DESIGN AND ANALYSIS ALGORITHMS (DAA)

PRACTICAL FILE (LAB RECORD)

Name: Mansi Sharma
Rollno: 1913018
ID NO: BTBTC19134
CLASS: BTECH 2YR 4SEM
BRANCH : CS
SUBJECT : DAA FILE
Programs : 40

PROGRAM INDEX:

1	10/01/21	Write a Menu driven program in C to create max heap
2	11/1/21	Write a program in C to create min heap
3	17/1/21	Write a program in c to perform heapsort
4	17/1/21	Write a program in c to implement Linear Search
5	18/1/21	Write a program in c to implement linear search using recursion
6	18/1/21	Write a program in c to implement binary search
7	18/1/21	Write a program in c to implement binary search using recursion
8	18/1/21	Write a program to implement bubble sort
9	18/1/21	Write a program to implement selection sort
10	18/1/21	Write a program to implement insertion sort
11	18/1/21	Write a program to implement tower of hanoi
12	24/1/21	Write a program to implement merge sort
13	25/1/21	Write a program to implement quicksort (first element as pivot)
14	25/1/21	Write a program to implement quicksort (last element as pivot)
15	25/1/21	Write a program to implement quicksort using HOARE partition (first element as pivot)
16	25/1/21	Write a program to implement quicksort using HOARE partition (last element as pivot)
17	25/1/21	Write a program to find kth smallest element in an array without sorting it
18	25/1/21	Write a program to find kth largest element in an array without sorting it

19	08/2/21	Write a program to implement simple union and simple find
20	08/2/21	Write a program to implement weighted union and collapsing find
21	08/2/21	Write a program to find adjacency matrix representation of a graph
22	08/2/21	Write a program to find adjacency list representation of a graph
23	21/2/21	Write a program to implement breadth first search on a graph
24	21/2/21	Write a program to implement breadth first traversal on a graph
25	21/2/21	Write a program to implement Depth first search on a graph
26	21/2/21	Write a program to implement Depth first search on a graph using linked list representation
27	21/2/21	Write a program to implement Depth first traversal on a graph
28	22/2/21	Write a program to implement fractional knapsack using greedy method
29	22/2/21	Write a program to implement job sequencing using greedy method
30	28/2/21	Write a program to find minimum spanning tree of a graph using prims algorithm
31	28/2/21	Write a program to find minimum spanning tree of a graph using kruskals algorithm
32	01/3/21	Write a program to find single source shortest path of all vertices using dijkstra's algorithm
33	01/3/21	Write a program to find single source shortest path of all vertices using dijkstra's algorithm and printing the path followed
34	7/3/21	Write a program to find single source shortest path of all vertices using bellmanford's algorithm
35	08/3/21	Write a program to find all pair shortest path of all vertices using Floydwarshall's algorithm
36	15/3/21	Write a program to implement threaded binary search tree

37	12/4/21	Write a program to implement N queens problem
38	12/4/21	Write a program to implement sum of subsets problem
39	19/4/21	Write a program to implement Graph colouring problem
40	19/04/21	Write a program to implement Hamiltonian cycle using backtracking

.....PROGRAMS.....

1: Write a Menu driven program in C to create max heap

```
// Max heap Program
//MANSI SHARMA ROLLNO-1913018
#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);
}

// Insert element in Max-Heap
```

```
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;

}

 // Delete an element from Max-Heap
 int delmax(int a[20], int n)
 {
        int x;
        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;
 }
int main()
{  int n, i, a[20],item,ch;
  do
  {
        printf("\n 1. create maxheap 2. insert element 3. delete element 4. Exit");
        printf("\n Enter your choice:"); scanf("%d",&ch);
        switch(ch)
         {
           case 1:  printf("\n enter the size of array:"); scanf("%d",&n);
                                printf("\n enter %d elements",n);
                                for(i=1;i<=n;i++)  scanf("%d",&a[i]);
                                printf("\n Array elements are\n");
                                 for(i=1;i<=n;i++)  printf("%d\t",a[i]);
                                create_maxheap(a,n);
                                 printf("\n Max heap elements are\n");
                                for(i=1;i<=n;i++)  printf("%d\t",a[i]);
                                break;
```

```
                case 2: printf("\n enter element to insert:");
                                scanf("%d",&item);
                                n=n+1;  a[n]=item;
                                insertintomaxheap(a,n);
                                printf("\n Max heap after inserting elements %d are\n",item);
                                for(i=1;i<=n;i++)  printf("%d\t",a[i]);
                                break;
                case 3: item=delmax(a,n);
                                if(item != 0)
                                        printf("\n deleted element is =%d",item);
                                printf("\n Max heap after deletion\n ");
                                for(i=1;i<=n-1;i++)  printf("%d\t",a[i]);
                                break;
                case 4: printf("\n Thank You");
                }
        }   while (ch>=1 && ch<=3);

           return 0;
           }
```

2: Write a program in C to create min heap

```
  // Min heap Program
 //MANSI SHARMA ROLLNO-1913018
#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);
```

```
}

int main()
{  int n, i, a[20];
   printf("\n enter the size of array:"); scanf("%d",&n);
   printf("\n enter %d elements",n);
   for(i=1;i<=n;i++)  scanf("%d",&a[i]);
   printf("\n Array elements are\n");
   for(i=1;i<=n;i++)  printf("%d\t",a[i]);
   create_minheap(a,n);
   printf("\n Min heap elements are\n");
   for(i=1;i<=n;i++)  printf("%d\t",a[i]);
         return 0;
}
```

3: Write a program in c to perform heapsort

```
//HEAPSORT PROGRAM
//MANSI SHARMA ,ROLLNO-1913018
#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 function
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 temp,i;
```

```
        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);
          }
}

int main()
{  int n, i, a[20],item;
   printf("\n enter the size of array:"); scanf("%d",&n);

   printf("\n enter %d elements to sort",n);
   for(i=1;i<=n;i++)     scanf("%d",&a[i]);

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

   heap_sort(a,n);

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

        return 0;
}
```

4: Write a program in c to implement Linear search

```
//LINEAR SEARCH PROGRAM
//MANSI SHARMA ROLLNO-1913018
#include<stdio.h>
int linearsearch(int a[20],int low,int high,int x)
{
        int i;
        int flag=0;
        if(low==high)
        {
                if(a[low]==x)
                  return low;
                else
                 return 0;
        }
```

```
	else
	{
		if(low<high)
		{
			for(i=low;i<=high;i++)
			{
				if(a[i]==x)
				{
					flag=1;
					return i;
				}
			}
			if(flag==0)
			   return 0;
		}
	}
}

int main()
{
	int i,a[20],x,n,pos;
	printf("\n Enter number of elements");
	scanf("\t%d",&n);
	printf("\n enter elemnts");
	for(i=1;i<=n;i++)
	{
		scanf("\n%d",&a[i]);

	}
	printf("\n enter element to search");
	scanf("\t%d",&x);
	pos=linearsearch(a,1,n,x);
	if(pos!=0)
	{
		printf("\n elemnt found at %d position",pos);
	}
	else
	printf("\n element not found !!");
}
```

5: Write a program in c to implement linear search using recursion

```
//LINEAR SEARCH USING RECURSION
//MANSI SHARMA ROLLNO-1913018
#include<stdio.h>
int linearsearch(int a[20],int low,int high,int x)
{
	if(low==high)
	{
		if(a[low]==x)
		  return low;
		else
		 return 0;
	}
	else
	{
		if(low<high)
		{
			if(a[low]==x)
			{
				return low;
			}
			else
			{
				linearsearch(a,low+1,high,x);
			}
		}
	}
}

int main()
{
	int i,a[20],x,n,pos;
	printf("\n Enter number of elements");
	scanf("\t%d",&n);
	printf("\n enter elemnts");
	for(i=1;i<=n;i++)
	{
		scanf("\n%d",&a[i]);

	}
	printf("\n enter element to search");
```

```
        scanf("\t%d",&x);
        pos=linearsearch(a,1,n,x);
        if(pos!=0)
        {
                printf("\n elemnt found at %d position",pos);
        }
        else
        printf("\n element not found !!");
}
```

6: Write a program to implement binary search

```
#include <stdio.h>
//binary search program without recursion
int main()
{
   int i, low, high, mid, n, key, array[100];
   printf("Enter number of elements : ");
    scanf("%d",&n);
   printf("Enter %d integers : ", n);
   for(i = 0; i < n; i++)
     scanf("%d",&array[i]);
   printf("Enter value to find:  ");
   scanf("%d", &key);
   low = 0;
   high = n - 1;
   mid = (low+high)/2;
   while (low <= high)
        {
      if(array[mid] < key)
         low = mid + 1;
      else if (array[mid] == key)
                {
         printf("%d found at location %d.n", key, mid+1);
         break;
      }
      else
         high = mid - 1;
         mid = (low + high)/2;
   }
   if(low > high)
      printf("Not found! %d isn't present in the list.n", key);

         return 0;}
```

7: Write a program to implement binary serach using recursion

```
#include <stdio.h>
//binary search program using recursion
int binarysearch(int a[], int low, int high, int x)
{
   int mid = (low + high) / 2;
   if (low > high) return -1;
   if (a[mid] == x)
       return mid;
   if (a[mid] < x)
     return binarysearch(a, mid + 1, high, x);
   else
     return binarysearch(a, low, mid-1, x);
}

int main()
{
  int a[100];
  int i,len, pos, search_item;

  printf("Enter the length of the array\n");
  scanf("%d", &len);

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

  printf("Enter the element to search\n");
  scanf("%d", &search_item);

  pos = binarysearch(a,0,len-1,search_item);

  if (pos < 0 )
    printf("Cannot find the element %d in the array.\n", search_item);

  else
    printf("The position of %d in the array is %d.\n", search_item, pos+1);

  return 0;
}
```

8: Write a program to implement bubble sort

```
//BUBBLE SORT PROGRAM
//MANSI SHARMA, ROLLNO-1913018
#include<stdio.h>
void bubblesort(int a[20],int n)
{
	int i,j,k,temp,flag=1;
	for(i=1; (i<=n)&& (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;
			}
		}
	}
}

int main()
{
	int a[20],n,i;
	printf("\n Enter number of elements");
	scanf(" %d",&n);
	printf("\n Enter elements :");
	for(i=1;i<=n;i++)
	{
		scanf(" %d",&a[i]);
	}
	printf("\n elements are :");
	for(i=1;i<=n;i++)
	{
		printf(" \t%d",a[i]);
	}
		bubblesort(a,n);
	printf("\n after sorting elements :");
	for(i=1;i<=n;i++)
	{
```

```
		printf(" %d\t",a[i]);
	}


}
```

9: Write a program to implement selection sort

```
//selection sort
//MANSI SHARMA, ROLLNO-1913018
#include<stdio.h>
void selectionsort(int a[20],int n)
{
	int i,j,temp;
	for(i=1; i<=n;i++)
	{
		for(j=i+1;j<=n;j++)
		{
			if(a[i]>a[j])
			{
				temp=a[i];
				a[i]=a[j];
				a[j]=temp;
			}
		}
	}
}

int main()
{
	int a[20],n,i;
	printf("\n Enter number of elements");
	scanf(" %d",&n);
	printf("\n Enter elements :");
	for(i=1;i<=n;i++)
	{
		scanf(" %d",&a[i]);
	}
	printf("\n elements are :");
	for(i=1;i<=n;i++)
	{
		printf(" \t%d",a[i]);
	}
```

```
        selectionsort(a,n);

    printf("\n after sorting elements :");
    for(i=1;i<=n;i++)
    {
        printf(" %d\t",a[i]);
    }


}
```

10: Write a program to implement insertion sort

```
//INSERTION SORT
//MANSI SHARMA, ROLLNO-1913018
#include<stdio.h>
void insertionsort(int a[20],int n)
{
    int i,j,key;
    for(i=2; i<=n;i++)
    {
        key=a[i];
       j=i-1;
        while(j>=1&&a[j]>key)
        {
            a[j+1]=a[j];
            j--;
        }
        a[j+1]=key;
    }
}

int main()
{
    int a[20],n,i;
    printf("\n Enter number of elements");
    scanf(" %d",&n);
    printf("\n Enter elements :");
    for(i=1;i<=n;i++)
    {
        scanf(" %d",&a[i]);
    }
    printf("\n elements are :");
```

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

                insertionsort(a,n);

        printf("\n after sorting elements :");
        for(i=1;i<=n;i++)
        {
                printf(" %d\t",a[i]);
        }


}
```

11: Write a program to implement tower of hanoi

```
//program for tower of hanoi
//made by mansi sharma   rollno-1913018

#include<stdio.h>
int cnt=0;
void towerofhanoi(int n,char TA, char TB,char TC)
{
        if(n>=1)
        {
                towerofhanoi(n-1,TA,TC,TB);
                printf("\n move disk- %d from %c to %c",n,TA,TB);
                towerofhanoi(n-1,TC,TB,TA);
                cnt++;
        }
}
int main()
{
        int n;
        printf("\n enter the number disk ");
        scanf("%d",&n);
        towerofhanoi(n,'A','B','C');
        return 0;

}
```

12: Write a program to implement merge sort

```
#include<stdio.h>
#include<conio.h>
void merge(int a[20],int low,int mid,int high)
{
	int b[20],j,i=low,l=low,h=mid+1;

	while(l<=mid&&h<=high)
	{
		if(a[l]<a[h])
		{
			b[i]=a[l];
			l++;
			i++;
		}
		else
		{
			b[i]=a[h];
			h++;
			i++;
		}
	}
	while(l<=mid)
	{
		b[i]=a[l];
		i++;l++;
	}
	while(h<=high)
	{
		b[i]=a[h];
		h++;i++;
	}


		for(j=low;j<=high;j++)
		{
			a[j]=b[j];
		}
}

void mergesort(int a[20],int low,int high)
{
```

```
        if(low<high)
        {
                int mid=((low+high)/2);
                mergesort(a,low,mid);
                mergesort(a,mid+1,high);
                merge(a,low,mid,high);
        }
}


int main()
{
        int a[20],n,i;
        printf("\n Enter no of elements ");
        scanf(" %d",&n);
        printf("\n enter elements\n");
        for(i=1;i<=n;i++)
        {
                scanf("%d",&a[i]);
        }

        printf("\n array before mergesort:  ");
        for(i=1;i<=n;i++)
        {
                printf("%d  ",a[i]);
    }

    printf("\n");

    mergesort(a,1,n);

    printf("\n Array after mergesort: ");
    for(i=1;i<=n;i++)
    {
        printf("%d  ",a[i]);
        }
        printf("\n");

}
```

13: Write a program to perform quicksort (last element as pivot)

```
//Quicksort program using last element as pivot
// MADE BY MANSI SHARMA, ROLLNO-1913018
#include<stdio.h>
//partition function
int partition(int a[20],int low,int high)
{
	int x=a[high],i=low-1,j;
	for( j=low;j<=high-1;j++)
	{
		if(a[j]<=x)
		{

			i=i+1;
			if(i!=j)
			{
				int temp;
				temp=a[i];
				a[i]=a[j];
				a[j]=temp;
			}

		}
	}
		a[high]=a[i+1]; //swapping a[i+1] with a[high]
		a[i+1]=x;
		return(i+1);


}
//quicksort function
void quicksort(int a[20],int low,int high)
{
	if(low<high)
	{
		int mid,q;
		mid=((low+high)/2);
		q=partition(a,low,high);
		quicksort(a,low,q-1);
		quicksort(a,q+1,high);

	}
```

```
}

int main()
{
        int a[20],n,i;
        printf("\n Enter no of elements ");
        scanf(" %d",&n);
        printf("\n enter elements\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }

        printf("\n array before quicksort:  ");
        for(i=0;i<n;i++)
        {
                printf("%d  ",a[i]);
   }

   printf("\n");

   quicksort(a,0,(n-1));

   printf("\n after quicksort: ");
   for(i=0;i<n;i++)
   {
        printf("%d  ",a[i]);
        }
        printf("\n");

}
```

14: Write a program to perform quicksort (first element as pivot)

```
//Quicksort program using first element as pivot
// MADE BY MANSI SHARMA, ROLLNO-1913018
#include<stdio.h>
//partition function
int partition(int a[20],int low,int high)
{
        int x=a[low],i=low+1,j;
        for( j=low+1;j<=high;j++)
        {
```

```
                if(a[j]<=x)
                {
                        int temp;
                        temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
                        i++;
                }

    }
                a[low]=a[i-1]; //swapping a[i-1] with a[low]
                a[i-1]=x;
                return(i-1);


}
//quicksort function
void quicksort(int a[20],int low,int high)
{
        if(low<high)
        {
                int mid,q;
                mid=((low+high)/2);
                q=partition(a,low,high);
                quicksort(a,low,q-1);
                quicksort(a,q+1,high);

        }
}

int main()
{
        int a[20],n,i;
        printf("\n Enter no of elements ");
        scanf(" %d",&n);
        printf("\n enter elements\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }

        printf("\n array before quicksort:  ");
        for(i=0;i<n;i++)
        {
```

```
            printf("%d  ",a[i]);
    }

    printf("\n");

    quicksort(a,0,(n-1));

    printf("\n after quicksort: ");
    for(i=0;i<n;i++)
    {
        printf("%d  ",a[i]);
        }
        printf("\n");

}
```

15: Write a program to perform quicksort using hoare partition (first element as pivot)

```
//Quicksort program using hoare partition(first element as pivot)
// MADE BY MANSI SHARMA, ROLLNO-1913018

#include<stdio.h>

//partition function
int partition(int a[20],int low,int high)
{
        int pivot=a[low],temp,i=low,j=high;
        while(true)
        {
                while(a[i]<pivot)
                {
                        i++;
                }
                while(a[j]>pivot)
                {
                        j--;
                }
                if(i>=j)
                {
                        return j;
                }
                else
```

```
                {
                  temp=a[i];
                  a[i]=a[j];
                  a[j]=temp;
                }
    }
}

//quicksort function
void quicksort(int a[20],int low,int high)
{
        if(low<high)
        {
                int mid,q;
                mid=((low+high)/2);
                q=partition(a,low,high);
                quicksort(a,low,q-1);
                quicksort(a,q+1,high);

        }
}

//main function
int main()
{
        int a[20],n,i;
        printf("\n Enter no of elements ");
        scanf(" %d",&n);
        printf("\n enter elements\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("\n array before quicksort:  ");
        for(i=0;i<n;i++)
        {
                printf("%d  ",a[i]);
    }
    printf("\n");
    quicksort(a,0,(n-1));
    printf("\n after quicksort: ");
    for(i=0;i<n;i++)
    {
        printf("%d  ",a[i]);
```

```
        }
        printf("\n");

}
```

16: Write a program to perform quicksort using hoare partition (last element as pivot)

```
//quicksort using hoare partition (last element as pivot)
//made by mansi sharma  rollno-1913018
#include<stdio.h>
//partition function
int partition(int a[20],int low,int high)
{

        int pivot=a[high];
        int temp;
        int i=low-1;
        int j=high+1;
   while(true)
   {
        do
        {
                i=i+1;
                }while(a[i]<pivot);

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

                if(i>=j)
                {
                        return j;
                }
                else
                {
                        temp=a[i];
                        a[i]=a[j];
                        a[j]=temp;
                }
        }

}
```

```
//quicksort function
void quicksort(int a[20],int low,int high)
{
        if(low<high)
        {
                int q,temp;
                temp=a[low];
                a[low]=a[high];
                a[high]=temp;
           q=partition(a,low,high);
                quicksort(a,low,q);
                quicksort(a,q+1,high);

        }
}

//main function
int main()
{
        int a[20],n,i;
        printf("\n Enter no of elements ");
        scanf(" %d",&n);
        printf("\n enter elements\n");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
        printf("\n array before quicksort:  ");
        for(i=0;i<n;i++)
        {
                printf("%d  ",a[i]);
   }
   printf("\n");
   quicksort(a,0,(n-1));
   printf("\n after quicksort: ");
   for(i=0;i<n;i++)
   {
        printf("%d  ",a[i]);
        }
        printf("\n");

}
```

17: Write a program to find kth smallest element in array without sorting it

```
//program to find kth smallest element in array without sorting it!!
#include<stdio.h>
// Partition function
int partition(int arr[], int l, int r)
{
   int x = arr[r],j,temp, i = l;
   for ( j = l; j <= r - 1; j++) {
      if (arr[j] <= x) {
         temp=arr[i];
                        arr[i]=arr[j];
                        arr[j]=temp;
         i++;
      }
   }
   temp=arr[i];
        arr[i]=arr[r];
        arr[r]=temp;
   return i;
}
// This function returns k'th smallest element in arr[l..r]
int kthSmallest(int arr[], int l, int r, int k)
{

   if (k > 0 && k <= r - l + 1) {
      int pos = partition(arr, l, r);

      if (pos - l == k - 1)
         return arr[pos];

      if (pos - l > k - 1)
         return kthSmallest(arr, l, pos - 1, k);

      return kthSmallest(arr, pos + 1, r, k - pos + l - 1);
   }

}

// main function
int main()
{
   int a[20],i,n,k,ans;
```

```
        printf("\n enter number of elements");
        scanf("%d",&n);
        printf("\n enter number of elements\n ");
        for(i=0;i<n;i++)
        {
                scanf("%d",&a[i]);
        }
    printf("\n enter value of k ");
    scanf("%d",&k);
        if(k>n)
        {
                printf("\n Value of k is out of range  !!");
        }
        else
        {
                ans=kthSmallest(a, 0, n - 1, k);
        }
        printf("  K'th smallest element is : %d",ans );
        printf("\n");
        for(i=0;i<n;i++)
        {
                printf("%d\t",a[i]);
        }
    return 0;
}
```

18: Write a program for finding Kth Largest Element in an array

```
// program for finding Kth Largest Element
//MANSI SHARMA ROLLNO-1913018
#include<stdio.h>
//max-heapify function
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);
}

 // Delete an element from Max-Heap
 int delmax(int a[20], int n)
 {
        int x;
        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;
 }
//finding kth largest element
 void kthlargest(int k,int a[20],int n)
 {
        int i,b[20];
        for(i=1;i<=k;i++)
        {
                b[i]=delmax(a,n);
        }
        printf("\n %d largest element is = %d",k,b[k]);
 }
//main function
int main()
{  int n, i, a[20],k;
printf("\n enter the size of array:");
scanf("%d",&n);
printf("\n enter %d elements",n);
for(i=1;i<=n;i++)
scanf("%d",&a[i]);
printf("\n Array elements are\n");
```

```
for(i=1;i<=n;i++)
printf("%d\t",a[i]);
create_maxheap(a,n);
printf("\n Enter the value of k ");
scanf("%d",&k);
kthlargest(k,a,n);
   return 0;
}
```

19: Write a program to implement simple union and simple find

```
//simple find and simple union program
#include<stdio.h>
int n,p[20];
void setunion(int i,int j)
{
        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("\n enter the value of n ");
        scanf("%d",&n);
        //set each element in their own set p[i]=-1
        for(i=1;i<=n;i++)
        {
                p[i]=-1;
        }
        printf("\n p[]= ");
        for(i=1;i<=n;i++)
        {
```

```
                printf("%d\t",p[i]);
        }
        printf("\n ele[]= ");
                for(i=1;i<=n;i++)
        {
                printf("%d\t",i);
        }
        do
        {
                printf("\n 1:UNION \n 2:FIND \n 3:EXIT");
                printf("\n enter ur choice ");
                scanf("%d",&ch);
                switch(ch)
                {
                        case 1:printf("\n enter the root of two trees ");
                              scanf("%d%d",&r1,&r2);
                              if((p[r1]==-1 )&& (p[r2]==-1))
                              {
                                 setunion(r1,r2);
                                   }
                                   break;
                        case 2:printf("\n enter the element to find ");
                              scanf("%d",&i);
                              j=sfind(i);
                              printf("\n the element %d is in the tree whose root is %d ",i,j);
                              break;
                    case 3:printf("\n THANK YOU!!");
                          break;
                    default:printf("\n enter the valid choice ");
                }
                        printf("\n p[]= ");
               for(i=1;i<=n;i++)
               {
                       printf("%d\t",p[i]);
                }
                printf("\n ele[]= ");
                      for(i=1;i<=n;i++)
                {
                        printf("%d\t",i);
                }

        }while(ch<=3&&ch>=1);
return 0;
}
```

20: Write a program to implement weighted union and collapsing find

```
//program to find weighted union and collapsing find
#include<stdio.h>
int n,p[20];
//weighted union
void wunion(int i,int j)
{
        int temp;
        if((p[i]>0)||(p[j]>0))
          printf("\n Invalid roots");
        else
        {
                temp=p[i]+p[j];
                if(p[i]>p[j])
                {
                        p[i]=j;
                        p[j]=temp;
                }
                else
                {
                        p[j]=i;
                        p[i]=temp;
                }
        }
}
//collapsing find
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;
```

```
}
//main function
int main()
{
        int i,j,r1,r2,ch;
    printf("\n enter the value of n ");
    scanf("%d",&n);
    //set each element in their own set p[i]=-1
    for(i=1;i<=n;i++)
    {
        p[i]=-1;
    }
    printf("\n p[]= ");
    for(i=1;i<=n;i++)
    {
        printf("%d\t",p[i]);
    }
    printf("\n ele[]= ");
        for(i=1;i<=n;i++)
    {
        printf("%d\t",i);
    }
    do
    {
        printf("\n 1:WUNION \n 2:COLLAPSINGFIND \n 3:EXIT");
        printf("\n enter ur choice ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:printf("\n enter the root of two trees ");
                scanf("%d%d",&r1,&r2);
                if(p[r1])
                    wunion(r1,r2);
                      break;
            case 2:printf("\n enter the element to find ");
                scanf("%d",&i);
                j=collapsingfind(i);
                printf("\n the element %d is in the tree whose root is %d ",i,j);
                break;
           case 3:printf("\n THANK YOU!!");
                 break;
           default:printf("\n enter the valid choice ");
        }
            printf("\n p[]= ");
```

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

        }while(ch<=2&&ch>=1);

}
```

21: Write a program to print adjacency matrix of a graph

```
//program to print adjacency matrix of a graph
#include<stdio.h>
void printmatrix(int a[10][10],int r)
{
        int i,j;
        printf("\n Adjacency matrix of size %d*%d\n",r,r);
        for(i=1;i<=r;i++)
        {
                for(j=1;j<=r;j++)
                {
                        printf("%d ",a[i][j]);
                }
                printf("\n");
        }

}
int main()
{
        int i,j,r,c,nv,ne,mat[10][10],v1,v2;
        printf("\n enter number of vertices and edges in graph ");
        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("\n enter the two end vertices ");
          scanf("%d%d",&v1,&v2);
          mat[v1][v2]=1;
          mat[v2][v1]=1;
   }
   printmatrix(mat,nv);

}
```

22: Write a program to print adjacency list representation of a graph

```
#include<stdio.h>
#include<stdlib.h>
//structure to represent adjacency list node
struct AdjListNode
{
    int value;
    struct AdjListNode* next;
};
// A structure to represent an adjacency list
struct AdjList
{
    struct AdjListNode *head;
};
//structure to represent graph
struct Graph
{
    int V;
   struct AdjList* array;
};
//function to create a new adjacency list node
struct AdjListNode* newAdjListNode(int value)
{
    struct AdjListNode* newNode = (struct AdjListNode*) malloc(sizeof(struct AdjListNode));
    newNode->value = value;
    newNode->next = NULL;
    return newNode;
}
// function that creates a graph of V vertices
struct Graph*createGraph(int V)
```

```
{
    struct Graph* graph =(struct Graph*) malloc(sizeof(struct Graph));
    graph->V =V;
    graph->array =(struct AdjList*)malloc(V*sizeof(struct AdjList));
    int i;
    for (i = 0; i < V; ++i)
        graph->array[i].head = NULL;
    return graph;
}
//function to Add an edge to an undirected graph
void addEdge(struct Graph* graph, int i, int value)
{
    struct AdjListNode*newNode = newAdjListNode(value);
    newNode->next = graph->array[i].head;
    graph->array[i].head = newNode;
    newNode = newAdjListNode(i);
    newNode->next = graph->array[value].head;
    graph->array[value].head = newNode;
}
// function to print the adjacency list
void printGraph(struct Graph* graph)
{
    int v;
    for (v = 1; v <= graph->V; ++v)
   {
        struct AdjListNode* temp = graph->array[v].head;
        printf("\n Adjacency list of vertex %d\n head ", v);
        while (temp)
        {
            printf("-> %d", temp->value);
            temp = temp->next;
        }
        printf("\n");
    }
}
// main function
int main()
{
    int V,v1,v2,i,e ;
          printf("\n enter number of vertices ");
          scanf("%d",&V);
    struct Graph* graph = createGraph(V);
          printf("\n enter number of edges ");
          scanf("%d",&e);
```

```
        for(i=1;i<=e;i++)
        {
                printf("\n enter %d end vertices ",i);
                scanf("%d%d",&v1,&v2);
                addEdge(graph,v1,v2);
   }

printGraph(graph);
 return 0;
}
```

23: Write a program to implement Breadth first search on a graph

```
//breadth first search undirected graph
#include<stdio.h>
#define max 10
int g[10][10],q[max],vis[10];
int n,front=-1,rear=-1;
//empty q function
int emptyq()
{
        if(front==-1&&rear==-1)
        {
                return 1;
        }
        else
          return 0;
}
//insert into q
qinsert(int j)
{
        if((front==0)&&(rear==max-1))
          printf("\n OVERFLOW ");
        else
        {
                rear++;
                q[rear]=j;
                if(front==-1)
                   front=0;
        }
}
//delete from q
int delq()
```

```
{
        int d;
        if(front==-1&&rear==-1)
          d=0;
        else
        {
                d=q[front];
                if(front==rear)
                {
                        front=-1;
                        rear=-1;
                }
                else
                   ++front;
        }
        return d;
}
//BFS function
bfs(int v,int n)
{
        int i,w;
        for(i=1;i<=n;i++)
          vis[i]=0;
        printf("\n traversal from vertex %d\n",v);
        printf("\t%d",v);
        vis[v]=1;
        qinsert(v);
        while(!emptyq())
        {
                v=delq();
                for(w=1;w<=n;w++)
                {
                        if(g[v][w]==1&&vis[w]==0)
                        {
                                printf("\t%d",w);
                                vis[w]=1;
                                qinsert(w);
                        }
                }
        }
}
int main()
{
        int i,j,v,e,k,v1,v2;
```

```
        printf("\n enter total no. of vertices ");
        scanf("%d",&n);
        printf("\n enter total number of edges ");
        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("\n enter end vertices for edge-%d",k);
                scanf("%d",&v1);
                scanf("%d",&v2);
                g[v1][v2]=1;
                g[v2][v1]=1;
        }
        //printing adjacency matrix
          for(i=1;i<=n;i++)
          {
                  printf("\n\n");
                  for(j=1;j<=n;j++)
                  {
                          printf("\t%d",g[i][j]);
                  }
          }
        printf("\n enter starting vertex ");
        scanf("%d",&v);
        bfs(v,n);
        return 0;

}
```

24: Write a program to implement breadth first traversal of a graph

```
//BREADTH FIRST TRAVERSAL
#include<stdio.h>
#define max 10
void bft(int n);
int g[10][10],q[max],vis[10];
int n,front=-1,rear=-1;
```

```
//empty q function
int emptyq()
{
        if(front==-1&&rear==-1)
        {
                return 1;
        }
        else
          return 0;
}
//insert into q
qinsert(int j)
{
        if((front==0)&&(rear==max-1))
          printf("\n OVERFLOW ");
        else
        {
                rear++;
                q[rear]=j;
                if(front==-1)
                   front=0;
        }
}
//delete from q
int delq()
{
        int d;
        if(front==-1&&rear==-1)
          d=0;
        else
        {
                d=q[front];
                if(front==rear)
                {
                        front=-1;
                        rear=-1;
                }
                else
                   ++front;
        }
        return d;
}
//BFS function
void bfs(int v)
```

```
{
        int i,w;
        for(i=1;i<=n;i++)
          vis[i]=0;
        printf("\n traversal from vertex %d\n",v);
        printf("\t%d",v);
        vis[v]=1;
        qinsert(v);
        while(!emptyq())
        {
                v=delq();
                for(w=1;w<=n;w++)
                {
                        if(g[v][w]==1&&vis[w]==0)
                        {
                                printf("\t%d",w);
                                vis[w]=1;
                                qinsert(w);
                        }
                }
        }
}
int main()
{
        int i,j,v,e,k,v1,v2;
        printf("\n enter total no. of vertices ");
        scanf("%d",&n);
        printf("\n enter total number of edges ");
        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("\n enter end vertices for edge-%d",k);
                scanf("%d",&v1);
                scanf("%d",&v2);
                g[v1][v2]=1;
                g[v2][v1]=1;
        }
```

```
        //printing adjacency matrix
          for(i=1;i<=n;i++)
          {
                  printf("\n\n");
                  for(j=1;j<=n;j++)
                  {
                          printf("\t%d",g[i][j]);
                  }
          }
        printf("\n Breadth First Traversal of graph ");
        bft(n);
        return 0;

}
void bft(int n)
{
        int i;
        for(i=1;i<=n;i++)
        {
                vis[i]=0;
        }
        for(i=1;i<=n;i++)
        {
                if(vis[i]==0)
                {
                        bfs(i);
                }
        }
}
```

25: Write a program to implement depth first search

```
//Depth First search program
#include<stdio.h>
#define max 10
int g[10][10],vis[10],n;
//depth first search
void dfs(int v)
{
        int i,w;
        printf("%d\t",v);
        vis[v]=1;
        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("\n enter total no. of vertices ");
        scanf("%d",&n);
        printf("\n enter total number of edges ");
        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("\n enter end vertices for edge-%d",k);
                scanf("%d",&v1);
                scanf("%d",&v2);
                g[v1][v2]=1;
                g[v2][v1]=1;
        }
        //printing adjacency matrix
          for(i=1;i<=n;i++)
          {
                  printf("\n");
                  for(j=1;j<=n;j++)
                  {
                          printf("\t%d",g[i][j]);
                  }
          }
          printf("\n");
        for(i=1;i<=n;i++)
          vis[i]=0;
        printf("\n enter the starting vertex : ");
        scanf("%d",&v);
        printf("\n traversal from vertex %d\n ",v);
```

```
        dfs(v);
        return 0;

}
```

26: Write a program to implement depth first traversal of a graph

```
//depth first traversal
#include<stdio.h>
#define max 10
int g[10][10],vis[10],n;
//depth first search
void dfs(int v)
{
        int i,w;
        printf("%d\t",v);
        vis[v]=1;
        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;
        }
        for(i=1;i<=n;i++)
        {
                if(vis[i]==0)
                   dfs(i);
        }
}


int main()
{
        int i,j,v,e,k,v1,v2;
```

```
        printf("\n enter total no. of vertices ");
        scanf("%d",&n);
        printf("\n enter total number of edges ");
        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("\n enter end vertices for edge-%d",k);
                scanf("%d",&v1);
                scanf("%d",&v2);
                g[v1][v2]=1;
                g[v2][v1]=1;
        }
        //printing adjacency matrix
          for(i=1;i<=n;i++)
          {
                  printf("\n");
                  for(j=1;j<=n;j++)
                  {
                          printf("\t%d",g[i][j]);
                  }
          }
          printf("\n");
        for(i=1;i<=n;i++)
          vis[i]=0;
        printf("\n Depth First Traversal of graph:\n ");
        dft(n);
        return 0;

}
```

28: Write a program to implement fractional knapsack by 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);

void sort(struct knaps *ob,int n)
{
        int i,j;
        struct knaps temp;
        for(i=1;i<=n;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;
                        }
                }
        }
}
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+=kn[i].p;
        }
        if(i<=n)
        {
                x[i]=u/kn[i].w;
                profit+=(kn[i].p*x[i]);
```

```
        }
        for(i=1;i<=n;i++)
        {
                profit=profit+kn[i].p *x[i];
                weight=weight+kn[i].w *x[i];
        }
        printf("\n The optimal solution vector: \n");
        for(i=1;i<=n;i++)
        {
                printf("x[%d]=%.2f\t",kn[i].id,x[i]);
        }
        printf("\n The profit=%.2f and total weight=%0.2f\n",profit,weight);
}
int main()
{
        int i,j;
        struct knaps kn[10],temp;float m;
        printf("\n enter the number of elements: ");
        scanf("%d",&n);
        printf("\n enter objectid profit and weight of %d objects",n);
        for(i=1;i<=n;i++)
        {
                scanf("%d%f%f",&kn[i].id,&kn[i].p,&kn[i].w);
        }
        printf("\n enter the capacity of the knapsack: ");
        scanf("%f",&m);
        printf("\n objectid \tprofit \tweight \t profit/weight\n");
        for(i=1;i<=n;i++)
        {
                printf("%d\t%f\t%f\t %.2f\n",kn[i].id,kn[i].p,kn[i].w);
        }
        sort(kn,n);
        knapsack(kn,m);
        return 0;
}
```

29: Write a program to implement job sequencing

```
//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("\n Enter the number of jobs: ");
        scanf("%d",&n);
        printf("Enter the job_no profit and deadline of jobs %d\n",n);
        for(i=1;i<=n;i++)
          scanf("%d%d%d",&jb[i].id,&jb[i].p,&jb[i].d);
        printf("\n JobID\tProfit\tdeadline\n");
        for(i=1;i<=n;i++)
         printf("\n %d  \t%d \t  %d",jb[i].id,jb[i].p,jb[i].d);

//sorting jobs in decreasing 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("\n\n after sorting order is: \n");
        printf("\nJOBID\tPROFIT\tDEADLINE");
        for(i=1;i<=n;i++)
         printf("\n%d\t%d\t%d",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;
                }
        }
        //total profit
        for(i=1;i<=k;i++)
         tprofit+=jb[i].p;
        printf("\n subsets of jobs are : {");
        for(i=1;i<=k;i++)
        {
                printf("%d,",jb[j[i]].id);
        }
        printf("}");
}
```

30: Write a program to find minimum spanning tree using prims algorithm

```
//prims 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("\n enter number 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++)
                  g[i][j]=999;
```

```
        }
        switch(ch)
        {
                case 1:printf("\n enter number of edges in the directed graph\n");
                    scanf("%d",&e);
                    printf("\n enter the pair of vertices V1-->V2 and weight");
                    for(i=1;i<=e;i++)
                    {
                        scanf("%d%d%d",&v1,&v2,&w);
                        g[v1][v2]=w;
                        }
                        break;
                case 2:printf("\n enter the number of edges in undirected graph\t");
                    scanf("%d",&e);
                    for(i=1;i<=e;i++)
                    {
                        printf("\n Enter the pair of vertices having edge b/w them ");
                            scanf("%d%d%d",&v1,&v2,&w);
                        g[v1][v2]=w;
                        g[v2][v1]=w;
                        }
                        break;
                default:printf("\n enter correct choice");
        }
        printf("\n MATRIX 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);
}
void prims(int a[10][10],int n)
{
        int i,v1,k,v2,j,v,minmst=0,min,t[10][2],q,w;
        //finding minimum cost edge
        min=999;
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
```

```
                if(min>a[i][j])
                {
                        min=a[i][j];
                        v1=i;
                        v2=j;
                }
        }
}
t[1][1]=v1;
t[1][2]=v2;
minmst+=min;
for(i=1;i<=n;i++)
{
        if(a[v1][i]<a[v2][i])
          near[i]=v1;
        else
          near[i]=v2;
}
near[v1]=near[v2]=0;
for(i=2;i<=n-1;i++)
{
        min=999;
        for(q=1;q<=n;q++)
        {
                if(a[q][near[q]]<min && near[q]!=0)
                {
                        min=a[q][near[q]];
                        j=q;
                }
        }
        //update cost of MST
        minmst+=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(a[k][near[k]]>a[k][j] && near[k]!=0)
                  near[k]=j;
        }

}
printf("\n the minimum spanning tree is : \n");
```

```
        for(i=1;i<=n-1;i++)
        {
                printf("%d--->%d",t[i][1],t[i][2]);
                printf("\n");
        }
        printf("\n minimum cost of mst=%d",minmst);
}
```

31: Write a program to find minimum spanning tree using kruskals algorithm

```
#include<stdio.h>
void kruskals();
void minheapify(int i);
void buildminheap();
int edge[10],g[10][10],n,e,p[10];
//main function
int main()
{
        int i,j,ch,v1,v2,w;
        printf("\n enter number 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++)
                  g[i][j]=999;
        }
        switch(ch)
        {
                case 1:printf("\n enter number of edges in the directed graph\n");
                       scanf("%d",&e);
                       printf("\n enter the pair of vertices V1-->V2 and weight");
                       for(i=1;i<=e;i++)
                       {
                            scanf("%d%d%d",&v1,&v2,&w);
                            g[v1][v2]=w;
                            edge[i]=w;
                            }
                            break;
                case 2:printf("\n enter the number of edges in undirected graph\t");
                       scanf("%d",&e);
                       for(i=1;i<=e;i++)
```

```
                {
                    printf("\n Enter the pair of vertices having edge b/w them ");
                        scanf("%d%d%d",&v1,&v2,&w);
                    g[v1][v2]=w;
                    g[v2][v1]=w;
                    edge[i]=w;
                    }
                    break;
            default:printf("\n enter correct choice");
        }
        printf("\n MATRIX REPRESENTATION\n");
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                printf("%d\t",g[i][j]);
            }
            printf("\n");
        }
        kruskals();
}
//find function to find the set that contain 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 dounion(int j,int k)
{
        p[j]=k;
}
//delete an min cost edge from minheap
int deletek()
{
        int x;
        x=edge[1];
        edge[1]=edge[e];
        e--;
        minheapify(1);
        return x;
}
```

```
void kruskals()
{
        int i=1,v1,v2,j,minmst=0,t[10][2],q,w,x,s,r;
        buildminheap();
        printf("\n array after min heap: \n");
        printf("\n");
   for(j=1;j<=n;j++)
     p[j]=-1;
   while(i<=n-1&& e>0)
   {
        x=deletek();  //delete an 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)
           {
                t[i][1]=v1;
                t[i][2]=v2;
                i++;
                dounion(v1,v2);
                minmst+=x;
                }
   }
   if(i<=n-1)
     printf("\n No minimum spanning tree ");
   else
   {
        printf("\n The minimum spanning 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("\n minimum cost =%d ",minmst);
```

```
        }

}
void buildminheap()
{
        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);
          }
}
```

32: Write a program to find single source shortest path using dijkstra's algorithm

```
#include<stdio.h>
#include<conio.h>
int main(){
        int G[20][20],w,cost[20][20],v,dist[20],s[20] ,min,i,k, j,e, n, u;
        printf("\nEnter the no. of vertices:: ");
        scanf("%d", &n);
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        G[i][j]=0;
        printf("\n Enter the total number of edges : ");
        scanf("%d",&e);
        for(i=1;i<=e;i++)
                for(j=1;j<=e;j++)
                {
                        if(i==j)  cost[i][j]=0;
```

```
                else  cost[i][j]=999;
            }

    for(k=1;k<=e;k++)
    {
            printf("\n enter two vertices & cost for edge-%d : ", k);
            scanf("%d", &i); scanf("%d" ,&j); scanf("%d",&cost[i][j]);
            G[i][j]=1;
            G[j][i]=1;

}
printf("\nAdjacency matrix::\n");
for(i=1;i<=n;i++)   //printing adjacency matrix
{
    printf("\n");
    for(j=1;j<=n;j++)
    {
            printf("\t%d",G[i][j]);
            }
}
    printf("\n\n cost matrix is : ");
     for(i=1;i<=n;i++)   //printing cost matrix
{
    printf("\n");
    for(j=1;j<=n;j++)
    {
            printf("\t%d",cost[i][j]);
            }
    }
    printf("\n\nEnter the starting node:: ");
    scanf("%d", &v);
for(i=1;i<=n;i++)
{
    dist[i]=cost[v][i];
    s[i]=0;
    }
    s[v]=1;
    dist[v]=0;
    u=v;
    for(k=2;k<=n;k++)
    {
            min=999;
            for(i=1;i<=n;i++)
            {
```

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

33: Write a program to find single source shortest path using dijkstra's algorithm and print the path along with it

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 999
#define MAX 10

void find_min(int G[MAX][MAX],int cost[MAX][MAX], int n, int startnode);

int main(){
        int G[MAX][MAX],cost[MAX][MAX], i,k, j,e, n, u;
        printf("\nEnter the no. of vertices:: ");
        scanf("%d", &n);
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        G[i][j]=0;
        printf("\n Enter the total number of edges : ");
        scanf("%d",&e);
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
```

```
                {
                        if(i==j)
                           cost[i][j]=0;
                        else
                           cost[i][j]=INFINITY;
                }

        for(k=1;k<=e;k++)
        {
                printf("\n enter two vertices & cost for edge-%d : ", k);
                scanf("%d", &i); scanf("%d" ,&j); scanf("%d",&cost[i][j]);
                G[i][j]=1;

    }
    printf("\nAdjacency matrix::\n");
    for(i=1;i<=n;i++)   //printing adjacency matrix
    {
        printf("\n");
        for(j=1;j<=n;j++)
        {
                printf("\t%d",G[i][j]);
                }
        }
        printf("\n\n cost matrix is : ");
         for(i=1;i<=n;i++)   //printing cost matrix
    {
        printf("\n");
        for(j=1;j<=n;j++)
        {
                printf("\t%d",cost[i][j]);
                }
        }
        printf("\n\nEnter the starting node:: ");
        scanf("%d", &u);
        printf("\n-------------------------------------------------------------------------------");
        find_min(G,cost,n,u);
        return 0;
}
 //function to find minimum distance of all vertices from starting node
void find_min(int G[MAX][MAX],int cost[MAX][MAX], int n, int startnode)
{
        int distance[MAX], pred[MAX];
        int visited[MAX], count, mindistance, nextnode, i,j;
        for(i=1;i<=n;i++)
```

```
{
        distance[i]=cost[startnode][i];
        pred[i]=startnode;
        visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=2;
while(count<=n-1){
        mindistance=INFINITY;
        for(i=1;i<=n;i++)
        {
        if(distance[i]<mindistance && visited[i]!=1)
                {
                        mindistance=distance[i];
                        nextnode=i;
                }
   }
        visited[nextnode]=1;
        for(i=1;i<=n;i++)
        {
                if(visited[i]!=1)
                {
                        if(mindistance+cost[nextnode][i] < distance[i])
                        {
                                distance[i]=mindistance+cost[nextnode][i];
                                pred[i]=nextnode;
                        }
                }
        }
        count++;
}
for(i=1;i<=n;i++)
{

        if(i!=startnode)
        {
                printf("\n Minimum Distance of vertex %d = %d", i, distance[i]);
                printf("\n PATH : %d -> ",startnode);
                j=i;
                j=pred[j];
              while(j!=startnode)
                {
                        printf("%d -> ", j);
```

```
                j=pred[j];
            }
            printf("%d",i);
        }
    }}
```

34: Write a program to find single source shortest path using bellmanford's algorithm

```
#include<stdio.h>
void bellmanford(int cost[10][10], int n ,int v);
int main()
{
    int i,j,n,e,ch,v1,v2,a[10][10],w;
    printf("\n enter the number of vertices in the graph: ");
    scanf("%d", &n);
    printf("graph is 1.directed 2.undirected\n enter your choice \t");
    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("\n enter number of edges in the directed graph\n");
           scanf("%d",&e);
           for(i=1;i<=e;i++)
           {
               printf("\n enter the pair of vertices V1-->V2 and weight");
                        scanf("%d%d%d",&v1,&v2,&w);
               a[v1][v2]=w;
               }
               break;
        case 2:printf("\n enter the number of edges in undirected graph\t");
```

```
                scanf("%d",&e);
                for(i=1;i<=e;i++)
                {
                    printf("\n Enter the pair of vertices having edge b/w them ");
                        scanf("%d%d%d",&v1,&v2,&w);
                    a[v1][v2]=w;
                    a[v2][v1]=w;
                    }
                    break;
            default:printf("\n enter correct choice");
        }
        printf("\n COST MATRIX REPRESENTATION\n");
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                printf("%d\t",a[i][j]);
            }
            printf("\n");
        }
        printf("\n enter the source vertex : ");
        scanf("%d",&v1);
        bellmanford(a,n,v1);
}
void bellmanford(int cost[10][10], int n ,int v)
{
        int a[10][10],i,j,k,min;
        for(i=1;i<=n;i++)
             a[1][i]=cost[v][i];
        for(i=2;i<=n-1;i++)
        {
            for(j=1;j<=n;j++)
            {
                min=99;
                for(k=1;k<=n;k++)
                {
                    if(k!=v && k!=j)
                    {
                        if(min>a[i-1][k]+cost[k][j])
                          min=a[i-1][k]+cost[k][j];
                    }
                }
                if(min>a[i-1][j])
                  a[i][j]=a[i-1][j];
```

```
                else
                  a[i][j]=min;
            }
        }
        printf("\n THE SOLUTION\n");
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                printf("%d\t",a[i][j]);
            }
            printf("\n");
        }

}
```

35: Write a program to find all pair shortest path using Floydwarshall's algorithm

```
#include<stdio.h>
void floyd(int cost[10][10], int n);
int main()
{
        int i,j,n,e,ch,v1,v2,a[10][10],w;
        printf("\n enter the number of vertices in the graph: ");
        scanf("%d", &n);
        printf("graph is 1.directed 2.undirected\n enter your choice \t");
        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("\n enter number of edges in the directed graph\n");
                scanf("%d",&e);
                printf("\n enter the pair of vertices V1-->V2 and weight");
                for(i=1;i<=e;i++)
```

```
                {
                    scanf("%d%d%d",&v1,&v2,&w);
                    a[v1][v2]=w;
                    }
                    break;
            case 2:printf("\n enter the number of edges in undirected graph\t");
                scanf("%d",&e);
                for(i=1;i<=e;i++)
                {
                    printf("\n Enter the pair of vertices having edge b/w them ");
                        scanf("%d%d%d",&v1,&v2,&w);
                    a[v1][v2]=w;
                    a[v2][v1]=w;
                    }
                    break;
            default:printf("\n enter correct choice");
        }
        printf("\n COST MATRIX REPRESENTATION\n");
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                printf("%d\t",a[i][j]);
            }
            printf("\n");
        }
        floyd(a,n);
}
void floyd(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(i=1;i<=n;i++)
        {
            for(j=1;j<=n;j++)
            {
                for(k=1;k<=n;k++)
                {
                    if(a[j][i]+a[i][k]<a[j][k])
                      a[j][k]=a[j][i]+a[i][k];
```

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

}
```

36: Write a program to implement threaded binary tree

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
//structure for threaded binary tree
struct node{
        struct node*left;
        struct node*right;
        int info;
        int rthread;
        int lthread;
};
struct node* tree=NULL;
void maketree(int n);
void createtree(int n);
void insert(struct node *,int x);
struct node* search(struct node*tree, int x);
struct node* insuccessor(struct node*x);
struct node* inpredecessor(struct node *x);
void inorder(struct node*p);
void preorder(struct node*p);
void setleft(struct node *p, int x);
void setright(struct node *p, int x);

int main()
{
        int a,n,x,c,b,ch,e;
```

```
struct node* p;
do{
        printf("\n 1.Create \t2.Insert \t3.Search");
        printf("\n 4.Inorder traversal \t5.Preorder traversal");
        printf("\n 6.Exit\n ENTER YOUR CHOICE-> ");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: printf("\n Enter how many nodes : ");
                        scanf("%d",&n);
                        createtree(n);
                        //inorder(tree);
                        break;
                case 2: printf("\n Enter new node value to insert ");
                        scanf("%d",&x);
                        insert(tree,x);
                       // inorder(tree);
                        break;
                case 3: printf("\n Enter element to search");
                        scanf("%d",&x);
                        p=search(tree,x);
                        if(p==NULL)
                          printf("\n node is not present");
                        else
                          printf("\n node is present ");
                        break;
                case 4: inorder(tree);
                        break;
                case 5: preorder(tree);
                        break;
                case 6: printf("\n Enter element whose predecessor you want ");
                        scanf("%d",&x);
                        p=search(tree,x);
                        if(p==NULL)
                         printf("\n Existing node not present ");
                        else
                        {
                          p=inpredecessor(p);
                          if(p==NULL)
                            printf("\n no predecessor ");
                          else
                            printf("\n predecessor of %d is %d", x, p->info);
                                }
                                break;
```

```
                        case 7: printf("\n Enter element whose successor you want ");
                                scanf("%d",&x);
                                p=search(tree,x);
                                if(p==NULL)
                                  printf("\n Existing node not present ");
                                else
                                {
                                  p=insuccessor(p);
                                  if(p==NULL)
                                    printf("\n no successor ");
                                  else
                                    printf("\n successor of %d is %d", x, p->info);
                                        }
                                        break;
                        case 8: printf("\n THANK YOU!!");
                                break;
                }
        }while(ch>=1 && ch<=7);
}

struct node *search(struct node *tree, int x)
{
        struct node *p, *q;
        p=tree;
        if(tree==NULL || tree->info==x)
           return p;
        else if(x<tree->info)
        {
                printf("\n seraching left");
                if(tree->lthread==1)
                  return NULL;
                else
                   return search(tree->left,x);
        }
        else if (x > tree->info)
        {
                printf("\n seraching right");
                if(tree->rthread==1)
                  return NULL;
                else
                   return search(tree->right,x);
        }
}
```

```
//create a singlr node and tree point to that node
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;
        tree=p;
        printf("\n inserted");
}

//create a TBT using n nodes
void createtree(int n)
{
        int i,x;
        printf("\n Enter the %d values\n",n);
        for(i=0; i<n; i++)
        {
                scanf("%d",&x);
                if(i==0)
                 maketree(x);
                else{
                        insert(tree,x);
                    printf("\n inserted1");

                }
   }
}

//insert a node into tbt
void insert(struct node *tree, int x)
{
        if(tree==NULL)
          maketree(x);
        else if(x < tree->info)
        {
                if(tree->lthread==1)
                {
                        setleft(tree,x);
                        printf("\n inserted left");
                }
```

```
                else
                {
                        insert(tree->left,x);
                }
        }
        else if(x >= tree->info)
        {
                if(tree->rthread==1)

                {
                        setright(tree,x);
                        printf("\n right inserted");
                }
                else
                {
                   insert(tree->right,x);
                }
        }
}

//set a new node to left of node P
void setleft(struct node *p1, int x1)
{
        struct node*q1, *r1;
        if(p1==NULL)
          printf("\n cant insert");
        else if(p1->lthread==0)
            printf("\n void insertion");
        else
        {
                q1=(struct node *)malloc(sizeof(struct node));
           q1->info=x1;
           r1=p1->left;
           p1->left=q1;
           q1->left=r1;
           q1->right=p1;
           p1->lthread=0;
           q1->lthread=1;
           p1->rthread=1;
        }

}
```

```
//find successor of a given node x
struct node* insuccessor(struct node *x)
{
        struct node *s;
        if(x->rthread==0)
            return x->right;
        else
        {
                s=x->right;
                while(s->lthread==1)
                  {
                        s=s->left;
                  }
                return s;
        }
}
//find predecessor of a given node x
struct node* inpredecessor(struct node *x)
{
   struct node*pred;
        pred=x->left;
        if(x->lthread==0)
        {
                while(pred->lthread==0)
                   pred=pred->left;
        }
        return pred;
}
//preorder traversal of TBT
void preorder(struct node *tree)
{
        struct node *q,*p;
        p=tree;
        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(q!=NULL);
}

void setright(struct node *p, int x)
{
        struct node*q, *r;
        if(p==NULL)
          printf("\n cant insert");
        else if(p->rthread==0)
             printf("\n void insertion");
        else
        {
                q=(struct node *)malloc(sizeof(struct node));
           q->info=x;
           r=p->right;
           p->right=q;
           q->right=r;
           p->rthread=0;
           q->rthread=1;
           q->left=p;
           p->lthread=1;
        }
}

//inorder traversal of TBT
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(q!=NULL);
}
```

37: Write a program to implement N-queens problem

```
//n-queens program
#include<stdio.h>
#include<stdlib.h>
int x[10],n;
void nqueen(int k);
int place(int k,int i);
int main()
{
        int i;
        printf("\n enter the number of queens ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
            x[i]=0;
        }
        nqueen(1);
}
//n-queens function
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("\n");
                        printf("\n The 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;
}
```

38: Write a program to implement sum of subsets problem

```
#include<stdio.h>
//program to implement sum of subsets problem
int n,w1[10],w[10],m,x1[10],x[10];
void sumofsub(int s,int k,int r);
int main()
{
    int i,j,temp,r=0;
    printf("\n enter the number of elements in the set ");
    scanf("%d",&n);
    printf("\n enter the elements of 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;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("\n enter the value of M: ");
        scanf("%d",&m);
        for(i=1;i<=n;i++)
        {
                x[i]=0;
        }
        sumofsub(0,1,r);
        return 0;

}

void sumofsub(int s, int k,int r)
{
        int i,j,l;
        x[k]=1;
        if(s+w1[k]==m)
        {
                printf("\n Solution 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]+w1[k+1]<=m)
        {
           sumofsub(s+w1[k],k+1,r-w1[k]);
        }

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

39: Write a program to implement graph coloring problem using backtracking

```
#include<stdio.h>
int G[50][50],x[50];
void next_color(int k)
{
  int i,j;
  x[k]=1;  //coloring vertex with color1
  for(i=0;i<k;i++)
{
    if(G[i][k]!=0 && x[k]==x[i])
      x[k]=x[i]+1;  //assign higher color than x[i]
  }
}

int main(){
 int n,e,i,j,k,l;
 printf("Enter no. of vertices : ");
 scanf("%d",&n);  //total vertices
 printf("Enter no. of edges : ");
 scanf("%d",&e);

 for(i=0;i<n;i++)
   for(j=0;j<n;j++)
     G[i][j]=0;

 printf("Enter indexes where value is 1-->\n");
 for(i=0;i<e;i++){
```

```
  scanf("%d %d",&k,&l);
  G[k][l]=1;
  G[l][k]=1;
 }

 for(i=0;i<n;i++)
  next_color(i);

 printf("Colors of vertices -->\n");
 for(i=0;i<n;i++)
  printf("Vertex[%d] : %d\n",i+1,x[i]);

 return 0;
}
```

40: Write a program to implement Hamiltonian cycle using backtracking

```
#include<stdio.h>
#define V 5
void printSolution(int path[]);
bool isSafe(int v, bool graph[V][V], int path[], int pos)
{

        if (graph [ path[pos-1] ][ v ] == 0)
                return false;

        for (int i = 0; i < pos; i++)
                if (path[i] == v)
                        return false;

        return true;
}
bool hamilCycle(bool graph[V][V], int path[], int pos)
{

        if (pos == V)
        {

                if ( graph[ path[pos-1] ][ path[0] ] == 1 )
                return true;
                else
                return false;
        }
```

```
        for (int v = 1; v < V; v++)
        {
                if (isSafe(v, graph, path, pos))
                {
                        path[pos] = v;

                        if (hamilCycle (graph, path, pos+1) == true)
                                return true;

                        path[pos] = -1;
                }
        }

        return false;
}
bool hamCycle(bool graph[V][V])
{
        int *path = new int[V];
        for (int i = 0; i < V; i++)
                path[i] = -1;
        path[0] = 0;
        if ( hamilCycle(graph, path, 1) == false )
        {
                printf("\nSolution does not exist");
                return false;
        }

        printSolution(path);
        return true;
}
void printSolution(int path[])
{
        printf ("Solution Exists:"
                        " Following is one Hamiltonian Cycle \n");
        for (int i = 0; i < V; i++)
                printf(" %d ", path[i]);

        printf(" %d ", path[0]);
        printf("\n");
}
int main()
{
bool graph1[V][V] = {{0, 1, 0, 1, 0},
```

```
                                        {1, 0, 1, 1, 1},
                                        {0, 1, 0, 0, 1},
                                        {1, 1, 0, 0, 1},
                                        {0, 1, 1, 1, 0},
                                        };
        hamCycle(graph1);

        bool graph2[V][V] = {{0, 1, 0, 1, 0},
                                        {1, 0, 1, 1, 1},
                                        {0, 1, 0, 0, 1},
                                        {1, 1, 0, 0, 0},
                                        {0, 1, 1, 0, 0},
                                        };

        hamCycle(graph2);

        return 0;
}
```