1.

a. Write a C program to implement Maximum-sub array problem where, you are given a one dimensional array that may contain both positive and negative integers, and find the sum of contiguous sub array of numbers which has the largest sum.

Program

```
#include <stdio.h>

int crosssubsum (int a[],int low,int mid,int high)
{
   int leftsum=0,rightsum=0,sum=0,i;
   for (i=mid;i>=low;i--)
   {
      sum = sum + a[i];
      if (sum > leftsum)
         leftsum = sum;
   }
   sum = 0;
   for (i=mid+1;i<=high;i++)
   {
      sum = sum + a[i];
      if (sum > rightsum)
         rightsum = sum;
   }
   return leftsum+rightsum;
}

int maxsubsum (int a[],int low,int high)
{
   int mid,leftsum,rightsum,crosssum;
   if (low == high)
      return a[low];
   mid = (low+high)/2;
   leftsum = maxsubsum( a,low,mid);
   rightsum = maxsubsum (a,mid+1,high);
   crosssum = crosssubsum (a,low,mid,high);
   if (leftsum >= rightsum && leftsum >=crosssum)
      return leftsum;
   else if (rightsum >= leftsum && rightsum >=crosssum)
      return rightsum;
   else
      return crosssum;
}

int main ()
{
```

```
    int n,a[10],i,low,high,maxsum;
    printf ("Enter the no. of Elements\n");
    scanf ("%d",&n);
    printf ("Enter an Array of +ve and -ve No.\n");
    for (i=0;i<n;i++)
       scanf ("%d",&a[i]);
    low = 0;
    high = n-1;
    maxsum = maxsubsum (a,low,high);
    printf ("Maximum Contiguous Sum = %d\n",maxsum);
    return 0;
}
```

## Algorithm

```
crosssubsum (a,low,mid,high)
//a: array of +ve and -ve elements
//low: position of first element
//mid: position of middle element
//high: position of last element
//o/p: maximum crossing sum

leftsum←0,rightsum←0,sum←0
for i←mid to low
 begin
        sum←sum + a[i]
        if sum > leftsum
          leftsum←sum
 end for
sum←0
for i←mid to high
begin
        sum←sum + a[i]
        if sum > rightsum
          rightsum←sum
 end for
return ledtsum+rightsum

maxsubsum (a,low,high)
//i/p: array of +ve & -ve elements
//o/p: maximim contiguous sum

if  low == high
    return a[low]
mid←(low+high)/2
leftsum←maxsubsum( a,low,mid)
rightsum←maxsubsum (a,mid+1,high)
crosssum←crosssubsum (a,low,mid,high)
if  leftsum >= rightsum && leftsum >=crosssum
    return leftsum
```

```
else if rightsum >= leftsum && rightsum >=crosssum
    return rightsum
else
    return crosssum
```

Output

```
Enter the no. of Elements
6
Enter an Array of +ve & -ve Numbers
13     -3          -25       20        -3         16
Maximum Contiguous Sum = 33
```

b. In a Stock market you can buy any unit of stock and then sell it at same date before the close of trading for the day or later date. If you want to learn what the price of the stock will be in the future and want to maximize your profit, use suitable algorithm design approach and implement the same.

Program

```
#include <stdio.h>

struct stock
{
   int left;
   int right;
   int sum;
};

struct stock crosssubsum (int *a,int low,int mid,int high)
{
   int leftsum=0,rightsum=0,sum=0,i,j,maxleft=0,maxright=0;
   struct stock res;
   for (i=mid;i>=low;i--)
   {
      sum = sum + a[i];
      if (sum > leftsum)
      {
         leftsum = sum;
         maxleft = i;
      }
   }
   sum = 0;
   for (j=mid+1;j<=high;j++)
   {
      sum = sum + a[j];
      if (sum > rightsum)
```

```
        {
            rightsum = sum;
            maxright = j;
        }
    }
    res.left = maxleft;
    res.right = maxright;
    res.sum = leftsum+rightsum;
    return res;
}

struct stock maxsubsum (int *a,int low,int high)
{
    int mid;
    struct stock leftsum,rightsum,crosssum,res;
    if (low == high)
    {
        res.left = low;
        res.right = high;
        res.sum = a[low];
        return res;
    }
    mid = (low+high)/2;
    leftsum = maxsubsum( a,low,mid);
    rightsum = maxsubsum (a,mid+1,high);
    crosssum = crosssubsum (a,low,mid,high);
    if (leftsum.sum >= rightsum.sum && leftsum.sum >=crosssum.sum)
        return leftsum;
    else if (rightsum.sum >= leftsum.sum && rightsum.sum >=crosssum.sum)
        return rightsum;
    else
        return crosssum;
}

int main ()
{
    int n,a[10],i,low,high;
    struct stock maxsum;
    printf ("Enter the no. of Days in Stock Market\n");
    scanf ("%d",&n);
    printf ("Enter the Gain or Loss in Stock Market per Day\n");
    for (i=0;i<n;i++)
        scanf ("%d",&a[i]);
    low =0; high = n-1;
    maxsum = maxsubsum (a,low,high);
    printf ("Maximum Profit in Stock Market lies from Day %d to Day
%d\n",maxsum.left+1,maxsum.right+1);
    printf ("Maximum Profit in Stock Market = %d\n",maxsum.sum);
    return 0;
}
```

## Algorithm

```
crosssubsum (a,low,mid,high)
//a: array of +ve and -ve elements
//low: position of first element
//mid: position of middle element
//high: position of last element
//o/p: maximum crossing sum

leftsum←0,rightsum←0,sum←0
for i←mid to low
 begin
        sum←sum + a[i]
        if sum > leftsum
          leftsum←sum
 end for
sum←0
for i←mid to high
begin
        sum←sum + a[i]
        if sum > rightsum
          rightsum←sum
 end for
return ledtsum+rightsum

maxsubsum (a,low,high)
//i/p: array of +ve & -ve elements
//o/p: maximim contiguous sum

if  low == high
    return a[low]
mid←(low+high)/2
leftsum←maxsubsum( a,low,mid)
rightsum←maxsubsum (a,mid+1,high)
crosssum←crosssubsum (a,low,mid,high)
if  leftsum >= rightsum && leftsum >=crosssum
    return leftsum
else if rightsum >= leftsum && rightsum >=crosssum
    return rightsum
else
    return crosssum
```

## Output

```
Enter the Days in Stock Market
6
Enter the Gain or Loss in Stock Market
13      -3          -25       20        -3         16
Maximum Profit in Stock Market lies from Day 4 to Day 6
Maximum Profit in Stock Market = 33
```

2.

a. i) Using Decrease and Conquer strategy design and execute a program in C, to print all the nodes reachable from a given starting node in a graph using BFS method.

Program

```
#include <stdio.h>

void BFS (int a[10][10],int n,int src,int s[10])
{
   int f=0,r=-1,q[20],i,v;
   printf ("Source Node: %d\n",src);
   s[src] = 1;
   q[++r] = src;
   printf("Visited Nodes using BFS:\n");
   while (f <= r)
   {
      v = q[f++];
      for (i=1;i<=n;i++)
         if (s[i]==0 && a[v][i])
         {
            q[++r] = i;
            printf ("%d\t",i);
            s[i] = 1;
         }
   }
}

int main ()
{
   int a[10][10],n,i,j,s[20],src;
   printf ("Enter no. of Nodes in Graph\n");
   scanf ("%d",&n);
   printf ("Enter the connections of Graph in Adjacency Matrix\n");
   for (i=1;i<=n;i++)
      for (j=1;j<=n;j++)
         scanf ("%d",&a[i][j]);
   printf ("Enter the Source Node\n");
   scanf ("%d",&src);
   for(i=1;i<=n;i++)
      s[i]=0;
   BFS (a,n,src,s);
   printf ("\n");
   return 0;
}
```

## Algorithm

```
BFS (a,n,src,s)
//a: adjacency matrix of given graph
//n: no. of nodes in given graph
//src: source node of given graph
//s: array of connected nodes
//o/p: breadth first search of given graph

f←0,r←-1
print src
s[src]←1
q[++r]←src
while f <= r
    v←q[f++];
    for i←1 to n
      if s[i]==0 && a[v][i]
        q[++r]←i
        print i
        s[i]←1
      end if
    end for
end while
```

## Output

```
Enter no. of Nodes in Graph
8
Enter the connections of Graph in Adjacency Matrix
0 1 0 1 0 0 1 0
1 0 0 0 1 1 0 0
0 0 0 0 0 1 0 1
1 0 0 0 0 1 0 0
0 1 0 0 0 0 1 0
0 1 1 1 0 0 0 0
1 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
Enter the Source Node
1
Source Node = 1
Visited Nodes using BFS:
2       4       7       5       6       3       8
```

ii) Using Decrease and Conquer strategy develop a program in C to check whether a given graph is connected or not using DFS method.

## Program

```
#include <stdio.h>

void DFS (int a[10][10],int n,int src,int s[10])
{
   int i;
   s[src] = 1;
   printf ("%d\t",src);
   for (i=1;i<=n;i++)
      if (s[i]==0 && a[src][i])
         DFS (a,n,i,s);
}

int main ()
{
   int a[10][10],n,i,j,s[20],src;
   printf ("Enter no. of Nodes in Graph\n");
   scanf ("%d",&n);
   printf ("Enter the connections of Graph in Adjacency Matrix\n");
   for (i=1;i<=n;i++)
      for (j=1;j<=n;j++)
         scanf ("%d",&a[i][j]);
   printf ("Enter the Source Node\n");
   scanf ("%d",&src);
   printf ("Source Node = %d\n",src);
   for (i=1;i<=n;i++)
      s[i]=0;
   printf("Visited Nodes using DFS:\n");
   DFS (a,n,src,s);
   printf ("\n");
   return 0;
}
```

## Algorithm

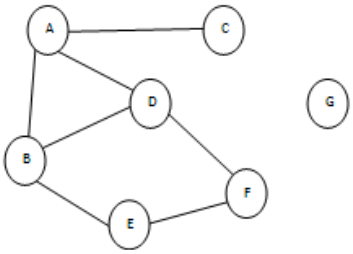
```
DFS (a,n,src,s)
//a: adjacency matrix of given graph
//n: no. of nodes in given graph
//src: source node of given graph
//s: array of connected nodes
//o/p: depth first search of given graph

s[src]←1
print src
for i←1 to n
```

```
  if s[i]==0 && a[src][i]
    DFS (a,n,i,s)
```

Output

```
Enter no. of Nodes in Graph
5
Enter the connections of Graph in Adjacency Matrix
0 1 0 1 0
1 0 1 0 1
0 1 0 0 0
1 0 0 0 1
0 1 0 1 0
Enter the Source Node
2
Source Node = 2
Visited Nodes using DFS:
    2       1       4       5       3
```

b. Navigation systems such as the Google Maps, which can give directions to reach from one place to another, take your location to be the source node and your destination as the destination node on the graph. Graph traversal methods are used to find if the destination entered can be reachable or not.
   Hint: Any city can be represented as a graph taking landmarks as nodes and roads as edges. Using an appropriate technique check if your desired destination is reachable or not considering the following graph.




Program

```
#include <stdio.h>

void BFS (int a[10][10],int n,int src,int s[10])
{
   int f=0,r=-1,q[20],i,v;
   s[src] = 1;
   q[++r] = src;
   while (f <= r)
   {
      v = q[f++];
```

```
        for (i=1;i<=n;i++)
            if (s[i]==0 && a[v][i])
            {
                q[++r] = i;
                s[i] = 1;
            }
    }
}

int main ()
{
    int a[10][10],n,i,j,s[20],src,dest;
    printf ("Enter no. of Nodes in Graph\n");
    scanf ("%d",&n);
    printf ("Enter the connections of Graph in Adjacency Matrix\n");
    for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
            scanf ("%d",&a[i][j]);
    printf ("Enter the Source Node\n");
    scanf ("%d",&src);
    printf ("Enter the Destination Node\n");
    scanf ("%d",&dest);
    printf ("Source Node = %d\n",src);
    printf ("Destination Node = %d\n",dest);
    for(i=1;i<=n;i++)
        s[i]=0;
    BFS (a,n,src,s);
    if (s[src]==1 && s[dest]==1)
        printf ("Destination is Reachable from %d to %d\n",src,dest);
    else
        printf ("Destination not Reachable\n");
    return 0;
}
```

(OR)

```
#include <stdio.h>

void DFS (int a[10][10],int n,int src,int s[10])
{
    int i;
    s[src] = 1;
    for (i=1;i<=n;i++)
        if (s[i]==0 && a[src][i])
            DFS (a,n,i,s);
}

int main ()
{
    int a[10][10],n,i,j,s[20],src,dest;
```

```
    printf ("Enter no. of Nodes in Graph\n");
    scanf ("%d",&n);
    printf ("Enter the connections of Graph in Adjacency Matrix\n");
    for (i=1;i<=n;i++)
       for (j=1;j<=n;j++)
          scanf ("%d",&a[i][j]);
    printf ("Enter the Source Node\n");
    scanf ("%d",&src);
    printf ("Enter the Destination Node\n");
    scanf ("%d",&dest);
    printf ("Source Node = %d\n",src);
    printf ("Destination Node = %d\n",dest);
    for(i=1;i<=n;i++)
       s[i]=0;
    DFS (a,n,src,s);
    if (s[src]==1 && s[dest]==1)
       printf ("Destination is Reachable from %d to %d\n",src,dest);
    else
       printf ("Destination not Reachable\n");
    return 0;
}
```

## Algorithm

```
BFS (a,n,src,s)
//a: adjacency matrix of given graph
//n: no. of nodes in given graph
//src: source node of given graph
//s: array of connected nodes
//o/p: breadth first search of given graph

f←0,r←-1
s[src]←1
q[++r]←src
while f <= r
    v←q[f++];
    for i←1 to n
      if s[i]==0 && a[v][i]
        q[++r]←i
        s[i]←1
      end if
    end for
end while
```

(OR)

```
DFS (a,n,src,s)
//a: adjacency matrix of given graph
//n: no. of nodes in given graph
```

```
//src: source node of given graph
//s: array of connected nodes
//o/p: depth first search of given graph

s[src]←1
for i←1 to n
  if s[i]==0 && a[src][i]
    DFS (a,n,i,s)
```

## Output

1. Enter no. of Nodes in Graph
   7
   Enter the connections of Graph in Adjacency Matrix
   0 1 1 1 0 0 0
   1 0 0 1 1 0 0
   1 0 0 0 0 0 0
   1 1 0 0 0 1 0
   0 1 0 0 0 1 0
   0 0 0 1 1 0 0
   0 0 0 0 0 0 0
   Enter the Source Node
   1
   Enter the Destination Node
   6
   Source Node = 1
   Destination Node = 6
   Destination Reachable form 1 to 6

2. Enter no. of Nodes in Graph
   7
   Enter the connections of Graph in Adjacency Matrix
   0 1 1 1 0 0 0
   1 0 0 1 1 0 0
   1 0 0 0 0 0 0
   1 1 0 0 0 1 0
   0 1 0 0 0 1 0
   0 0 0 1 1 0 0
   0 0 0 0 0 0 0
   Enter the Source Node
   1
   Enter the Destination Node
   7
   Source Node = 1
   Destination Node = 7
   Destination not Reachable

3.

a. Design and execute a program in C to search for the pattern string in given text string using Boyer-Moore String Matching algorithm.

Program

```
#include <stdio.h>
#include <string.h>
#define MAX 126

int t[MAX];

int max (int a, int b)
{
   return (a > b)? a: b;
}

void shifttable (char p[])
{
   int i,j,m;
   m = strlen(p);
   for (i=0;i<MAX;i++)
      t[i] = m;
   for (j=0;j<m-1;j++)
      t[p[j]] = m-1-j;
}

int boyermoore (char src[],char p[])
{
   int i,k,m,n;
   n = strlen(src);
   m = strlen(p);
   printf ("Length of Text = %d\n",n);
   printf ("Length of Pattern = %d\n",m);
   i = m-1;
   while (i<n)
   {
      k = 0;
      while ((k<m) && (p[m-1-k] == src[i-k]))
        k++;
      if (k == m)
        return (i-m+1);
      else
         i = i + max(t[src[i]]-k,1);
   }
   return -1;
}
```

```
int main ()
{
   char src[100],p[100];
   int pos;
   printf ("Enter the Text in which Pattern is to be Searched\n");
   gets (src);
   printf ("Enter the Pattern to be Searched\n");
   gets (p);
   shifttable (p);
   pos = boyermoore (src,p);
   if (pos >= 0)
     printf ("Pattern was found starting from position %d\n",pos+1);
   else
     printf ("Pattern was not found\n");
   return 0;
}
```

## Algorithm

```
shifttable (p)
//p: Pattern String
//o/p: Shift Table of Pattern String

m ← strlen(p)
for i←0 to MAX
   t[i] ← m
for i←0 to m-1
   t[p[j]] ← m-1-j

boyermoore (src,p)
//src: Source String
//p: Pattern String
//o/p: Starting Position of Pattern String in Source String

n ← strlen(src)
m ← strlen(p)
print Length of Text
print Length of Pattern
i ← m-1
while i<n
begin
   k ← 0
   while (k<m) && (p[m-1-k] == src[i-k])
     k++
   if k == m
     return i-m+1
   else
      i ← i + max(t[src[i]]-k,1)
end while
return -1
```

Output

```
Enter the Text in which Pattern is to be Searched
JIM SAW ME IN BARBER SHOP
Enter the Pattern to be Searched
BARBER
Length of Text = 25
Length of Pattern = 6
Pattern was found starting from position 15
```

b. Spam (Unsolicited and unwanted emails) is annoying, no doubt, but it can also be dangerous. Malware and phishing are hugely profitable for scammers and can be costly for mailbox providers' customers, as well as the mailbox providers who face intense market competition. To avoid such problem mailbox providers' use Spam filters. All spam filters use the concept of string matching to identify and discard the spam.
   Considering the above scenario use an appropriate technique to classify the mails if it's a spam or not when there are patterns like "free," "money," "help" and "prize" in the mail content.

Program

```
#include <stdio.h>
#include <string.h>
#define MAX 126

int t[MAX];

int max (int a, int b)
{
   return (a > b)? a: b;
}

void shifttable (char p[])
{
   int i,j,m;
   m = strlen(p);
   for (i=0;i<MAX;i++)
      t[i] = m;
   for (j=0;j<m-1;j++)
      t[p[j]] = m-1-j;
}

int boyermoore (char src[],char p[])
{
```

```
    int i,k,m,n;
    n = strlen(src);
    m = strlen(p);
    i = m-1;
    while (i<n)
    {
       k = 0;
       while ((k<m) && (p[m-1-k] == src[i-k]))
          k++;
       if (k == m)
          return (i-m+1);
       else
           i = i + max(t[src[i]]-k,1);
    }
    return -1;
}

int main ()
{
    char src[100],p[100];
    int i,pos,count=0;
    char keywords [4][10] = {"free","money","help","price"};
    printf ("Enter Mail Message\n");
    gets (src);
    for (i=0;i<4;i++)
    {
       strcpy (p,keywords[i]);
       shifttable (p);
       pos = boyermoore (src,p);
       if (pos >= 0)
          count++;
    }
    if (count > 3)
       printf ("SPAM!\n");
    else
       printf ("NOT SPAM\n");
    return 0;
}
```

## Algorithm

```
shifttable (p)
//p: Pattern String
//o/p: Shift Table of Pattern String

m ← strlen(p)
for i←0 to MAX
   t[i] ← m
for i←0 to m-1
   t[p[j]] ← m-1-j
```

```
boyermoore (src,p)
//src: Source String
//p: Pattern String
//o/p: Starting Position of Pattern String in Source String

n ← strlen(src)
m ← strlen(p)
i ← m-1
while i<n
begin
   k ← 0
   while (k<m) && (p[m-1-k] == src[i-k])
     k++
   if k == m
     return i-m+1
   else
      i ← i + max(t[src[i]]-k,1)
end while
return -1
```

## Output

1. Enter Mail Message
   Hey User! You have won free gifts and prices! If help and money required do contact us.
   SPAM!

2. Enter Mail Message
   Hello World!
   NOT SPAM

4.

a. Given two sequences X = <x1; x2; : : : ; xm>, Y = <y1; y2; : : : ; yn>and required to find a longest-common-subsequence , of X and Y using dynamic programming.

Program

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

int MAX (int a,int b)
{
   return (a>b)?a:b;
}

void LCS (char X[],char Y[],int m,int n)
{
   int i,j,L[m+1][n+1];
   for (i=0;i<=m;i++)
   {
      for (j=0;j<=n;j++)
      {
         if (i == 0 || j == 0)
            L[i][j] = 0;
         else if (X[i-1] == Y[j-1])
            L[i][j] = L[i-1][j-1] + 1;
         else
            L[i][j] = MAX(L[i-1][j], L[i][j-1]);
      }
   }
   printf ("The Length of Longest Common Subsequence is %d\n",L[m][n]);
   int index = L[m][n];
   char lcs[index+1];
   lcs[index] = '\0';
   i = m;
   j = n;
   while (i > 0 && j > 0)
   {
    if (X[i-1] == Y[j-1])
    {
       lcs[index-1] = X[i-1];
       i--;
       j--;
       index--;
    }
    else if (L[i-1][j] > L[i][j-1])
      i--;
    else
```

```
        j--;
    }
    printf ("The Longest Common Subsequence of %s and %s is %s\n",X,Y,lcs);
}

int main ()
{
    char X[20],Y[20];
    int m,n;
    printf ("Enter 1st Sequence\n");
    gets (X);
    printf ("Enter 2nd Sequence\n");
    gets (Y);
    m = strlen (X);
    n = strlen (Y);
    LCS (X,Y,m,n);
    return 0;
}
```

## Algorithm

```
LCS (X,Y,m,n)
//X: 1st Sequence
//Y: 2nd Sequence
//m: Length of 1st Sequence
//n: Length of 2nd Sequence
//o/p: Longest Common Subsequence and it's length

int L[m+1][n+1]
for i←0 to m
    for j←0 to n
    begin
          if i == 0 || j == 0
             L[i][j] ← 0
          else if X[i-1] == Y[j-1
             L[i][j] ← L[i-1][j-1] + 1
          else
             L[i][j] ← MAX(L[i-1][j], L[i][j-1])
    end for
end for
print L[m][n]
int index ← L[m][n]
char lcs[index+1]
lcs[index] ← '\0'
i ← m
j ← n
while i > 0 && j > 0
begin
  if X[i-1] == Y[j-1]
  begin
```

```
    lcs[index-1] ← X[i-1]
    i--
    j--
    index--
  end if
  else if L[i-1][j] > L[i][j-1]
    i--;
  else
    j--;
end while
print lcs
```

Output

```
Enter 1st Sequence
PRESIDENT
Enter 2nd Sequence
PROVIDENCE
The Length of Longest Common Subsequence is 6
The Longest Common Subsequence of PRESIDENT and PROVIDENCE is PRIDEN
```

b. DNA-based identity testing is extensively used in the forensic field. DNA sequences can be viewed as strings of A, C, G, and T characters, which represent nucleotides. To compare and analyze two such strings, the longest subsequence is necessary. With an appropriate approach print the longest subsequence for two DNA sequences.

Program

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

int MAX (int a,int b)
{
   return (a>b)?a:b;
}

void LCS (char X[],char Y[],int m,int n)
{
   int i,j,L[m+1][n+1];
   for (i=0;i<=m;i++)
   {
      for (j=0;j<=n;j++)
      {
         if (i == 0 || j == 0)
            L[i][j] = 0;
```

```
            else if (X[i-1] == Y[j-1])
                L[i][j] = L[i-1][j-1] + 1;
            else
                L[i][j] = MAX(L[i-1][j], L[i][j-1]);
        }
    }
    printf ("The Length of Longest Common DNA Subsequence is %d\n",L[m][n]);
    int index = L[m][n];
    char lcs[index+1];
    lcs[index] = '\0';
    i = m;
    j = n;
    while (i > 0 && j > 0)
    {
      if (X[i-1] == Y[j-1])
      {
          lcs[index-1] = X[i-1];
          i--;
          j--;
          index--;
      }
      else if (L[i-1][j] > L[i][j-1])
         i--;
      else
         j--;
    }
    printf ("The Longest Common DNA Subsequence of %s and %s is %s\n",X,Y,lcs);
}

int main ()
{
    char X[20],Y[20];
    int m,n;
    //DNA Sequence contain string of A,C,G and T characters only
    printf ("Enter 1st DNA Sequence\n");
    gets (X);
    printf ("Enter 2nd DNA Sequence\n");
    gets (Y);
    m = strlen (X);
    n = strlen (Y);
    LCS (X,Y,m,n);
    return 0;
}
```

## Algorithm

LCS (X,Y,m,n)
//X: 1st Sequence
//Y: 2nd Sequence
//m: Length of 1st Sequence

```
//n: Length of 2nd Sequence
//o/p: Longest Common Subsequence and it's length

int L[m+1][n+1]
for i←0 to m
   for j←0 to n
   begin
         if i == 0 || j == 0
            L[i][j] ← 0
         else if X[i-1] == Y[j-1]
            L[i][j] ← L[i-1][j-1] + 1
         else
            L[i][j] ← MAX(L[i-1][j], L[i][j-1])
   end for
end for
print L[m][n]
int index ← L[m][n]
char lcs[index+1]
lcs[index] ← '\0'
i ← m
j ← n
while i > 0 && j > 0
begin
  if X[i-1] == Y[j-1]
  begin
     lcs[index-1] ← X[i-1]
     i--
     j--
     index--
  end if
  else if L[i-1][j] > L[i][j-1]
    i--;
  else
    j--;
end while
print lcs
```

## Output

```
Enter 1st DNA Sequence
AATTCCGT
Enter 2nd DNA Sequence
AACCGTCG
The Length of Longest Common DNA Subsequence is 6
The Longest Common DNA Subsequence of AATTCCGT and AACCGTCG is AACCGT
```

5.

a. Implement Kruskal algorithm in a function KRUSKAL. Execute this function, giving cost adjacency matrix of a undirected graph as input and output its Minimum Spanning Tree.

Program

```
#include <stdio.h>

void kruskal (int cost[20][20],int n)
{
   int parent[20]={0},min,mincost=0,ne=1;
   int a,b,i,j,u,v;
   while (ne < n)
   {
      for (i=1,min=999;i<=n;i++)
         for (j=1;j<=n;j++)
            if (cost[i][j] < min)
            {
               min = cost[i][j];
               a = u = i;
               b = v = j;
            }
      while (parent[u])
         u = parent[u];
      while (parent[v])
         v = parent[v];
      if (u != v)
      {
         printf ("%d edge (%d,%d) = %d\n",ne++,a,b,min);
         mincost = mincost + min;
         parent[v] = u;
      }
      cost[a][b] = cost[b][a] = 999;
   }
   printf ("The Minimum Cost = %d\n",mincost);
}

int main()
{
   int n,i,j,cost[20][20];
   printf ("Enter the no. of Nodes\n");
   scanf ("%d",&n);
   printf ("Enter the Cost Adjacency Matrix\n");
   for (i=1;i<=n;i++)
      for (j=1;j<=n;j++)
      {
         scanf ("%d",&cost[i][j]);
```

```
            if (cost[i][j] == 0)
               cost[i][j] = 999;
         }
      printf ("The edges of Minimum Cost Spanning Tree are\n");
      kruskal (cost,n);
      return 0;
}
```

## Algorithm

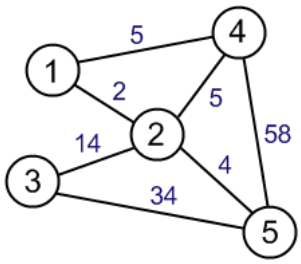
```
kruskal (cost,n)
//cost: Cost Adjacency Matrix
//n: No. of Nodes
//o/p: Minimum Spanning Tree and Minimum Cost

int parent[20]←{0},min,mincost←0,ne←1
int a,b,i,j,u,v
while ne < n
begin
   for i←1 to n,min←999
      for j←1 to n
         if cost[i][j] < min
         begin
            min ← cost[i][j]
            a ← u ← i
            b ← v ← j
         end if
   while parent[u]
      u ← parent[u]
   while parent[v]
      v ← parent[v]
   if u != v
   begin
      print Minimum Spanning Tree
      mincost ← mincost + min
      parent[v] ← u
   end if
   cost[a][b] ← cost[b][a] ← 999
end while
print Minimum Cost
```

## Output

```
Enter the no. of Nodes
5
Enter the Cost Adjacency Matrix
999 2 999 6 999
2 999 3 8 5
999 3 999 999 7
```

```
6 8 999 999 9
999 5 7 9 999
The edges of Minimum Cost Spanning Tree are
1 edge (1,2) = 2
2 edge (2,3) = 3
3 edge (2,5) = 5
4 edge (1,4) = 6
The Minimum Cost = 16
```

b. Laying out electrical networks to be carried in a city that minimizes the total cost of the wiring. Choose appropriate design strategy to connect all the houses so that wiring cost is minimum.
Given : Layout input as a graph with the distances. Nodes 1,2,3,4 and 5 represent homes and the edges with cost gives the wiring cost between the homes.




Program

```
#include <stdio.h>

void kruskal (int cost[20][20],int n)
{
   int parent[20]={0},min,mincost=0,ne=1;
   int a,b,i,j,u,v;
   while (ne < n)
   {
      for (i=1,min=999;i<=n;i++)
         for (j=1;j<=n;j++)
            if (cost[i][j] < min)
            {
               min = cost[i][j];
               a = u = i;
               b = v = j;
            }
      while (parent[u])
         u = parent[u];
      while (parent[v])
         v = parent[v];
      if (u != v)
      {
         printf ("%d edge (%d,%d) = %d\n",ne++,a,b,min);
```

```
            mincost = mincost + min;
            parent[v] = u;
        }
        cost[a][b] = cost[b][a] = 999;
    }
    printf ("The Minimum Wiring Cost = %d\n",mincost);
}

int main()
{
    int n,i,j,cost[20][20];
    printf ("Enter the no. of Nodes\n");
    scanf ("%d",&n);
    printf ("Enter the Cost Adjacency Matrix\n");
    for (i=1;i<=n;i++)
        for (j=1;j<=n;j++)
        {
            scanf ("%d",&cost[i][j]);
            if (cost[i][j] == 0)
                cost[i][j] = 999;
        }
    printf ("The edges of Minimum Cost Spanning Tree are\n");
    kruskal (cost,n);
    return 0;
}
```

## Algorithm

```
kruskal (cost,n)
//cost: Cost Adjacency Matrix
//n: No. of Nodes
//o/p: Minimum Spanning Tree and Minimum Cost

int parent[20]←{0},min,mincost←0,ne←1
int a,b,i,j,u,v
while ne < n
begin
    for i←1 to n,min←999
        for j←1 to n
            if cost[i][j] < min
            begin
                min ← cost[i][j]
                a ← u ← i
                b ← v ← j
            end if
    while parent[u]
        u ← parent[u]
    while parent[v]
        v ← parent[v]
    if u != v
    begin
```

```
        print Minimum Spanning Tree
        mincost ← mincost + min
        parent[v] ← u
    end if
    cost[a][b] ← cost[b][a] ← 999
end while
print Minimum Wiring Cost
```

## Output

```
Enter the no. of Nodes
5
Enter the Cost Adjacency Matrix
999 2 999 5 999
2 999 14 5 4
999 14 999 999 34
5 5 999 999 58
999 4 34 58 999
The edges of Minimum Cost Spanning Tree are
1 edge (1,2) = 2
2 edge (2,5) = 4
3 edge (1,4) = 5
4 edge (2,3) = 14
The Minimum Wiring Cost = 25
```

6.

a. Design and execute a program in C to create a function called bellman-ford that represents the cost adjacency matrix. From a given vertex in a weighted connected graph, find shortest paths from a single source vertex to all of the other vertices using Bellman-Ford algorithm.

Program

```
#include <stdio.h>

int BellmanFord (int G[20][20] ,int n,int E,int edge[20][2])
{
   int i,u,v,k,distance[20],parent[20],S,flag=1;
   for (i=0;i<n;i++)
   {
      distance[i] = 1000;
      parent[i] = -1;
   }
   printf ("Enter Source\n");
   scanf ("%d",&S);
   printf ("Shortest Path from Source %d:\n",S);
   distance[S-1] = 0;
   for (i=0;i<n-1;i++)
   {
      for (k=0;k<E;k++)
      {
         u = edge[k][0];
         v = edge[k][1];
         if (distance[u]+G[u][v] < distance[v])
         {
            distance[v] = distance[u] + G[u][v];
            parent[v] = u;
         }
      }
   }
   for(k=0;k<E;k++)
   {
      u = edge[k][0];
      v = edge[k][1];
      if (distance[u]+G[u][v] < distance[v])
         flag = 0;
   }
   if (flag)
      for (i=0;i<n;i++)
         printf ("Vertex %d -> Cost = %d Parent = %d\n",i+1,distance[i],parent[i]+1);
   return flag;
```

```
}

int main ()
{
   int n,edge[20][2],G[20][20],i,j,k=0;
   printf ("Enter no. of Nodes in Graph\n");
   scanf ("%d",&n);
   printf ("Enter Cost Adjacency Matrix of Graph\n");
   for (i=0;i<n;i++)
      for (j=0;j<n;j++)
      {
         scanf ("%d",&G[i][j]);
         if (G[i][j] != 0)
         {
            edge[k][0] = i;
            edge[k++][1] = j;
         }
      }
   if (BellmanFord(G,n,k,edge))
      printf ("No negative weight cycle\n");
   else
      printf ("Negative weight cycle exists\n");
   return 0;
}
```

## Algorithm

```
BellmanFord (G,n,E,edge)
//G: Cost Adjacency Matrix
//n: No. of Nodes
//E: No, of Edges
//edge: Edge of each nodes
//o/p: Shortest Path from Source

int i,u,v,k,distance[20],parent[20],S,flag←1
for i←0 to n
begin
      distance[i] ← 1000
      parent[i] ← -1
end for
print source
scan source
print Shortest Path from Source
distance[S-1] ← 0
for i←0 to n-1
begin
   for k←0 to E
   begin
```

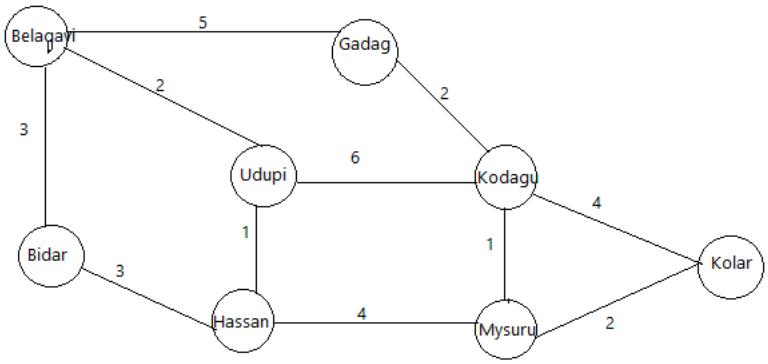
```
        u ← edge[k][0]
        v ← edge[k][1]
        if distance[u]+G[u][v] < distance[v]
        begin
            distance[v] ← distance[u] + G[u][v]
            parent[v] ← u
         end if
      end for
end for
for k←0 to E
begin
   u ← edge[k][0]
   v ← edge[k][1]
   if distance[u]+G[u][v] < distance[v]
      flag ← 0
end for
if flag
   for i←0 to n
      print Shortest Path of each vertex from a source
return flag
```

Output

```
Enter no. of Nodes in Graph
5
Enter Cost Adjacency Matrix of Graph
0 6 0 7 0
0 0 5 8 -4
0 -2 0 0 0
0 0 -3 0 9
2 0 7 0 0
Enter Source
1
Shortest Path from Source 1:
Vertex 1 -> Cost = 0 Parent = 0
Vertex 2 -> Cost = 2 Parent = 3
Vertex 3 -> Cost = 4 Parent = 4
Vertex 4 -> Cost = 7 Parent = 1
Vertex 5 -> Cost = -2 Parent = 2
No negative weight cycle
```

b. In the google map, the navigation system always shows the shortest path to travel from source to destination. State can be represented as a weighted graph by taking districts as nodes and roads as the edges that connects the districts.

Considering the above scenario, apply single source shortest path method to find the shortest distance between two districts for the graph.




Program

```
#include <stdio.h>

int BellmanFord (int G[20][20] ,int n,int E,int edge[20][2])
{
   int i,u,v,k,distance[20],parent[20],S,D,flag=1;
   for (i=0;i<n;i++)
   {
      distance[i] = 1000;
      parent[i] = -1;
   }
   printf ("Enter Source\n");
   scanf ("%d",&S);
   printf ("Enter Destination\n");
   scanf ("%d",&D);
   printf ("Shortest Path from Source %d to Destination %d:\n",S,D);
   distance[S-1] = 0;
   for (i=0;i<n-1;i++)
   {
      for (k=0;k<E;k++)
      {
         u = edge[k][0];
         v = edge[k][1];
         if (distance[u]+G[u][v] < distance[v])
         {
            distance[v] = distance[u] + G[u][v];
            parent[v] = u;
         }
      }
   }
   for(k=0;k<E;k++)
   {
      u = edge[k][0];
      v = edge[k][1];
```

```
        if (distance[u]+G[u][v] < distance[v])
            flag = 0;
    }
    if (flag)
        printf("Vertex %d -> Cost = %d Parent = %d\n",D,distance[D-1],parent[D-1]+1);
    return flag;
}

int main ()
{
    int n,edge[20][2],G[20][20],i,j,k=0;
    printf ("Enter no. of Nodes in Graph\n");
    scanf ("%d",&n);
    printf ("Enter Cost Adjacency Matrix of Graph\n");
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
        {
            scanf ("%d",&G[i][j]);
            if (G[i][j] != 0)
            {
                edge[k][0] = i;
                edge[k++][1] = j;
            }
        }
    if (BellmanFord(G,n,k,edge))
        printf ("No negative weight cycle\n");
    else
        printf ("Negative weight cycle exists\n");
    return 0;
}
```

## Algorithm

```
BellmanFord (G,n,E,edge)
//G: Cost Adjacency Matrix
//n: No. of Nodes
//E: No, of Edges
//edge: Edge of each nodes
//o/p: Shortest Path from Source to Destination

int i,u,v,k,distance[20],parent[20],S,D,flag←1
for i←0 to n
begin
        distance[i] ← 1000
        parent[i] ← -1
end for
print source
scan source
print destination
scan destination
```

```
print Shortest Path from Source to Destination
distance[S-1] ← 0
for i←0 to n-1
begin
   for k←0 to E
   begin
        u ← edge[k][0]
        v ← edge[k][1]
        if distance[u]+G[u][v] < distance[v]
        begin
             distance[v] ← distance[u] + G[u][v]
             parent[v] ← u
         end if
     end for
end for
for k←0 to E
begin
   u ← edge[k][0]
   v ← edge[k][1]
   if distance[u]+G[u][v] < distance[v]
      flag ← 0
end for
if flag
   for i←0 to n
      print Shortest Path from Source to Destination
return flag
```

## Output

```
Enter no. of Nodes in Graph
8
Enter Cost Adjacency Matrix of Graph
999 3 5 2 999 999 999 999
3 999 999 999 3 999 999 999
5 999 999 999 999 2 999 999
2 999 999 999 1 6 999 999
999 3 999 1 999 999 4 999
999 999 2 6 999 999 1 4
999 999 999 999 4 1 999 2
999 999 999 999 999 4 2 999
Enter Source
3
Enter Destination
8
Shortest Path from Source 3 to Destination 8:
Vertex 8 -> Cost = 5 Parent = 7
No negative weight cycle
```

Belagavi→1
Bidar→2
Gadag→3
Udupi→4
Hassan→5
Kodagu→6
Mysuru→7
Kolar→8

7.

a. Design and execute a program in C to create a function called SUMOFSUBSET that represents the array of elements, and to find a subset of a given set S = {sl, s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S={1, 2, 5, 6, 8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.

Program

```
#include <stdio.h>
int a[100],x[100];
int d;

void sum_of_subset (int weightsofar,int k,int remwt)
{
   int i;
   x[k] = 1;
   if (weightsofar + a[k] == d)
   {
      printf ("Subset ===>> {");
      for (i=1;i<=k;i++)
         if (x[i] == 1)
            printf ("%d,",a[i]);
      printf ("}\n");
   }
   else if (weightsofar + a[k] + a[k+1] <= d)
      sum_of_subset (weightsofar+a[k],k+1,remwt-a[k]);
   if ((weightsofar+remwt-a[k] >= d) && (weightsofar+a[k+1] <= d))
   {
      x[k] = 0;
      sum_of_subset (weightsofar,k+1,remwt-a[k]);
   }
}

int main ()
{
   int n,i,sum=0;
   printf ("Enter no. of Elements\n");
   scanf ("%d",&n);
   printf ("Enter the Elements\n");
   for (i=1;i<=n;i++)
   {
      scanf ("%d",&a[i]);
      sum = sum + a[i];
   }
```

```
   printf ("Enter the Required Sum to be Computed\n");
   scanf ("%d",&d);
   if (sum < d)
      printf ("No Solution Exists\n");
   else
   {
      printf("The Solution is\n");
      sum_of_subset (0,1,sum);
   }
   return 0;
}
```

## Algorithm

```
sum_of_subset (weightsofar,k,remwt)
//weightsofar: Weight of Array
//k: Element in Array
//remwt: Remaining Weight in Array
//o/p: Subset leading to required Sum

int i
x[k] ← 1
if weightsofar + a[k] == d
begin
   print subset leading to required sum
end if
else if weightsofar + a[k] + a[k+1] <= d
   sum_of_subset (weightsofar+a[k],k+1,remwt-a[k])
if (weightsofar+remwt-a[k] >= d) && (weightsofar+a[k+1] <= d)
begin
   x[k] ← 0
   sum_of_subset (weightsofar,k+1,remwt-a[k])
end if
```

## Output

```
Enter no. of Elements
5
Enter the Elements
1 2 5 6 8
Enter the Required Sum to be Computed
9
The Solution is
Subset ===>> {1,2,6,}
Subset ===>> {1,8,}
```

b. In an Education institution, there are faculties with varied number of teaching experience. For an upcoming semester Computer Science department requires a team of faculty with combined experience of 9 years to form a syllabus for Data Science course.
Consider the table below with information about the teaching experience of Computer Science faculty. Apply an appropriate backtracking technique to solve the above scenario.

| SL.No | Faculty | Teaching Experience |
|---|---|---|
| 1) | Mr. John | 1 yr |
| 2) | Mr. Jacob | 2 yrs |
| 3) | Mr. Dave | 3 yrs |
| 4) | Mrs. Emily | 6 yrs |
| 5) | Mrs. Ava | 7 yrs |
| 6) | Ms. Jessica | 8 yrs |

Program

```
#include <stdio.h>

struct faculty
{
   char name[20];
   int exp;
};

int x[100];
int d=9;
struct faculty a[7]={{"_________",},{"Mr. John",1},{"Mr. Jacob",2},{"Mr. Dave",3},
   {"Mrs. Emily",6},{"Mrs. Ava",7},{"Ms. Jessica",8}};

void sum_of_subset (int weightsofar,int k,int remwt)
{
   int i=0;
   x[k] = 1;
   if (weightsofar + a[k].exp == d)
   {
      printf ("\nTeam of:\n{\n");
      for (i=0;i<k;i++)
      {
         if (x[i+1] == 1)
            printf ("\t%s -> %dyrs of Experience\n",a[i+1].name,a[i+1].exp);
      }
      printf ("}\n");
   }
```

```
   else if (weightsofar+a[k].exp+a[k+1].exp <= d)
      sum_of_subset (weightsofar+a[k].exp,k+1,remwt-a[k].exp);
   if ((weightsofar+remwt-a[k].exp >= d) && (weightsofar+a[k+1].exp <= d))
   {
      x[k] = 0;
      sum_of_subset (weightsofar,k+1,remwt-a[k].exp);
   }
}

int main ()
{
   int i,sum=0;
   printf ("Sl.No.\t\tFaculty\t\t\tTeaching Experience\n");
   for (i=1;i<=6;i++)
   {
      printf ("%d\t\t\t%s\t\t%dyrs\n",i,a[i].name,a[i].exp);
      sum = sum + a[i].exp;
   }
   printf ("\nRequired Years of Experience: %dyrs\n",d);
   printf ("\nTeams with %d Years of Experience:\n",d);
   if (sum < d)
        printf ("No Team Exists\n");
   else
      sum_of_subset (0,1,sum);
return 0;
}
```

## Algorithm

```
sum_of_subset (weightsofar,k,remwt)
//weightsofar: Weight of Array
//k: Element in Array
//remwt: Remaining Weight in Array
//o/p: Subset leading to required Sum

int i
x[k] ← 1
if weightsofar + a[k] == d
begin
   print subset leading to required sum
end if
else if weightsofar + a[k] + a[k+1] <= d
   sum_of_subset (weightsofar+a[k],k+1,remwt-a[k])
if (weightsofar+remwt-a[k] >= d) && (weightsofar+a[k+1] <= d)

begin
   x[k] ← 0
   sum_of_subset (weightsofar,k+1,remwt-a[k])
end if
```

## Output

| Sl.No. | Faculty | Teaching Experience |
|---|---|---|
| 1 | Mr. John | 1yrs |
| 2 | Mr. Jacob | 2yrs |
| 3 | Mr. Dave | 3yrs |
| 4 | Mrs. Emily | 6yrs |
| 5 | Mrs. Ava | 7yrs |
| 6 | Ms. Jessica | 8yrs |

Required Years of Experience: 9yrs

Teams with 9 Years of Experience:

Team of:
{
    Mr. John -> 1yrs of Experience
    Mr. Jacob -> 2yrs of Experience
    Mrs. Emily -> 6yrs of Experience
}

Team of:
{
    Mr. John -> 1yrs of Experience
    Ms. Jessica -> 8yrs of Experience
}

Team of:
{
    Mr. Jacob -> 2yrs of Experience
    Mrs. Ava -> 7yrs of Experience
}

Team of:
{
    Mr. Dave -> 3yrs of Experience
    Mrs. Emily -> 6yrs of Experience
}