
        q->left=r;
        q->lthread=1;
        q->right=p;
        q->rthread=1;
}
void setright(struct node *p,int x)
{
        struct node *q,*r;
        q=(struct node *)malloc(sizeof(struct node));
        q->info=x;
        r=p->right;
        p->right=q;
        p->rthread=0;
        q->left=p;
        q->rthread=1;
        q->right=r;
        q->lthread=1;
}
struct node* insuccessor(struct node *p)
{
        struct node *succ;
        succ=p->right;
        if(p->rthread==0)
        {
            while(succ->lthread==0)
```

```
            succ=succ->left;
        }
        return succ;
}
struct node* inpredecessor(struct node *p)
{
        struct node *pred;
        pred=p->left;
        if(p->lthread==0)
        {
        while(pred->rthread==0)
          pred=pred->right;
        }
        return pred;
}
void preorder(struct node *p)
{
        struct node *q;
        do
        {
                q=p;
                while((p!=NULL)&&(p->lthread==0))
                {
                        printf("%d\t",p->info);
                        p=p->left;
                   q=p;
                }
                if(q!=NULL)
                {
                        printf("%d\t",q->info);
                        p=q->right;
                        while((p!=NULL)&&(q->rthread==1))
                        {
                                q=p;
                                p=p->right;
                        }
                }
        }while(p!=NULL);
}
void inorder( struct node *p)
```

```
{
	struct node *q;
	do
	{
		q=p;
		while((p!=NULL)&&(p->lthread==0))
		{
		 p=p->left;
		 q=p;
		}
		if(q!=NULL)
		{
			printf("%d\t",q->info);
			p=q->right;
			while((p!=NULL)&&(q->rthread==1))
			{
				printf("%d\t",p->info);
				q=p;
				p=p->right;
			}
		}
	}while(p!=NULL);
}
void postorder(struct node *p)
{
	struct node *q;
	do
	{
		q=p;
		while((p!=NULL)&&(p->lthread==0))
		{
		 p=p->left;
		 q=p;
		}
		printf("%d\t",q->info);
		if(q!=NULL)
		{

			p=q->right;
			while((p!=NULL)&&(q->rthread==1))
```

```
                {
                        q=p;
                        p=p->right;

                }printf("%d\t",p->info);
        }
    }while(p!=NULL);
}
```

P33. Program to implement All Pair Shortest Path Floyd Warshall.

```
//ALL PAIR SHORTESTPATH FLOYD WARSHALL
#include<stdio.h>
void FloydWarshal(int cost[10][10],int n);
int main()
{
    int i,j,n,e,ch,v1,v2,a[10][10],w;
    printf("\nEnter the no. of vertices in the graph");
    scanf("%d",&n);
printf("Graph is 1]DIRECTED 2]UNDIRECTED\nEnter your choice:
");
    scanf("%d",&ch);
    for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
    {
        if(i==j)
        a[i][j]=0;
        else
        a[i][j]=99;
    }
    switch(ch)
    {
        case 1: printf("\nEnter the no. of edges in the directed
 graph");
                scanf("%d",&e);
                printf("\nEnter the pair of vertices having edge
between and their cost: \n");
```

```
                        for(i=1;i<=e;i++)
                        {
                                scanf("%d%d%d",&v1,&v2,&w);
                                  a[v1][v2]=w;
                        }
                        break;
                case 2:     printf("\nEnter the no. of edges in the undirected
graph");
                        scanf("%d",&e);
                        printf("\nEnter the pair of vertices having edge
between and their cost: \n");
                        for(i=1;i<=e;i++)
                        {
                                scanf("%d%d%d",&v1,&v2,&w);
                                  a[v1][v2]=w;
                                  a[v2][v1]=w;
                        }
                        break;
                default: printf("\nEnter correct choice ");
        }
        printf("\nCOST Matrix of graph :\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        printf("%d\t",a[i][j]);
                }
                printf("\n");
        }
        FloydWarshal(a,n);
        return 0;
}
void FloydWarshal(int cost[10][10],int n)
{
        int a[10][10],i,j,k;
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        a[i][j]=cost[i][j];
```

```
            }
        }
        for(k=1;k<=n;k++)
        {
            for(i=1;i<=n;i++)
            {
                for(j=1;j<=n;j++)
                {
                    if(a[i][k]+a[k][j] < a[i][j])
                    a[i][j]=a[i][k]+a[k][j];
                }
            }
        }
        printf("\nThe solution is \n");
        for(i=1;i<=n;i++)
        {
            printf("\n");
            for(j=1;j<=n;j++)
            printf("%d\t",a[i][j]);
        }
}
```

P34. Program to implement N- Queens Problem.

```
//N-QUEEN PROBLEM
#include<stdio.h>
#include<stdlib.h>
int x[10],n;
void nqueen(int k);
int place(int k,int i);
main()
{
    int i;
    printf("\nEnter no of queens:");
    scanf("%d",&n);
    for(i=1;i<=n;i++)
    x[i]=0;
    nqueen(1);
```

```
}

void nqueen(int k)
{
	int i,j;
	static int ns;
	for(i=1;i<=n;i++)
	{
		if(place(k,i))
		{
			x[k]=i;
			if(k==n)
			{
				ns++;
				printf("\nThe solution-%d:\n",ns);
				for(j=1;j<=n;j++)
				printf("x[%d]=%d\t",j,x[j]);
			}
			else
				nqueen(k+1);
		}
	}
}
int place(int k,int i)
{
	int j;
	for(j=1;j<=k-1;j++)
	{
		if(x[j]==i||(abs(k-j))==abs(i-x[j]))
		return 0;
	}
	return 1;
}
```

P35. Program to implement Sum of Subset Problem

```
//SUM OF SUBSETS PROBLEM
#include <stdio.h>
int w[10],x[10],w1[10],m,n;
void sumofsubset(int s,int k,int r);
int main()
{
	//arrange the elements in increasing order
	int i,j,r=0,temp;
	printf("\nEnter the number of elements in the set: ");
	scanf("%d",&n);
	printf("Enter the value to the set: ");
	for(i=1;i<=n;i++)
	{
		scanf("%d",&w[i]);
		w1[i]=w[i];
		r+=w[i];
	}

	//sorting subset
	for(i=1;i<=n-1;i++)
	{
		for(j=1;j<=n-i;j++)
		{
			if(w1[j]>w1[j+1])
			{
				temp=w1[j];
				w1[j]=w1[j+1];
				w1[j+1]=temp;
			}
		}
	}
	printf("\nEnter the value of M: ");
	scanf("%d",&m);
	for(i=1;i<=n;i++)
		x[i]=0;
	sumofsubset(0,1,r);
	return 0;
}
```

```
void sumofsubset(int s,int k,int r)
{
	int i,j,l;
	//generating left child
	x[k]=1;
	if(s+w1[k]==m)
	{
		printf("\nSolution subset\n");
		for(l=k+1;l<=n;l++)
			x[l]=0;
		for(i=1;i<=n;i++)
		{
			for(j=1;j<=n;j++)
			{
				if(w[i]==w1[j])
					printf("%d\t",x[j]);
			}
		}
	}
	else if(s+w1[k]+w[k+1]<=m)
		sumofsubset(s+w[k],k+1,r-w1[k]);

	//generating right child
	if((s+r-w1[k]>=m) && (s+w1[k+1]<=m))
	{
		x[k]=0;
		sumofsubset(s,k+1,r-w1[k]);
	}
}
```

P36. Program to implement Hamiltonian Cycle

```
//HAMILTONIAN CYCLE
#include<stdio.h>
#include<stdlib.h>
void hamiltonion(int k);
void next_value(int k);
int x[10],a[10][10],n, flag=0;
```

```
int main()
{
	int i,j,e,ch,v1,v2;
	printf("\nEnter the number of vertices in the graph: ");
	scanf("%d",&n);
	printf("\nGraph is 1.Directed  2.Undirected\nEnter your choice: ");
	scanf("%d",&ch);
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		a[i][j]=0;
	}
	switch(ch)
	{
		case 1: printf("\nEnter the number of edges in the directed
graph: ");
				scanf("%d",&e);
				printf("\nEnter the pair of vertices having edge
b/w them");
				for(i=1;i<=e;i++)
				{
					printf("\nEnter the two vertices having edge-
%d : ",i);
					scanf("%d",&v1);scanf("%d",&v2);
					a[v1][v2]=1;
				}
				break;
		case 2: printf("\nEnter the number of edges in the undirected
graph: ");
				scanf("%d",&e);
				printf("\nEnter the pair of vertices having edge
b/w them");
				for(i=1;i<=e;i++)
				{
					printf("\nEnter the two vertices having edge-
%d : ",i);
					scanf("%d",&v1);scanf("%d",&v2);
					a[v1][v2]=1;
					a[v2][v1]=1;
				}
```

```
                        break;
                default: printf("\nPlease enter correct choice!");
        }
        printf("\nMatrix representation of graph: \n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        printf("%d\t",a[i][j]);
                }
                printf("\n");
        }
        for(i=1;i<=n;i++)
                x[i]=0;
        x[1]=1;
        hamiltonion(2);
        if(flag==0)
        {
                printf("\nNo Hamiltonion Cycle exists.\n");
        }
        return 0;
}
//Hamiltonion function
void hamiltonion(int k)
{
        int i; static int count;
        do
        {
                next_value(k);          //return next possible kth vertex in hc
                if(x[k]==0)
                return;
                if(k==n)
                {
                        count++;
                        flag=1;
                        printf("\nHamiltonion Cycle - %d \n",count);
                        for(i=1;i<=n;i++)
                                printf("%d\t",x[i]);
                }
                else
```

```
                hamiltonion(k+1);
            }while(k<=n);
        }
//NextValue function to find next vertex in HC
void next_value(int k)
{
        int j;
        do
        {
            x[k]= (x[k]+1)%(n+1);
            if(x[k]==0)
                return;
            if(a[x[k-1]][x[k]]==1)
            {
                for(j=1;j<=k;j++)
                {
                    if(x[k]==x[j])
                        break;
                }
                if(j==k)
                {
                    if((k<n)||((k==n) && (a[x[k]][x[1]]==1)))
                        return;
                }
            }
        }while(x[k]!=0);
}
```

P37. Program to implement 0/1 Knapsack problem.

```
//0/1 KNAPSACK PROBLEM
#include<stdio.h>
#define MAX 20
float bound(float w[],float p[],int n,float m,float cp,float cw,int k)
{
        float np,nw;
        int i;
        np=cp;
```

```
        nw=cw;
        for(i=k+1;i<=n;i++)
        {
                nw=nw+w[i];
                if(nw<m)
                        np=np+p[i];
                else
                        return (np+(1-(nw-m)/w[i])*p[i]);
        }
        return np;
}
void bknap(float w[],float p[],int x[],int y[],float m,int n,int k,float
cp,float cw,float *fw,float *fp)
{
        int j;
        if(cw+w[k]<=m)
        {
                y[k]=1;
                if(k<n)
                {
                        bknap(w,p,x,y,m,n,k+1,cp+p[k],cw+w[k],fw,fp);
                }
                if((cp+p[k]>*fp)&&(k==n))
                {
                        *fp=cp+p[k];
                        *fw=cw+w[k];

                        for(j=1;j<=k;j++)
                        {
                                x[j]=y[j];
                        }
                }
        }
        if(bound(w,p,n,m,cp,cw,k)>=*fp)
        {
                y[k]=0;
                if(k<n)
                        bknap(w,p,x,y,m,n,k+1,cp,cw,fw,fp);
                if((cp>*fp)&&(k==n))
                {
```

```
                        *fp=cp;
                        *fw=cw;
                        for(j=1;j<=k;j++)
                                x[j]=y[j];

                }
        }
}
int main()
{
        int i,n,x[MAX],y[MAX];
        float w[MAX],p[MAX],m,fp=0.0f,fw=0.0f;
        printf("Enter number of Objects you want: ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
           printf("\nEnter Weight and profit for object%d:\n",i);
           scanf("%f %f",&w[i],&p[i]);
        }
        printf("\nEnter Capacity of Knapsack: ");
        scanf("%f",&m);
        for(i=1;i<=n;i++)
        {
                x[i]=y[i]=0;
        }
        bknap(w,p,x,y,m,n,1,0,0,&fw,&fp);
        printf("\nOPTIMAL SOLUTION: \n");
        for(i=1;i<=n;i++)
        {
                printf("%d  ",x[i]);
        }
        printf("\nFinal Weight: %0.2f",fw);
        printf("\nFinal Profit: %0.2f",fp);
        return 0;
}
```

P38. Program to implement M colouring problem.

```
//M COLOURING PROBLEM
#include<stdio.h>
#include<stdlib.h>
int g[10][10],n,m,x[10];
void next_value(int k);
void mcolouring(int k);
//main function
main()
{
	int i,j,e,ch,v1,v2;
	printf("\nEnter the number of vertices in the graph: ");
	scanf("%d",&n);
	printf("\nGraph is 1.Directed  2.Undirected\nEnter your choice: ");
	scanf("%d",&ch);
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		g[i][j]=0;
	}
	switch(ch)
	{
		case 1:	printf("\nEnter the number of edges in the directed graph: ");
				scanf("%d",&e);
				printf("\nEnter the pair of vertices having edge between them: ");
				for(i=1;i<=e;i++)
				{
					printf("\nEnter the two vertices having edge-%d : ",i);
					scanf("%d",&v1);scanf("%d",&v2);
					g[v1][v2]=1;
				}
				break;
		case 2: printf("\nEnter the number of edges in the undirected graph: ");
				scanf("%d",&e);
```

```
                        printf("\nEnter the pair of vertices having edge
between them: ");
                        for(i=1;i<=e;i++)
                        {
                                printf("\nEnter the two vertices having edge-
%d : ",i);
                                scanf("%d",&v1);scanf("%d",&v2);
                                g[v1][v2]=1;
                                g[v2][v1]=1;
                        }
                        break;
                default:printf("\nPlease enter correct choice!");
                        return 0;
        }
        printf("\nAdjacency Matrix representation of graph: \n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        printf("%d\t",g[i][j]);
                }
                printf("\n");
        }
        printf("\nEnter the number of possible colours(m): ");
        scanf("%d",&m);
        for(i=1;i<=n;i++)
                x[i]=0;
        mcolouring(1);
}
//function mcolouring
void mcolouring(int k)
{
        int i; static int count=0;
        do
        {
                next_value(k);          //return next possible kth vertex in hc
                if(x[k]==0)
                        return;
                if(k==n)
                {
```

```
                count++;
                printf("\nSolution-%d\n",count);
                for(i=1;i<=n;i++)
                        printf("%d\t",x[i]);
        }
        else
                mcolouring(k+1);
    }while(k<=n);
}
//next value function
void next_value(int k)
{
    int j;
    do
    {
        x[k]=(x[k]+1)%(m+1);
        if(x[k]==0)
             return ;
        for(j=1;j<=n;j++)
        {
            if((g[k][j]==1)&&(x[j]==x[k]))
                break;
        }
        if(j==n+1)
            return ;
    } while(x[k]!=0);
}
```