Design and Analysis of Algorithms

LAB MANUAL

Subject code: CS402
IV Semester B.E.
(2015)

Computer Science & Engineering Department
NMAMIT, Nitte

PART A

1. N- Queen's Problems

The problem is to place n queens on a n-by-n chessboard so that no two queens attack each other by being in the same row or in the same column or on the same diagonal.

PROGRAM

```
/* N - Queens problem using Backtracking */

#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>

int place(int k, int a[])
{
 int i;
        for(i=1;i<k;i++)
        {
                if( (a[k]==a[i]) || ( abs(i-k) == abs(a[i]-a[k]) ) )
                    return 0;
        }
        return 1;
}

void nqueen(int n)
{
 int k=1;                   // queen 1 in row 1

 int i,count=0;     // count indicates the no of solutions
 int a[10];               // a[k] holds column no of k th queen
a[k]=0;            //queen 1,in row 1 ,column 0

 while(k!=0)        //backtrack till k=1
 {
  a[k]=a[k]+1;     //queen k,in next column

  while( (a[k]<=n) && (place(k,a)==0) )
  {
        a[k]=a[k]+1;
  }

  if(a[k]<=n)
  {
        if(k==n)         //if we reached last row then print solution
        {
                count++;
                printf("Solution No %d:\n", count);
                for(i=1;i<=n;i++)
                {
```

```
                        printf("Queen No: %d is placed in  Row No: %d  and Column No %d\n",
i, i,a[i]);
                }
        }
        else
        {
                k++;
                a[k]=0;
        }
  }
  else
  {
        k--;    //backtrack to previous row
  }
}

}



int main()
{
 int n,noofsoln;

 printf("Enter the number of queens\n");
 scanf("%d",&n);
 if(n<=3 && n>1)
   printf("no solutions\n");
 else
nqueen(n);
 getch();
}
```

2. Dijkstra's Algorithm

ALGORITHM Dijkstra(G,s)
//Dijkstra's algorithm for single source shortest paths
//Input: A weighted connected graph G=<V,E> and its vertex s
//Output: The length d_v of a shortest path from s to v
//and its penultimate vertex p_v for every vertex v in V

Initialize(Q) //initialize vertex priority queue to empty
for every vertex v in V **do**
$d_v \leftarrow \infty$; $p_v \leftarrow$ **null**
Insert(Q,v,d_v) //Initialize vertex priority in the priority queue
$d_s \leftarrow 0$; Decrease(Q,s,d_s) //Update priority of s with d_s
$V_T \leftarrow \Phi$

for i ← 0 **to** |V| -1 **do**
    u* ← DeleteMin(Q)   //delete the minimum priority element
    V_T ← V_T U {u*}
     for every vertex u in V-V_T that is adjacent to u* **do**
        if d_u* + w(u*,u) < d_u
            d_u ← d_u* + w(u*,u); p_u ← u*
             Decrease(Q,u,d_u)

PROGRAM

```
/* Dijkstra's algorithm to find Single Source Shortest Path in a graph*/

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

 int A[10][10]; //A-given graph;
 int n,e;
 int dist[10],visited[10];

int min(int i,int k)
   {
    if (i<k )return i;
      else return k ;
   }
void dijkstra(int s)
{
  int i, w, u, k, min_cost;
  for(i=1;i<=n;i++)
  {
        visited[i]=0;
        dist[i]=A[s][i];
  }
  for(k=1;k<=n;k++)
  {
        min_cost=999;
        for(i=1;i<=n;i++)
        {
            if( (dist[i]<min_cost) && (visited[i]==0) )
            {
                u=i;
                min_cost=dist[i];
            }
        }
        visited[u]=1;
        for(w=1;w<=n;w++)
        {
            if(visited[w]==0)
                dist[w]=min(dist[w],dist[u]+A[u][w]);
        }
```

```
  }
}




int main()
{
 int v1,v2, s;
 int i,j,cost;
 //clrscr();

 printf("Enter no of vertices\n");
 scanf("%d",&n);

 printf("Enter no of edges\n");
 scanf("%d",&e);

 for(i=1;i<=n;i++)                        //initialization of adjacency matrix
         for(j=1;j<=n;j++)
         {
          if(i==j)
                  A[i][j]=0;
          else
                  A[i][j]=999;
         }

 printf("Enter the Edges one by one\n");
 for(i=1;i<=e;i++)
 {
         printf("Edge %d: ",i);
         scanf("%d%d",&v1,&v2);
         printf("Weight: ");
         scanf("%d",&cost);
         A[v1][v2]=A[v2][v1]=cost;
 }

 printf("Enter the Starting Vertex\n");
 scanf("%d",&s);

 printf("The given graph is:\n");
 for(i=1;i<=n;i++)
 {
         for(j=1;j<=n;j++)
                  printf("%d\t",A[i][j]);
         printf("\n");
 }

 dijkstra(s);
```

```
 printf("The Shortest Distance from %d to : \n",s);

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

 getch();
}
```

3. Kruskal's Algorithm

ALGORITHM Kruskal(G)
//Kruskal's algorithm for construction a minimum spanning tree
//Input: A weighted connected graph G=<V,E>
//Output: E_T, the set of edges composing a minimum spanning tree of G

Sort E in nondecreasing order of the edge weights $w(e_{i1}) \leq \ldots \leq w(e_i |E|)$
$E_T \leftarrow \Phi$; ecounter $\leftarrow$ 0 //Initialize the set of tree edges and its sizes
k $\leftarrow$ 0 // Initialize the number of processed edges
while ecounter < |V| -1
    k $\leftarrow$ k+1
    if E_T U $\{e_{ik}\}$ is acyclic
        $E_T \leftarrow E_T$ U $\{e_{ik}\}$; ecounter $\leftarrow$ ecounter +1
return E_T

PROGRAM

```
/* Kruskal's algorithm to find minimum spanning tree */
#include<stdio.h>
#include<conio.h>

 int A[10][10],B[10][10]; //A-given graph; B - Minimum Spanning Tree
 int n,e;
 int root[10];

void update(int v1, int v2)
{
  int i, t;
  t=root[v2];
  for(i=1;i<=n;i++)
  {
        if(root[i]==t)
                root[i]=root[v1];
  }
}


void FindMin(int *v1, int *v2)
```

```
{
 int edge=0, i,j;
 for(i=1;i<=n;i++)
        for(j=i+1;j<=n;j++)
        {
                if( (edge>A[i][j] || edge==0) && A[i][j]>0 )
                {
                        edge=A[i][j];
                        *v1=i;
                        *v2=j;
                }

        }
}

int Kruskals()
{
 int i,j,ecounter=0;
 int v1, v2;
 int cost;
 int total_cost=0;

 for(i=1;i<=n;i++)              //initialization of adjacency matrix
        for(j=1;j<=n;j++)
        {
          if(i==j)
                B[i][j]=0;
          else
                B[i][j]=999;
        }

 while(ecounter<(n-1))
 {
        FindMin(&v1, &v2);

        cost=A[v1][v2];

        A[v1][v2]=A[v2][v1]=0;

        if(root[v1]!=root[v2])
        {
                B[v1][v2]=B[v2][v1]=cost;
                update(v1, v2);
                total_cost=total_cost+cost;
                ecounter++;
        }
 }
return total_cost;
}
```

```
int main()
{
 int v1,v2;
 int i,j,cost;
 //clrscr();

 printf("Enter no of vertices\n");
 scanf("%d",&n);

 printf("Enter no of edges\n");
 scanf("%d",&e);

 for(i=1;i<=n;i++)                    //initialization of adjacency matrix
         for(j=1;j<=n;j++)
         {
          if(i==j)
                  A[i][j]=0;
          else
                  A[i][j]=999;
         }

 printf("Enter the Edges one by one\n");
 for(i=1;i<=e;i++)
 {
         printf("Edge %d: ",i);
         scanf("%d%d",&v1,&v2);
         printf("Weight: ");
         scanf("%d",&cost);
         A[v1][v2]=A[v2][v1]=cost;
 }

 for(i=1;i<=n;i++)
 {
         root[i]=i;
 }

  printf("The given graph is:\n");
 for(i=1;i<=n;i++)
 {
         for(j=1;j<=n;j++)
                 printf("%d\t",A[i][j]);
         printf("\n");
 }


 cost=Kruskals();


 printf("The minimum spanning tree for the given graph is:\n");
 for(i=1;i<=n;i++)
```

```
{
        for(j=1;j<=n;j++)
                printf("%d\t",B[i][j]);
        printf("\n");
}
printf("\nThe total cost is: %d",cost);
getch();
}
```

4. Prim's Algorithm

ALGORITHM Prim(G)
//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph G=<V,E>
//Output: E_T, the set of edges composing a minimum spanning tree of G

$V_T \leftarrow \{v_0\}$ // The set of tree vertices can be initialized with any vertex
$E_T \leftarrow \Phi$
for i ← 1 **to** |V|-1 **do**
    find a minimum-weight edge e*=(v*,u*) among all the edges (u,v)
    such that v is in V_T and u is in V-V_T
    $V_T \leftarrow V_T \cup \{u^*\}$
    $E_T \leftarrow E_T \cup \{e^*\}$
return E_T

PROGRAM

```
/*Prim's algorithm to find minimum spanning tree*/
#include<stdio.h>
#include<conio.h>

 int A[10][10],B[10][10]; //A-given graph; B - Minimum Spanning Tree
 int n,e;
 int visited[10];

void FindMin(int *v1, int *v2)
{
 int edge=0, i,j;
 for(i=1;i<=n;i++)
        for(j=i+1;j<=n;j++)
        {
                if( (visited[i]==1 && visited[j]!=1) || (visited[i]!=1 && visited[j]==1) )
                {
                        if( (edge>A[i][j] || edge==0) && A[i][j]>0 )
                        {
                                edge=A[i][j];
                                *v1=i;
                                *v2=j;
                        }
                }
        }
```

```
}

int Prims()
{
 int i,j;
 int v1, v2;
 int cost;
 int total_cost=0;

 for(i=1;i<=n;i++)                    //initialization of adjacency matrix
        for(j=1;j<=n;j++)
        {
         if(i==j)
                B[i][j]=0;
         else
                B[i][j]=999;
        }

 for(i=1;i<n;i++)
 {
        FindMin(&v1, &v2);
        cost=A[v1][v2];
        B[v1][v2]=B[v2][v1]=cost;
        visited[v1]=visited[v2]=1;
        total_cost=total_cost+cost;
 }
 return total_cost;
}


int main()
{
 int v1,v2;
 int i,j,cost;
 //clrscr();

 printf("Enter no of vertices\n");
 scanf("%d",&n);

 printf("Enter no of edges\n");
 scanf("%d",&e);

 for(i=1;i<=n;i++)                    //initialization of adjacency matrix
        for(j=1;j<=n;j++)
        {
         if(i==j)
                A[i][j]=0;
         else
                A[i][j]=999;
        }
```

```
printf("Enter the Edges one by one\n");
for(i=1;i<=e;i++)
{
        printf("Edge %d: ",i);
        scanf("%d%d",&v1,&v2);
        printf("Weight: ");
        scanf("%d",&cost);
        A[v1][v2]=A[v2][v1]=cost;
}

for(i=1;i<=n;i++)
{
        visited[i]=0;
}

visited[1]=1;

cost=Prims();

printf("The given graph is:\n");
for(i=1;i<=n;i++)
{
        for(j=1;j<=n;j++)
                printf("%d\t",A[i][j]);
        printf("\n");
}

printf("The The minimum spanning tree for the given graph is:\n");
for(i=1;i<=n;i++)
{
        for(j=1;j<=n;j++)
                printf("%d\t",B[i][j]);
        printf("\n");
}
printf("\nThe total cost is: %d",cost);
getch();
}
```

5.Knapsack problem with memory funcions

ALGORITHM MFKnapsack(i,j)
//Implements the memory function method for the knapsack problem
//Input: A nonnegative integer i indicating the number of first
//items being considered and a nonnegative integer j indicating the knapsack's capacity
//Output: The value of an optimal feasible subset of the first i items
//Note: Uses as global variables input arrays Weights[1..n],Values[1..n]
//and table V[0..n,0..W] whose entries are initialized with -1's except for
//row 0 and column 0 initilized wit 0's

if V[i,j] < 0

```
    if j<Weights[i]
        value ← MFKanpsack(i-1,j)
    else
        value ← max(MFKnapsack(i-1,j),Values[i]+MFKnapsack(i-1,j-Weights[i]))
    V[i,j] ← value
 return V[i,j]
```

PROGRAM

```
/*Knapsack problem using memory functions - dynamic programming*/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int Values[15], Weights[15];
int W,n;
int V[15][15];

int max(int i,int k)
   {
    if (i>k )return i;
      else return k ;
   }

int MFKnapsack(int i,int j)
{
 int value;
 if(V[i][j]<0)
 {
        if(j<Weights[i])
                value=MFKnapsack(i-1,j);
        else
                value=max(MFKnapsack(i-1,j), Values[i]+MFKnapsack(i-1,j-Weights[i]));

        V[i][j]=value;
 }
 return V[i][j];
}

int main()
{
 int i,j,val;
 //clrscr();

 printf("Enter no of items\n");
 scanf("%d",&n);

 printf("Enter the capacity of knapsack\n");
```

```
scanf("%d",&W);

printf("Enter the weight and value of each item\n");
for(i=1;i<=n;i++)
{
        printf("\nItem No: %d ->",i);
        printf("Weight: ");
        scanf("%d",&Weights[i]);
        printf("Value: ");
        scanf("%d",&Values[i]);
}
for(i=0;i<=n;i++)
  for(j=0;j<=W;j++)
  {
        if(i==0 || j==0)
        {
          V[i][j]=0;
        }
        else
        {
          V[i][j]=-1;
        }
  }

val=MFKnapsack(n,W);

printf("The optimal value is %d\n",val);
printf("The items added are:\n");
for(i=n,j=W;i>0;i--)
{
  if(V[i][j]!=V[i-1][j])
  {
        printf("%d\t",i);
        j=j-Weights[i];
  }
}
getch();
return 0;
}
```

6. Knapsack problem using dynamic programming

```
/*Knapsack problem using dynamic programming*/
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>

int Values[15], Weights[15];
int W,n;
```

```
int V[15][15];

int max(int i,int k)
    {
      if (i>k )return i;
        else return k ;
    }

int Knapsack()
{
int i,j;
   for(i=1;i<=n;i++)
   {
    for (j=1;j<=W;j++)
   {
      if(j-Weights[i]<0)
      {
          V[i][j]=V[i-1][j];
      }
      else
      {
          V[i][j]=max(V[i-1][j],Values[i]+V[i-1][j-Weights[i]]);
      }

   }
   }
    return V[n][W];
}

int main()
{
 int i,j,val;
// clrscr();

 printf("Enter no of items\n");
 scanf("%d",&n);

 printf("Enter the capacity of knapsack\n");
 scanf("%d",&W);

 printf("Enter the weight and value of each item\n");
 for(i=1;i<=n;i++)
 {
          printf("\nItem No: %d ->",i);
          printf("Weight: ");
          scanf("%d",&Weights[i]);
          printf("Value: ");
          scanf("%d",&Values[i]);
 }
 for(i=0;i<=n;i++)
    V[i][0]=0;//initialize first row
```

```
 for(j=0;j<=W;j++)
    V[0][j]=0;//initialize first column

val=Knapsack();
printf("The Knapsack table %d\n");
for(i=0;i<=n;i++)
{
   for(j=0;j<=W;j++)
     {
    printf("%d\t",V[i][j]);
     }
   printf("\n");
}

 printf("The optimal value is \n",val);
 printf("The items added are:\n");
 for(i=n,j=W;i>0;i--)
 {
    if(V[i][j]!=V[i-1][j])
    {
          printf("%d\t",i);
          j=j-Weights[i];
    }
 }
 getch();
 return 0;
}
```

7. Write a C/C++ program to implement Descending Priority Queue using Heap.

ALGORITHM HeapBottomUp(H[1..n])
//Constructs a heap from the elements of a given array
//by the bottom-up algorithm
//Input: An array H[1..n] of orderable items
//Output: A heap H[1..n]

for i ← ⌊n/2⌋ **downto** 1 **do**
    k ← i; v ← H[k]
    heap ← **false**
    while not heap and 2*k≤n **do**
        j ← 2*k
        if j<n //there are two children
            if H[j] < H[j+1] j ← j+1
        if v≥H[j]
            heap ← **true**
        else H[k] ← H[j]; k ← j
    H[k] ← v

```
#include<stdio.h>
#include<conio.h>
```

```
int n=0,h[20];
void heapifybottomup()
  {
    int i,k,v,heapify,j;
    for(i=(n/2);i>=1;i--)
       {
       k=i;
       v=h[k];
       heapify=0;
            while(heapify==0 && 2*k<=n)
               {
           j=2*k;
       if(j<n) //if there are 2 children
           {
           if(h[j]<h[j+1]) //checking both children
                          j=j+1;
           }
           if(v>=h[j])
           {
                       heapify=1;
           }
           else
                       {
                        h[k]=h[j];
              k=j;
                       }
                }
            h[k]=v;
           }

  }
void insert_ele(int ele)
{

   n++;
   h[n]=ele;
}
void remove_ele()
{
   int temp;

   heapifybottomup();

           temp=h[1];
           h[1]=h[n];
           h[n]=temp;
           printf("Deleted element is %d",h[n]);
           n--;

}
void display()
```

```
{
int i;
if (n == 0)
       printf("Queue is empty \n");
   else
   {
   printf("\n The queue is : \n");
   for(i=1;i<=n;i++)
   {
    printf("%d\t",h[i]);
   }
   }

}

int main()
  {
   int i,ele,choice;

   while (1)
   {
      printf("\n1.Insert element to queue \n");
      printf("\n2.Delete element from queue \n");
      printf("\n3.Display all elements of queue \n");
      printf("\n4.Quit \n");
      printf("Enter your choice : ");
      scanf("%d", &choice);
      switch (choice)
      {
         case 1:
            printf("\n Enter the element : ");
            scanf("%d",&ele);

         insert_ele(ele);
         break;
         case 2:
         remove_ele();
         break;
         case 3:
         display();
         break;
         case 4:
         exit(1);
       }
   }
    getch();
   return 0;
  }
```

8. Heapsort

Heapsort Technique

1. (Heap construction): Construct a heap for a given array.
2. (Maximum deletions): Apply the root –deletion operation n – 1 times to the remaining heap.

PROGRAM

```
#include<stdio.h>
#include<conio.h>

void heapifybottomup(int h[],int n)
 {
  int i,k,v,heapify,j;
  for(i=(n/2);i>=1;i--)
    {
     k=i;
     v=h[k];
     heapify=0;
          while(heapify==0 && 2*k<=n)
             {
        j=2*k;
     if(j<n) //if there are 2 children
        {
        if(h[j]<h[j+1]) //checking both children
                        j=j+1;
        }
        if(v>=h[j])
        {
                heapify=1;
        }
        else
                {
                 h[k]=h[j];
           k=j;
                }
            }
         h[k]=v;
        }

 }
void heapsort(int h[],int n)
 {
  int i,temp,j;
  for(i=n;i>1;i--)
    {

  heapifybottomup(h,i);

          temp=h[1];
          h[1]=h[i];
```

```
            h[i]=temp;

        }
    }
int main()
  {
    int i,n,h[20];
//    clrscr();
    printf("\n Enter the number of elements : ");
    scanf("%d",&n);
    printf("\n Enter the elements : ");
    for(i=1;i<=n;i++)
      scanf("%d",&h[i]);
    printf("\n The Entered array: ");
    for(i=1;i<=n;i++)
    {
      printf("%d\t",h[i]);
    }

    heapsort(h,n);

    printf("\n The sorted array : ");
    for(i=1;i<=n;i++)
    {
      printf("%d\t",h[i]);
    }
    getch();
    return 0;
  }
```

9. Mergesort

ALGORITHM Mergesort(A[0..n-1])
//Sorts array A[0..n-1] by recursive mergesort
//Input:An array A[0..n-1] of orderable elements
//Output:Array A[0..n-1] sorted in nondecreasing order

if n>1
    copy A[0.. ⌊n/2⌋ -1] to B[0.. ⌊n/2⌋ -1]
    copy A[⌊n/2⌋ ..n-1] to C[0.. ⌈n/2⌉ -1]
    Mergesort(B[0.. ⌊n/2⌋ -1])
    Mergesort(C[0.. ⌈n/2⌉ -1)
    Merge(B,C,A)

ALGORITHM Merge(B[0..p-1],C[0..q-1],A[0..p+q-1])
//Merges two sorted arrays into one sorted array
//Input:Arrays B[0..p-1] and C[0..q-1] both sorted
//Output:Sorted array A[0..p+q-1] of the elements of B and C

```
i ← 0; j ← 0; k ← 0
while i<p and j<q do
        if B[i] ≤ C[j]
             A[k] ← B[i]; i ← i+1
         else A[k] ← C[j]; j ← j+1
         k ← k+1
if i=p
     copy C[j..q-1]  to A[k..p+q-1]
else copy B[i..p-1] to A[k..p+q-1]
```

PROGRAM

```
 #include<stdio.h>
#include<conio.h>
void MergeSort(int [], int);
void Merge(int [],int, int [],int, int []);

void Merge(int b[], int p, int c[], int q, int a[])
{
  int i=0, j=0,k=0;
          if b[i]<=c[j]
          {
                  a[k]=b[i];
                  i++;
          }
          else
          {
                  a[k]=c[j];
                  j++;
          }
          k++;
  }

  while(i<p)
  {
          a[k]=b[i];
          i++;
          k++;
  }

   while(j<q)
  {
          a[k]=c[j];
          j++;
          k++;
  }
}
void MergeSort(int a[], int n)
{
```

```
 int i,j,k,p,q;
 int b[10],c[10];
  if(n>1)
  {
          for(i=0;i<n/2;i++)
          {
                  b[i]=a[i];
          }

          p=i;

          for(i=n/2,j=0;i<n;i++,j++)
          {
                  c[j]=a[i];
          }

          q=j;

          MergeSort(b,p);
          MergeSort(c,q);
          Merge(b,p,c,q,a);
    }
}
void main()
{
 int i,a[20],n;
 clrscr();
 printf("Enter no of elements\n");
 scanf("%d",&n);

 printf("Enter the nos one by one...\n");
 for(i=0;i<n;i++)
   scanf("%d",&a[i]);

 MergeSort(a,n);

 printf("The numbers in ascending order are...\n");
 for(i=0;i<n;i++)
   printf("%d\n",a[i]);
 getch();
}
```

10. Quicksort

ALGORITHM Quicksort(A[l..r])
//Sorts a subarray by quicksort
//Input: A subarray A[l..r] of A[0..n-1],defined by its left and right indices l and r
//Output: The subarray A[l..r] sorted in nondecreasing order

if l<r

s ← Partition (A[l..r]) //s is the split position
Quicksort(A[l....s-1])
Quicksort(A[s+1..r])

ALGORITHM Partition(A[l..r])
//Partitions a subarray by using its first element as pivot
//Input: A subarray A[l..r] of A[0..n-1], defined by its left and right indices l and r (l<r)
//Output: A partition of A[l..r], with split position returned as this function's value

p ← A[l]
i ← l; j ← r+1
repeat
  repeat i ← i+1 **until** A[i]≥ p
  repeat j ← j-1 **until** A[j] ≤p
  swap(A[i],A[j])
until i≥j
swap(A[i],A[j]) //undo last swap when i≥j
swap(A[l],A[j])
return j

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int a[20];
void QuickSort(int, int);
int partition(int, int);

int partition(int l, int r)
{
 int i,j,p,temp;
 p=a[l];
 i=l;
 j=r+1;
 do
 {
        do
        {
                i++;
        }while(a[i]<=p);

        do
        {
                j--;
        }while(a[j]>p);

        temp=a[i];
        a[i]=a[j];
        a[j]=temp;

 }while(i<=j);
```

```
  temp=a[i];            /* undo last swap*/
  a[i]=a[j];
  a[j]=temp;

  temp=a[l];            /* swap pivote element and a[j] */
  a[l]=a[j];
  a[j]=temp;

  return j;
}
void QuickSort(int l, int r)
{
 int s;
 if(l<r)
 {
        s=partition(l,r);
        QuickSort(l,s-1);
        QuickSort(s+1,r);
  }
}
void main()
{
 int i,n;
 clrscr();
 printf("Enter no of elements\n");
 scanf("%d",&n);

 printf("Enter the nos one by one...\n");
 for(i=0;i<n;i++)
   scanf("%d",&a[i]);

 QuickSort(0,n-1);

 printf("The numbers in ascending order are...\n");
 for(i=0;i<n;i++)
   printf("%d\n",a[i]);
 getch();
}
```

11. Topological Sorting

In a digraph if we can list its vertices in such an order that ,for every edge in the graph, the vertex where the edge starts is listed before the vertex where the edge ends then, such a problem is called **Topological Sorting**

Write a C/C++ program to implement DFS based Topological Sorting algorithm for a given digraph.

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int A[20][20],visited[20],count=0,n,order[10],x=0;

void dfs(int v)
{
 int w;
count=count+1;
 visited[v]=count;
 //printf("%d is visited \n",v);
 for(w=1; w<=n; w++)
 {
  if(A[v][w] && visited[w]==0)
  {
    dfs(w);
  }
 }
 //printf("%d is popped \n",v);
order[x]=v;
x++;
}

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

```
//  clrscr();
  printf("Enter the number of vertices of the graph\n");
  scanf("%d",&n);
  printf("Enter the number of edges of the graph:\n");
  scanf("%d",&e);
 for(i=1;i<=n;i++)
  {
     for(j=1;j<=n;j++)
            {

               A[i][j]=0;
     }
  }

printf("Enter the edges of the graph one by one:\n");
  for(i=1;i<=e;i++)
  {
     scanf("%d %d",&v1,&v2);

       A[v1][v2]=1;
  }

  for(i=1;i<=n;i++)
  {
     visited[i]=0;
  }
  for(k=1;k<=n;k++)
```

```
    {
        if(visited[k]==0)
        dfs(k);
    }

    printf("The topological ordering is:\n");
    for(i=n-1;i>=0;i--)
    {
        printf("%d\t",order[i]);
    }

     getch();
    return 0;
    }
```

12. Write a C/C++ program to implement Topological Sorting algorithm for a given digraph using source removal technique.

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
int order[10],A[10][10],n,in[10],visited[10],e,k=0;
int is_all_nodes_visited()
 {
   int i;
 for(i=1;i<=n;i++)
 {
    if (visited[i]==0)
    return 0;
 }
 return 1;
 }
```

```
int  main()
 {
  int i,j,v1,v2,w;;


  printf("Enter no of nodes\n");
  scanf("%d",&n);
  printf("Enter the number of edges of the graph:\n");
  scanf("%d",&e);
  for(i=1;i<=n;i++)
  {
          for(j=1;j<=n;j++)
       {

          A[i][j]=0;
       }
  }
  for(i=1;i<=n;i++)
  {
     in[i]=0;
  }

  for(i=1;i<=n;i++)
  {
   visited[i]=0;
  }

 printf("Enter the edges of the graph one by one:\n");
          for(i=1;i<=e;i++)
          {
       scanf("%d %d",&v1,&v2);

                   A[v1][v2]=1;
          }

  for(i=1;i<=n;i++)
   {
    for(j=1;j<=n;j++)
     {
     if(A[i][j]==1)
       in[j]=in[j]+1;
     }
   }

while(1)
{
   if(is_all_nodes_visited())
    break;
   else
   for(i=1;i<=n;i++)
    {
```

```
    if(in[i]==0 && visited[i]==0)
     {
        order[k]=i;

        k++;
        visited[i]=1;
        //printf("%d is visited \n",i);
        for(w=1;w<=n;w++)
        {
           if(A[i][w]==1 && visited[w]==0)
           {
              in[w]--;
         }
        }

     }
   }

}


   printf("The Topological ordering is:\n");
   for(i=0;i<n;i++)
   {
     printf("%d\t",order[i]);
   }




//getch();
return 0;
}
```

OR

```
#include <stdio.h>
#include <stdlib.h>
int n,e, in[30], A[30][30],k,order[30],visited[30];
void topo()
{
int i,w;
k=0;
for(i=1;i<=n;i++)
{
if(in[i]==0 && visited[i]==0)
{
        order[k]=i;
        k++;
        visited[i]=1;
```

```
for(w=1;w<=n;w++)
        {
           if(A[i][w]==1 && visited[w]==0)
           {
              in[w]--;
         }
        }
i=0;
}
}
}
int main()
{
int i,j,v1,v2;
printf("Enter the the no. of vertices\n");
scanf("%d",&n);
printf("enter the number of edges\n");
scanf("%d",&e);
for(i=0;i<=n;i++)
for(j=0;j<=n;j++)
{

   A[i][j]=0;
}
printf("enter edges one by one\n");
for(i=1;i<=e;i++)
{
scanf("%d%d",&v1,&v2);
A[v1][v2]=1;
}
for(i=1;i<=n;i++)
{
in[i]=0;
}
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
if(A[i][j]==1)
{
in[j]=in[j]+1;
}
}
}
topo();
if(k==n)
{
printf("Topological order is\n");
for(i=0;i<n;i++)
{
printf("%d\n",order[i]);
```

```
}
}
else
{
printf("The graph is cyclic graph Topological Sort not possible\n");
}
return 0;
}
```

13. Hosrpool's Algorithm

In time and space tradeoffs technique if time is at premium, we can precompute the function's values and store then in a table. In somewhat more general terms , the idea is to preprocess the problem's input, in whole or in part, and store the additional information obtained to accelerate solving the problem afterward. We call this approach input enhancement.

The algorithm considered under Space and Time Tradesoffs is

- Horspool's Algorithm

Horspool's Algorithm

ALGORITHM ShiftTable(P[0..m-1])
//Fills the shift table used by Horspool's algorithm
//Input: Pattern P[0..m-1] and alphabet of possible characters
//Output: Table[0..size-1] indexed by the alphabet's characters and filled with shift sizes the given formula

$$T(c) = \begin{cases} \text{the pattern's length m,} \\ \text{if c is not among the first m-1 characters of the pattern} \\ \\ \text{the distance from the rightmost c among the first m-1 characters} \\ \text{of the pattern to its last character ,otherwise} \end{cases}$$

initialize all elements of Table with m
for j ← 0 **to** m-2 **do** Table[P[j]] ← m-1-j
return Table

ALGORITHM HorspoollMatching(P[0..m-1],T[0..n-1])
//Implements Horspool's algorithm for string matching
//Input: Pattern P[0..m-1] and Text T[0..n-1]
//Output: The index of the left end of the first matching substring
//or -1 if there are no matches
ShiftTable(P[0..m-1]) // generate Table of shifts
i ← m-1
while i≤n-1 do
    k ← 0
    while k≤m-1 and P[m-1-k]=T[i-k] **do**
        k ← k+1

```
    if k=m
        return i-m+1
   else i ← i+Table[T[i]]
return -1
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

char p[50],t[50];
int table[256];
int m,n;

void shifttable()
{
  int i,j,temp;
  for(i=0;i<256;i++)
  {
    table[i]=m;

  }
  for(j=0;j<m-1;j++)
  {
    temp=p[j];
    table[temp]=m-1-j;
  }
}
int horspool()
{
  int i,k;
  int temp;

  shifttable();
  i=m-1;
  while(i<=n-1)
  {
    k=0;
    while((k<=m) && (p[m-1-k]==t[i-k]))
      k++;
    if(k==m)
      return (i-m+1);
    else
       temp=t[i];
      i+=table[temp];
  }
  return -1;
}
int main()
```

```
{
   int r;
   printf("Enter the text\n");
   scanf("%s",t);
   n=strlen(t);
   printf("Enter the pattern\n");
   scanf("%s",p);
   m=strlen(p);
   r=horspool();
   if(r==-1)
      printf("Pattern not found\n");
  else
      printf("Pattern found at position %d\n",r);
 getch();
}
```

14. Sub set Problems

Write a C/C++ program to implement Subset Sum problem using Backtracking technique.

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>

int n,x[20],w[20],count=0,d;

void subsetsum(int s,int k,int r)
{

   int i;
   x[k]=1;
   if(s+w[k]==d)
   {

      count++;
                printf("\nSolution No %d:\n", count);
                for(i=1;i<=k;i++)
                {
                        if(x[i]==1)
        printf("%d\t", w[i]);
                }
   }
   else
      if(s+w[k]+w[k+1]<=d)
   {
      subsetsum(s+w[k],k+1,r-w[k]);
```

```
    }
    if((s+(r-w[k]))>=d && (s+w[k+1])<=d)
    {
       x[k]=0;
       subsetsum(s,k+1,r-w[k]);
    }
}

int main()
{
    int i,j,sum=0;
     printf("Enter the number of elements\n");
       scanf("%d",&n);
       for (i=1;i<=n;i++)
       {
         x[i]=0;
       }
       printf("Enter the  elements in ascending order\n");
       for (i=1;i<=n;i++)
       {
         scanf("%d",&w[i]);
       }
       printf("Enter the  sum\n");
       scanf("%d",&d);
       for (i=1;i<=n;i++)
       {
         sum=sum+w[i];
       }

       if(sum<d)
       printf("No solution\n");
       else
          subsetsum(0,1,sum);

    return 0;
}
```

Design and Analysis of Algorithms

LAB MANUAL

Subject code: CS402
IV Semester B.E.
(2012)

Computer Science & Engineering Department
NMAMIT, Nitte

PART B

1. Write a C/C++ program to implement Selection Sort algorithm.

ALGORITHM SelectionSort(A[0..n-1])
//The algorithm sorts a given array by selection sort
//Input: An array A[0..n-1] of orderable elements
//Output: Array A[0..n-1] sorted in ascending order

for i ← 0 **to** n-2 **do**
  min← i
  for j ← i+1 **to** n-1 **do**
    if A[j]<A[min] min ← j
 swap A[i] and A[min]

PROGRAM

```
#include<stdio.h>
#include<conio.h>

int findmax(int[],int);
void exchange(int [],int);

int  main()
{
    int array[10];
    int i,j,n,temp;
    printf("Enter the size of the array\n");
    scanf("%d",&n);
    printf("Enter the array elements\n");
    for(i=0;i<n;i++)
    {
       scanf("%d",&array[i]);
    }
    printf("The array elements entered are\n");
    for(i=0;i<n;i++)
    {
      printf("%d\n",array[i]);
    }
    exchange(array,n);
    printf("The sorted array is\n");
    for(i=0;i<n;i++)
    {
       printf("%d\n",array[i]);
    }
 getch();
 }
int findmax(int b[10],int k)
{
    int max=0,j;
    for(j=1;j<=k;j++)
```

```
    {
       if(b[j]>b[max])
       {
         max=j;
       }
    }
   return(max);
 }
 void exchange(int b[10],int k)
 {
     int temp,big,j;
     for(j=k-1;j>=1;j--)
   {
         big=findmax(b,j);
         temp=b[big];
         b[big]=b[j];
         b[j]=temp;
  }

 }
```

2. Write a C/C++ program to implement bubble Sort algorithm.

ALGORITHM BubbleSort(A[0..n-1])
//The algorithm sorts array A[0..n-1] by bubble sort
//Input: An array A[0..n-1] of orderable elements
//Output: Array A[0..n-1] sorted in ascending order

for i← 0 **to** n-2 **do**
    for j ← 0 **to** n-2-i **do**
        if A[j+1]<A[j] swap A[j] and A[j+1]

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<time.h>

void bubble(int [],int);
void main()
{
    int a[1000],n,i;
    clock_t st,et;
    clrscr();
    printf("Enter the array size\n");
    scanf("%d",&n);
    printf("Enter the elements of the array\n");
    for(i=0;i<n;i++)
```

```
   {
      a[i]=rand()%100; //random numbers from 0 to 99
    }
   printf("The array elements before sorting are\n");
   for(i=0;i<n;i++)
   {
     printf("%d ",a[i]);
   }

 st=clock(); //starting clock ticks
 for(i=0;i<5000;i++)   //looping the same algorithm
 {
     bubble(a,n);
  }
 et=clock();//ending clock ticks
 printf("The array elements after sorting are\n");
 for(i=0;i<n;i++)
{
    printf("%d ",a[i]);
 }
 printf("The time complexity is %f secs\n",(double)(et-st)/CLK_TCK);
//CLOCKS_PER_SEC(IN C++) they are processor clock ticks per sec
 getch();
}
void bubble(int a[],int size)
{
   int i,j,temp;
 for(i=0;i<size-1;i++)
 {
    for(j=0;j<size-i-1;j++)
    {
    if(a[j]>a[j+1])
    {
       temp=a[j];
      a[j]=a[j+1];
      a[j+1]=temp;
   }
  }
 }
}
```

Note: Time complexity can be calculated in the same way for the rest of the programs.

3. Write a C/C++ program to implement Brute Force String Match algorithm.

ALGORITHM BruteForceStringMatch(T[0..n-1], P[0..m-1])
//The algorithm implements brute-force string matching
//Input: An array T[0..n-1] of n characters representing a text
// and array P[0..m-1]of m characters representing a pattern.
//Output: The position of the first character in the text that starts the first matching

//substring if the search is unsuccessful and -1 otherwise.

```
for i ← 0 to n-m do
    j ← 0
    while j<m and P[j]=T[i+j] do
          j ← j+1
    if j=m return i
return -1
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<string.h>

int brute(char [],char [],int ,int );

int  main()
{
     char text[100],pattern[50];int i,j,m,n,val;
     printf("Enter the text\n");
     scanf("%s",text);
     n=strlen(text);
    printf("Enter the pattern\n");
    scanf("%s",pattern);
    m=strlen(pattern);
    val=brute(text,pattern,n,m);
    if(val==-1)
   {
       printf("The string matching is unsuccessful\n");
    }
  else
  {
      printf("The position of the first character in the text\n that starts the first matching substring
is %d\n",val+1);
}
getch();
}

int brute(char str1[100],char str2[50],int len1,int len2)
{
      int i,j;
     for(i=0;i<=len1-len2;i++)
     {
           j=0;
          while(j<len2 && str2[j]==str1[i+j])
          {
                j=j+1;
           }
            if(j==len2)
```

```
            return i;
    }
    return -1;
}
```

4. Write a C/C++ program to implement Binary search algorithm

ALGORITHM BinarySearch(A[0..n-1],K)
//Implements nonrecursive binary search
//Input: An array A[0..n-1] sorted in ascending order and a search key K
//Output: An index of the array's elements that is equal to K or -1 if there is no such element

l← 0; r ← n-1
while l ≤ r do
    m ← ⌊(l+r)/2⌋
    if K=A[m] **return** m
    else if K< A[m] r← m-1
    else l ← m+1
return -1

PROGRAM

```
#include<stdio.h>
int binsearch(int, int);
int a[20],key;

int binsearch(int low, int high)
{
 int mid;
 while(low<=high)
 {
        mid=(low+high)/2;
        if(key==a[mid])
           return mid;
        else if(key<a[mid])
           high=mid-1;
        else
           low=mid+1;
  }
return -1;
}
void main()
{
 int n,i,pos;
 clrscr();
 printf("Enter no of elements\n");
 scanf("%d",&n);
 printf("Enter the nos one by one in ascending order\n");
 for(i=0;i<n;i++)
```

```
  scanf("%d",&a[i]);
 printf("Enter the key to be searched\n");
 scanf("%d",&key);
 pos=binsearch(0,n-1);
 if(pos==-1)
        printf("Element not found\n");
 else
        printf("Element found at position %d\n",pos+1);
 getch();
}
```

5. Write a C/C++ program to implement insertion sort algorithm

ALGORITHM InsertionSort(A[0..n-1])
//Sorts a given array by insertion sort
//Input: An array a[0..n-1] of n orderable elements
//Output: Array A[0..n-1] sorted in nondecreasing order

for i ← 1 **to** n-1 **do**
    v ← A[i]
    j ← i-1
    while j≥0 and A[j]>v **do**
        A[j+1] ← A[j]
        j← j-1
    A[j+1] ←v

PROGRAM

```
#include<stdio.h>
#include<conio.h>
void InsertionSort(int [], int);
void InsertionSort(int a[], int n)
{
 int i, j, v;
 for(i=1;i<n;i++)
 {
        v=a[i];
        j=i-1;
        while(j>=0 && a[j]>v)
        {
                a[j+1]=a[j];
                j--;
        }
        a[j+1]=v;
 }
}
void main()
```

```
{
 int i,n,a[20];
 clrscr();
 printf("Enter no of elements\n");
 scanf("%d",&n);
 printf("Enter the nos one by one...\n");
 for(i=0;i<n;i++)
   scanf("%d",&a[i]);
 InsertionSort(a,n);
 printf("The numbers in ascending order are...\n");
 for(i=0;i<n;i++)
   printf("%d\n",a[i]);
 getch();
}
```

6. Write a C/C++ program to implement the traversal of a graph (undirected graph or digraph) using DFS technique.

ALGORITHM DFS(G)

```
//Implements a depth first traversal of a given graph
//Input:Graph G= <V,E>
//Output:Graph G with its vertices marked with consecutive integers
//in the order they have been first encountered by the DFS traversal

mark each vertex in V with 0 as a mark of being "unvisited"
count ← 0
for each vertex v in V do
     if v is marked with 0
        dfs(v)

dfs(v)
//visits recursively all the unvisited vertices connected to vertex v and
//assigns them the numbers in the order they are encountered
//via global variable count
count ← count +1; mark v with count
for each vertex w in V adjacent to v do
    if w is marked with 0
        dfs(w)
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int A[20][20],visited[20],count=0,n;

void dfs(int v)
{
 int w;
count=count+1;
```

```
 visited[v]=count;
 printf("%d is visited \n",v);
 for(w=1; w<=n; w++)
 {
  if(A[v][w] && visited[w]==0)
  {
    dfs(w);
  }
 }
}

int main()
{
  int i,j,k,v1,v2,e;
//  clrscr();
  printf("Enter the number of vertices of the graph\n");
  scanf("%d",&n);
  printf("Enter the number of edges of the graph:\n");
  scanf("%d",&e);
 for(i=1;i<=n;i++)
  {
          for(j=1;j<=n;j++)
            {

          A[i][j]=0;
          }
  }

printf("Enter the edges of the graph one by one:\n");
  for(i=1;i<=e;i++)
  {
     scanf("%d %d",&v1,&v2);
          A[v1][v2]=A[v2][v1]=1;//undirected graph
//A[v1][v2]=1//directed graph
  }

  for(i=1;i<=n;i++)
  {
     visited[i]=0;
  }
  for(k=1;k<=n;k++)
  {
     if(visited[k]==0)
          dfs(k);
}
   getch();
  return 0;
  }
```

7. Write a C/C++ program to check which vertices are reachable from a given vertex using DFS traversal technique.

PROGRAM

```
#include<stdio.h>

#include<conio.h>

int A[20][20],visited[20],count=0,n;


void dfs(int v)

{

 int w;

count=count+1;

 visited[v]=count;

 printf("%d  \n",v);

 for(w=1; w<=n; w++)

 {

  if(A[v][w] && visited[w]==0)

  {

    dfs(w);

  }

 }

}


int main()

{

  int i,j,v1,v2,s,e;


  printf("Enter the number of vertices of the graph:\n");

  scanf("%d",&n);
```

```
  printf("Enter the number of edges of the graph:\n");
  scanf("%d",&e);
for(i=1;i<=n;i++)
  {
          for(j=1;j<=n;j++)
            {

          A[i][j]=0;
          }
  }

printf("Enter the edges of the graph one by one:\n");
  for(i=1;i<=e;i++)
  {
     scanf("%d %d",&v1,&v2);
          A[v1][v2]=A[v2][v1]=1;//undirected graph
//A[v1][v2]=1//directed graph
  }

  for(i=1;i<=n;i++)
  {
     visited[i]=0;
  }
  printf("Enter the vertex:\n");
  scanf("%d",&s);
  printf("The vertices reachable from %d are:\n" ,s);
          dfs(s);
```

```
 getch();
 return 0;
 }
```

8. a)Write a C/C++ program to check if the given graph is connected or not using DFS traversal technique(undirected).

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int A[20][20],visited[20],con=0,n;

void dfs(int v)
{
 int w;

 visited[v]=1;
 printf("%d is visited \n",v);
 for(w=1; w<=n; w++)
 {
  if(A[v][w] && visited[w]==0)
  {
    dfs(w);
  }
 }
}

int main()
```

```
{
  int i,j,k,e,v1,v2;
//  clrscr();
  printf("Enter the number of vertices of the graph\n");
  scanf("%d",&n);
  printf("Enter the number of edges of the graph:\n");
  scanf("%d",&e);
 for(i=1;i<=n;i++)
  {
          for(j=1;j<=n;j++)
            {

          A[i][j]=0;
          }
  }

printf("Enter the edges of the graph one by one:\n");
  for(i=1;i<=e;i++)
  {
     scanf("%d %d",&v1,&v2);
          A[v1][v2]=A[v2][v1]=1;//undirected graph
//A[v1][v2]=1//directed graph
  }

  for(i=1;i<=n;i++)
  {
     visited[i]=0;
```

```
  }
  for(k=1;k<=n;k++)
  {
     if(visited[k]==0)
{
printf("DFS traversal starts from node: %d\n",k);
          con++;
          dfs(k);
}
}

if(con==1)
printf("The graph is connected\n");
else
printf("The graph is not connected\n");

  getch();
  return 0;
  }
```

8 b) Write a C/C++ program to check if the given graph is connected or not using DFS traversal technique(digraph).

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int A[20][20],visited[20],con=0,n;

void dfs(int v)
{
 int w;

 visited[v]=1;
 printf("%d is visited \n",v);
```

```
 for(w=1; w<=n; w++)
 {
  if(A[v][w] && visited[w]==0)
  {
    dfs(w);
  }
 }
}

int main()
{
  int i,j,k,e,v1,v2;
  int flag;
  //clrscr();
  printf("Enter the number of vertices of the graph\n");
  scanf("%d",&n);
  printf("Enter the number of edges of the graph:\n");
  scanf("%d",&e);
 for(i=1;i<=n;i++)
  {
          for(j=1;j<=n;j++)
            {

          A[i][j]=0;
          }
  }

printf("Enter the edges of the graph one by one:\n");
  for(i=1;i<=e;i++)
  {
     scanf("%d %d",&v1,&v2);
          //A[v1][v2]=A[v2][v1]=1//undirected graph
A[v1][v2]=1;//directed graph
  }


  for(k=1;k<=n;k++)
  {
     flag=0;//assume starting with k all other vertices are reachable
          for(i=1;i<=n;i++)
                  {visited[i]=0;
                  }
                  printf("DFS traversal from node %d:\n",k);
                  dfs(k);
                          for(i=1;i<=n;i++)
                          {
                                  if(visited[i]==0)
                                  {flag=1;
                                  break;
                                  }
                          if(flag==0)
```

```
                        con=con+1;
                        }
}
if(con==n)
printf("The graph is strongly connected\n");
else if(con==0)
printf("The graph is not connected\n");
else
printf("The graph is weakly connected\n");

 getch();
 return 0;
 }
```

9. Write a C/C++ program to Implement the traversal of a graph (undirected graph or digraph) using BFS technique

Breadth First Search

ALGORITHM BFS(G)

```
//Implements a breadth-first search traversal of a given graph
//Input: Graph G=<V,E>
//Output:Graph G with its vertices marked with consecutive integers
//in the order they have been visited by the BFS traversal

mark each vertex in V with 0 as a mark of being "unvisited"
count ← 0
for each vertex v in V do
    if v is marked with 0
      bfs(v)

bfs(v)
//visits all the unvisited vertices connected to vertex v
///and assign them the numbers in the order they are visited
//via global variable count

count ← count +1; mark v with count and initialize a queue with v
while the queue is not empty do
    for each vertex w in V adjacent to the front vertex do
        if w is marked with 0
            count ← count + 1; mark w with count
            add w to the queue
    remove the front vertex from the queue
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
int q[50],front=1,rear=0,n, count=0,i,j,A[20][20],visited[20];
```

```
void bfs(int v)
{
int q[50],front=1,rear=0,w;
count=count+1;
visited[v]=count;
printf("%d is visited\n",v);
q[++rear]=v;//insert v to queue
while(front<=rear)
{
v=q[front];
for(w=1;w<=n;w++)
        {
                if(visited[w]==0&&A[v][w]==1)
                {
                   count++;
                        visited[w]=count;
                        printf("%d is visited\n",w);
                        q[++rear]=w;

                }
        }
front++;

}
}
int main()
{
int v1,v2,e,k;

printf("Enter the number of vertices of the graph\n");
  scanf("%d",&n);
  printf("Enter the number of edges of the graph:\n");
  scanf("%d",&e);
 for(i=1;i<=n;i++)
  {
        for(j=1;j<=n;j++)
          {

        A[i][j]=0;
        }
  }

printf("Enter the edges of the graph one by one:\n");
        for(i=1;i<=e;i++)
        {
                scanf("%d %d",&v1,&v2);
                A[v1][v2]=A[v2][v1]=1;//undirected graph
                //A[v1][v2]=1//directed graph
        }
```

```
        for(i=1;i<=n;i++)
        {
                visited[i]=0;
        }
        for(k=1;k<=n;k++)
        {
                 if(visited[k]==0)
                bfs(k);
        }

getch();
return 0;
}
```

10. Write a C/C++ program to check which vertices are reachable from a given vertex using BFS traversal technique.

```
#include<stdio.h>

#include<conio.h>

int q[50],front=1,rear=0,n, count=0,i,j,A[20][20],visited[20];



void bfs(int v)

{

int q[50],front=1,rear=0,w;

count=count+1;

visited[v]=count;

printf("%d \n",v);

q[++rear]=v;//insert v to queue

while(front<=rear)

{

v=q[front];

for(w=1;w<=n;w++)

        {

                if(visited[w]==0&&A[v][w]==1)
```

```
                {
                   count++;
                        visited[w]=count;
                        printf("%d\n",w);
                        q[++rear]=w;

                }
          }
front++;

}
}

int main()
{
int v1,v2,e,s;

printf("Enter the number of vertices of the graph\n");
  scanf("%d",&n);
  printf("Enter the number of edges of the graph:\n");
  scanf("%d",&e);
 for(i=1;i<=n;i++)
  {
          for(j=1;j<=n;j++)
           {
```

```
            A[i][j]=0;
            }
 }

printf("Enter the edges of the graph one by one:\n");
            for(i=1;i<=e;i++)
            {
                        scanf("%d %d",&v1,&v2);
                        A[v1][v2]=A[v2][v1]=1;//undirected graph
                        //A[v1][v2]=1//directed graph
            }

            for(i=1;i<=n;i++)
            {
                        visited[i]=0;
            }
            printf("Enter the vertex:\n");
             scanf("%d",&s);
             printf("The vertices reachable from %d are :\n",s);
                        bfs(s);

getch();
return 0;
}
```

11. a)Write a C/C++ program to check if the given graph is connected or not using BFS traversal technique(undirected).

```
#include<stdio.h>
#include<conio.h>
int q[50],front=1,rear=0,n, count=0,i,j,A[20][20],visited[20];

void bfs(int v)
{
int q[50],front=1,rear=0,w;
//count=count+1;
visited[v]=1;
printf("%d is visited\n",v);
q[++rear]=v;//insert v to queue
while(front<=rear)
{
v=q[front];
for(w=1;w<=n;w++)
        {
                if(visited[w]==0&&A[v][w]==1)
                {
                   //count++;
                        visited[w]=1;
                        printf("%d is visited\n",w);
                        q[++rear]=w;

                }
        }
front++;
```

```
}
}
int main()
{
int v1,v2,e,con,k;

printf("Enter the number of vertices of the graph\n");
 scanf("%d",&n);
 printf("Enter the number of edges of the graph:\n");
 scanf("%d",&e);
for(i=1;i<=n;i++)
 {
        for(j=1;j<=n;j++)
         {

        A[i][j]=0;
        }
 }

printf("Enter the edges of the graph one by one:\n");
        for(i=1;i<=e;i++)
        {
                scanf("%d %d",&v1,&v2);
                A[v1][v2]=A[v2][v1]=1;//undirected graph
                //A[v1][v2]=1//directed graph
```

```
        }

        for(i=1;i<=n;i++)
            {
                visited[i]=0;
            }
        for(k=1;k<=n;k++)
            {
                if(visited[k]==0)
                {
                    printf("BFS traversal starts from node: %d\n",k);
                    con++;
                    bfs(k);
                }
            }

    if(con==1)
    printf("The graph is connected\n");
    else
    printf("The graph is not connected\n");

    getch();
    return 0;
    }
```

11 b) Write a C/C++ program to check if the given graph is connected or not using BFS traversal technique(directed).

```
#include<stdio.h>
```

```
#include<conio.h>
int q[50],front=1,rear=0,n, count=0,i,j,A[20][20],visited[20];

void bfs(int v)
{
int q[50],front=1,rear=0,w;
//count=count+1;
visited[v]=1;
printf("%d is visited\n",v);
q[++rear]=v;//insert v to queue
while(front<=rear)
{
v=q[front];
for(w=1;w<=n;w++)
        {
                if(visited[w]==0&&A[v][w]==1)
                {
                  //count++;
                        visited[w]=1;
                        printf("%d is visited\n",w);
                        q[++rear]=w;

                }
        }
front++;

}
}
int main()
{
int v1,v2,e,con=0,flag,k;

printf("Enter the number of vertices of the graph\n");
 scanf("%d",&n);
 printf("Enter the number of edges of the graph:\n");
 scanf("%d",&e);
for(i=1;i<=n;i++)
 {
        for(j=1;j<=n;j++)
         {

        A[i][j]=0;
        }
 }

printf("Enter the edges of the graph one by one:\n");
        for(i=1;i<=e;i++)
        {
```

```
                scanf("%d %d",&v1,&v2);
                A[v1][v2]=A[v2][v1]=1;//undirected graph
                //A[v1][v2]=1//directed graph
        }


        for(k=1;k<=n;k++)
 {
    flag=0;//assume starting with k all other vertices are reachable
        for(i=1;i<=n;i++)
                {
                   visited[i]=0;
                }
                printf("BFS traversal from node %d:\n",k);
                bfs(k);
                        for(i=1;i<=n;i++)
                        {
                                if(visited[i]==0)
                                {flag=1;
                                break;
                                }
                        if(flag==0)
                        con=con+1;
                        }
}
if(con==n)
printf("The graph is strongly connected\n");
else if(con==0)
printf("The graph is not connected\n");
else
printf("The graph is weakly connected\n");


getch();
return 0;
}
```

12. Computing a Binomial Coefficeint

Write a C/C++ program to find Binomial Coefficient using Dynamic Programming technique.

ALGORITHM Binomial(n,k)

// Computes C(n,k) by the dynamic programming algorithm

//Input: A pair of nonnegative integers $n \geq k \geq 0$

//Output: The value of C(n,k)

for i ← 0 **to** n **do**

  for j ← 0 to min(i,k) **do**

```
        if j=0 or j=i
            C[i,j] ← C[i-1,j-1] + C[i-1,j]
return C[n,k]
```

PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int c[20][20];

int min(int i,int k)
   {
    if (i<k )return i;
      else return k ;
   }

void binomial(int n,int k)
{
 int i,j;
 for(i=0;i<=n;i++)
 {
  for(j=0;j<=min(i,k);j++)
  {
   if(j==0||j==i)
                          c[i][j]=1;
   else
                         c[i][j]=c[i-1][j-1]+c[i-1][j];
  }
 }
}
void main()
{
 int n,k,i,j;
 //clrscr();
 printf("Enter the value of n\n");
 scanf("%d",&n);
 printf("Enter the value of k\n");
 scanf("%d",&k);
 if(n<k)
  printf("Invalid input: n cannot be less than k\n");
 else if(k<0)
  printf("Invalid input: k cannot be less than 0\n");
 else
 {
  binomial(n,k);
  printf("Computed matrix is \n");
  for(i=0;i<=n;i++)
  {
   for(j=0;j<=min(i,k);j++)
```

```
                        printf("%d\t",c[i][j]);
     printf("\n");
    }
   printf("binomial coefficient c[%d,%d]=%d\n",n,k,c[n][k]);
  }
 getch();
}
```

13. Warshall's Algorithm

ALGORITHM Warshall(A[1..n,1..n])
//Implements Warshall's algorithm for computing the transitive closure
//Input: The adjacency matrix A of a digraph with n vertices
//Output: The transitive closure of the digraph

$\mathbf{R}^{(0)} \leftarrow A$
for k ← 1 **to** n **do**
    for i ← 1 **to** n **do**
        for j ← 1 **to** n **do**
            $\mathbf{R}^{(k)}[i,j] \leftarrow \mathbf{R}^{(k-1)}[i,j]$ or $\mathbf{R}^{(k-1)}[i,k]$ and $\mathbf{R}^{(k-1)}[k,j]$
return $\mathbf{R}^{(n)}$

PROGRAM

```
/*Warshall's algorithm for computing the Transitive Closure of a digraph*/
#include<stdio.h>
#include<conio.h>
int A[10][10],n,e,R[10][10];
void Warshall_Transitive()
{
 int i,j,k;
 for(i=1;i<=n;i++)                  // initially R = A
         for(j=1;j<=n;j++)
                 R[i][j]=A[i][j];

 for(k=1;k<=n;k++)
   for(i=1;i<=n;i++)
         for(j=1;j<=n;j++)
             R[i][j]=R[i][j] | (R[i][k] & R[k][j]);
 }

int main()
{
  //A-given graph; R - will have Transitive Closure
 int v1,v2;
 int i,j;
 //clrscr();

 printf("Enter no of vertices\n");
 scanf("%d",&n);
```

```
printf("Enter no of edges\n");
scanf("%d",&e);

for(i=1;i<=n;i++)                    //initialization of adjacency matrix
        for(j=1;j<=n;j++)
                A[i][j]=0;

printf("Enter the Edges one by one\n");
for(i=1;i<=e;i++)
{
        printf("Edge %d: ",i);
        scanf("%d%d",&v1,&v2);
        A[v1][v2]=1;
}


Warshall_Transitive();

printf("The given digraph is:\n");
for(i=1;i<=n;i++)
{
        for(j=1;j<=n;j++)
                printf("%d\t",A[i][j]);
        printf("\n");
}

printf("The Transitive Closure for the given digraph is:\n");
for(i=1;i<=n;i++)
{
        for(j=1;j<=n;j++)
                printf("%d\t",R[i][j]);
        printf("\n");
}
getch();
}
```

14. Floyd's Algorithm

ALGORITHM Floyd(W[1..n,1..n])
//Implements Floyd's algorithm for all pairs-shortest-paths problem
//Input: The weight matrix W of a graph
//Output: The distance matrix of the shortest path's length

D ← W // is not necessary if W can be overwritten
for k ← 1 **to** n **do**
  for i ← 1 **to** n **do**
    for j ← 1 **to** n **do**
      D[i,j] ← min{**D**[i,j],**D**[i,k]+**D**[k,j]}
return D

PROGRAM

```
/*Floyd's algorithm for the All-Pairs Shortest-Paths Problem*/

#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int W[10][10],n,e, D[10][10];

int min(int i,int k)
   {
    if (i<k )return i;
      else return k ;
   }

void Floyd_AllPair()
{
 int i,j,k;
 for(i=1;i<=n;i++)                  // initially R = A
          for(j=1;j<=n;j++)
                   D[i][j]=W[i][j];

 for(k=1;k<=n;k++)
   for(i=1;i<=n;i++)
          for(j=1;j<=n;j++)
               D[i][j]=min(D[i][j], D[i][k]+D[k][j]);
}

int main()
{
//W-given digraph weight matrix;D-will have All Pair Shortest Path
 int v1,v2;
 int i,j,w;
 //clrscr();

 printf("Enter no of vertices\n");
 scanf("%d",&n);

 printf("Enter no of edges\n");
 scanf("%d",&e);

 for(i=1;i<=n;i++)         //initialization of adjacency matrix to infinity
          for(j=1;j<=n;j++)
          {
                   if(i==j)
                            W[i][j]=0;
                   else
                            W[i][j]=999;
           }

 printf("Enter the Edges with distance one by one\n");
```

```
for(i=1;i<=e;i++)
{
        printf("Edge %d: ",i);
        scanf("%d%d%d",&v1,&v2,&w);
        W[v1][v2]=w;
}


Floyd_AllPair();

printf("The given digraph is:\n");
for(i=1;i<=n;i++)
{
        for(j=1;j<=n;j++)
                printf("%d\t",W[i][j]);
        printf("\n");
}

printf("The All Pair Shortest Path for the given digraph is:\n");
for(i=1;i<=n;i++)
{
        for(j=1;j<=n;j++)
                printf("%d\t",D[i][j]);
        printf("\n");
}
getch();
}
```