ANALYSIS AND DESIGN OF ALGORITHMS

(3150703)

LABORATORY MANUAL

B.E. Semester-V

Computer Engineering, Information Technology

PREPARED BY:
CE/IT DEPARTMENT- 2017-2018
VADODARA INSTITUTE OF ENGINEERING, KOTAMBI-391510

List ofEquipments and components for a Batch of 20 students (1 per batch)

1. SOFTWARE REQUIRED – **Turbo C,C++**
2. OPERATING SYSTEM – **WINDOWS 2000 / XP / NT**
3. COMPUTERS SPECIFICATION– **20 Nos**. (Minimum Requirement: Pentium III orPentium IV with 1GB RAM and 40 GB hard disk)

SR. NO.	EXPERIMENT
1.	Implementation and Time analysis of Bubble sort and Selections sort.
2.	Write a program to implement Insertion sort.
3.	Write a program to implement Quick sort.
4.	Implementation and Time analysis of linear and binary search algorithm.
5.	Implementation of max-heap sort algorithm
6.	Implementation and Time analysis of factorial program using iterative and recursive method.
7.	Implementation of a knapsack problem using dynamic programming.
8.	Implementation of chain matrix multiplication using dynamic programming.
9.	Implementation of making a change problem using dynamic programming.
10.	Implementation of a knapsack problem using greedy algorithm
11.	Implementation of Graph and Searching (DFS and BFS).
12.	Implement prim's algorithm
13.	Implement Kruskal's algorithm.
14.	Implement Travelling Salesman problem.

EXPERIMENT - 1

AIM: Implementation of Bubble sort and Selectionssort.

Bubble sort

Input:

```
#include<stdio.h>
#include<conio.h>
int n;
void main( )
{
 int i,A[10];
 void bubble(int A[10]);
 clrscr( );
 printf("\n\t\t Bubble  Sort\n");
 printf("\n How many elements are there?");
 scanf("%d",&n);
 printf("\n Enter the elements\n");
 for(i=0;i<n;i++)
     scanf("%d",&A[i]);
     bubble(A);

 getch( );
}
void bubble(int A[10])
{
 int i,j,temp;
 for(i=1;i<=n;i++)
 {
     for(j=1;j<=n;j++)
     {
     if(A[j]>A[j+1])
     {
      temp=A[j];
      A[j]=A[j+1];
      A[j+1]=temp;
     }
 }
}
 printf("\n The sorted List is ...\n");
 for(i=0;i<n;i++)
     printf("  %d",A[i]);
  }
```

Selection sort

Input:

```
#include<stdio.h>
#include<conio.h>
int n;
void main( )
{
 int i,A[10];
 void bubble(int A[10]);
 clrscr( );
 printf("\n\t\t Bubble  Sort\n");
 printf("\n How many elements are there?");
 scanf("%d",&n);
 printf("\n Enter the elements\n");
 for(i=0;i<n;i++)
     scanf("%d",&A[i]);
     bubble(A);

 getch( );
}
void bubble(int A[10])
{
 int i,j,temp;
 for(i=1;i<=n;i++)
 {
     for(j=i+1;j<=n;j++)
     {
     if(A[i]>A[j])
     {
      temp=A[i];
      A[i]=A[j];
      A[j]=temp;
     }
 }
}
 printf("\n The sorted List is ...\n");
 for(i=0;i<n;i++)
     printf("  %d",A[i]);
}
```

EXPERIMENT -2

AIM: Write a program to implement Insertion sort.

Input:

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

EXPERIMENT – 3

AIM: ***Write a program to implement Quick sort.***

Input:

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

#define SIZE 10
void Quick(int A[SIZE],int,int);
int Partition(int A[SIZE],int,int);
void swap(int A[SIZE],int *,int *);
int n;
int main()
{
          int i;
          int A[SIZE];
          printf("\n\t\t Quick Sort Method \n");
          printf("\n Generating the list using the random
numbers");
          srand(10000);
          n=10;
          for (i = 0; i < n; i++)
          {
           int val = n * ((float)rand() / ((float)RAND_MAX
+(float) 1));
           A[i] = val;
          }
          printf("\n The Original List is\n");
          for (i = 0; i < n; i++)
           printf(" %d",A[i]);
          Quick(A,0,n-1);

          printf("\n\n\t Sorted Array Is: \n");
          for(i=0;i<n;i++)
           printf(" %d",A[i]);
          return 0;
}

/*
This function is to sort the elements in a sublist
```

```
*/
void Quick(int A[SIZE],int low,int high)
{
          int m,i;
          if(low<high)
          {
             m=Partition(A,low,high);//setting pivot element
             Quick(A,low,m-1);//splitting of list
             Quick(A,m+1,high);//splitting  of list
          }

}
/*
This function is to partition a list and decide the pivot
element
*/
int Partition(int A[SIZE],int low,int high)
{
          int pivot=A[low],i=low,j=high;
          while(i<=j)
          {
           while(A[i]<=pivot)
                i++;
           while(A[j]>pivot)
                j--;
           if(i<j)
                swap(A,&i,&j);
          }
          swap(A,&low,&j);
          return j;
}

void swap(int A[SIZE],int *i,int *j)
{
          int temp;

          temp=A[*i];
          A[*i]=A[*j];
          A[*j]=temp;
}
```

EXPERIMENT – 4

AIM: **Implementation of linear and binary search algorithm.**

Input:

```
#include<stdio.h>
#include<conio.h>
#define SIZE 10
int n;
void main()
{
     int A[SIZE],KEY,i,flag,low,high;
     int BinSearch(int A[SIZE],int KEY,int low,int high);
     clrscr();
     printf("\n How Many elements in an array?");
     scanf("%d",&n);
     printf("\n Enter The Elements");
     for(i=0;i<n;i++)
     scanf("%d",&A[i]);
     printf("\n Enter the element which is to be searched");
     scanf("%d",&KEY);
     low=0;
     high=n-1;
     flag=BinSearch(A,KEY,low,high);
     printf("\n The element is at A[%d] location",flag);
     getch();
}
int BinSearch(int A[SIZE],int KEY,int low,int high)
{
  int m;
  m=(low+high)/2;   //mid of the array is obtained
  if(KEY==A[m])
          return m;
  else if(KEY<A[m])
          BinSearch(A,KEY,low,m-1);//search the left sub list
  else
           BinSearch(A,KEY,m+1,high);//search the right sub list
}
```

EXPERIMENT – 5

AIM: **Implementation of max-heap sort algorithm**

Input:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#define MAX 10
void main()
{
     int i,n;
     int arr[MAX];
     void makeheap(int arr[MAX],int n);
     void heapsort(int arr[MAX],int n);
     void display(int arr[MAX],int n);
     clrscr();
      for(i=0;i<MAX;i++)
           arr[i]=0;
     printf("\n How many elements you want to insert?");
     scanf("%d",&n);
     printf("\n Enter the elements");
      for(i=0;i<n;i++)
          scanf("%d",&arr[i]);
     printf("\n The Elements are ...");
     display(arr,n);
     makeheap(arr,n);
     printf("\n Heapified");
     display(arr,n);
     heapsort(arr,n);
     printf("\nElements sorted by Heap sort... ");
     display(arr,n);
     getch();
}
void makeheap(int arr[MAX],int n)
{
     int i,val,j,father;
     for(i=1;i<n;i++)
     {
          val=arr[i];
          j=i;
          father=(j-1)/2;//finding the parent of node j
```

```
            while(j>0&&arr[father]<val)//creating a MAX heap
            {
                  arr[j]=arr[father];//preserving parent dominance
                  j=father;
                  father=(j-1)/2;
            }
            arr[j]=val;
      }
}
void heapsort(int arr[MAX],int n)
{
            int i,k,temp,j;
            for(i=n-1;i>0;i--)
            {
                  temp=arr[i];
                  arr[i]=arr[0];
                  k=0;
                  if(i==1)
                     j=-1;
                  else
                    j=1;
            if(i>2&&arr[2]>arr[1])
                   j=2;
            while(j>=0&& temp <arr[j])
            {
                  arr[k]=arr[j];
                  k=j;
                  j=2*k+1;
                  if(j+1<=i-1&&arr[j]<arr[j+1])
                        j++;
                  if(j>i-1)
                        j=-1;
            }
                  arr[k]=temp;
            }
}

void display(int arr[MAX],int n)
{
      int i;
      for(i=0;i<n;i++)
       printf("\n %d",arr[i]);
}
```

EXPERIMENT – 6

AIM:- **Implementation of factorial program using iterative and recursive method.**

Input:

```
#include<stdio.h>
int factorial(int n);
int main()
{
    int n;
    printf("Enter an positive integer: ");
    scanf("%d",&n);
    printf("Factorial of %d = %ld", n, factorial(n));
    return 0;
}
int factorial(int n)
{
    if(n!=1)
     return n*factorial(n-1);
}
```

Output:

```
Enter an positive integer: 6
Factorial of 6 = 720
```

EXPERIMENT – 7

AIM:- **Implementation of a knapsack problem using dynamic programming**

Input:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int table[5][6];
void main()
{
int w[]={0,2,3,4,5};
int v[]={0,3,4,5,6};
int W=5;
int n=4;
void DKP(int n,int W,int w[],int v[]);
clrscr();
printf("\n\t\t 0/1 Knapsack Problem using Dynamic Programming");
/*initialization of table*/
for(int i=0;i<=n;i++)
{
 for(int j=0;j<=W;j++)
 {
   table[i][j]=0;
 }
}
 DKP(n,W,w,v);
}
int max(int a,int b)
{
 if(a>b)
  return a;
 else
  return b;
}
void DKP(int n,int W,int w[5],int v[5])
{
  void find_item(int,int,int[]);
  int i,j;
  int val1,val2;
     for(i=0;i<=n;i++)
     {
       for(j=0;j<=W;j++)
```

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

`

EXPERIMENT – 8

AIM:-**Implementation of chain matrix multiplication using dynamic programming**

Input:

```
#include<stdio.h>
int chain(int index[20][20],int chain_matrix[],int i,int j);
void print_order(int index[20][20],int i,int j);

int main()
{
    int size,i,value;
    int chain_matrix[100];
    int index[20][20];
    printf("Enter the number of matrices: ");
    scanf("%d",&size);

    printf("\nEnter the dimensions of %d matrices...\n",size);
    printf("\n");

    for(i=0;i<size+1;i++)
    {
        printf("Enter the dimensions of the matrix: ");
        scanf("%d",&chain_matrix[i]);
    }


    value=chain(index,chain_matrix,1,size);
    printf("\nNumber of multiplication: %d\n\n",value);
    printf("\nThe Optimal order is:\t");
    print_order(index,1,size);
    getch();
    return 0;

}
int chain(int index[20][20],int chain_matrix[],int i,int j)
{
  if(i==j)
    return 0;
  int  k,count;
  int min=1000000;
```

```
  for (k=i;k<j;k++)
  {

count=chain(index,chain_matrix,i,k)+chain(index,chain_matrix,k+1,j)+ch
ain_matrix[i-1]*chain_matrix[k]*chain_matrix[j];
    if(count<min)
    {
      min=count;
      index[i][j]=k;
    }
  }
  return min;
}
void print_order(int index[20][20],int i,int j)
{
  if(i==j)
  printf("M%d",i);
  else
 {
   printf("(");
   print_order(index,i,index[i][j]);
   print_order(index,index[i][j]+1,j);
   printf(")");
 }
}
```

EXPERIMENT – 9

AIM:Implementation of making a change problem using dynamic programming

Input:

```
#include<stdio.h>

// Returns the count of ways we can sum S[0...m-1] coins to get sum n
int count( int S[], int m, int n )
{
      // If n is 0 then there is 1 solution (do not include any coin)
      if (n == 0)
            return 1;

      // If n is less than 0 then no solution exists
      if (n < 0)
            return 0;

      // If there are no coins and n is greater than 0, then no
solution exist
      if (m <=0 && n >= 1)
            return 0;

      // count is sum of solutions (i) including S[m-1] (ii) excluding
S[m-1]
      return count( S, m - 1, n ) + count( S, m, n-S[m-1] );
}

// Driver program to test above function
int main()
{
      int i, j;
      int arr[] = {1, 2, 3};
      int m = sizeof(arr)/sizeof(arr[0]);
      printf("%d ", count(arr, m, 4));
      getchar();
      return 0;
}
```

Output:4

EXPERIMENT - 10

AIM:**Implementation of a knapsack problem using greedy algorithm**

Input:

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

void knapsack(int n,float m,float w[ ],float p[ ]);

void main( )
{
     int i,j,n;
     float p[15],w[15],c[15],temp,m;
     clrscr( );
     printf("\nEnter number of objects:");
     scanf("%d",&n);
     printf("\nEnter weights:");
     for(i=0;i<n;i++)
      scanf("%f",&w[i]);
     flushall( );
     printf("\nEnter profits:");

     for(i=0;i<n;i++)
      scanf("%f",&p[i]);
     printf("\nEnter knapsack size:");
     scanf("%f",&m);

     for(i=0;i<n;i++)
           c[i]=p[i]/w[i];

     for(i=0;i<n;i++)
     {
          for(j=0;j<n-1;j++)
           {
                if(c[j] < c[j+1])
                    {
                     temp=c[j];
                     c[j]=c[j+1];
                     c[j+1]=temp;

                     temp=w[j];
```

```
                        w[j]=w[j+1];
                        w[j+1]=temp;

                        temp=p[j];
                        p[j]=p[j+1];
                        p[j+1]=temp;
                    }
               }
          }
          printf("\n The items are arranged as  ...\n");
          printf("\n\nItems\tweights \tProfits");
          {
               for(i=0;i<n;i++)
               printf("\nx[%d]\t%.0f\t\t%.0f",i,w[i],p[i]);
          }
          knapsack(n,m,w,p);
          getch( );
}

void knapsack(int n,float m,float w[ ],float p[ ])
{
          float x[15],U,profit=0.0,weight=0.0;
          int i;
          U=m;

          for(i=0;i<n;i++)
               x[i]=0.0;

          for(i=0;i<n;i++)
          {
               if(w[i]>U)
               break;
               x[i]=1.0;
               U=U-w[i];
          }
          if(i<n)
               x[i]=U/w[i];//take fractional part of item to fulfil
the size

          printf("\nThe solution vector is:");

          for(i=0;i<n;i++)
               printf("\n%d\t\%.2f",i,x[i]);

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

```
        {
            w[i]=w[i]*x[i];
            p[i]=p[i]*x[i];
        }

        for(i=0;i<n;i++)
        {
            profit=profit+p[i];//computing total profit & wt.
            weight=weight+w[i];
        }

        printf("\nMaximum profit is:");
        printf("\n\t\t%.2f",profit);
        printf("\nMaximum weight is:");
        printf("\n\t\t%.2f",weight);

}
```

EXPERIMENT - 11

AIM:Implementation of Graph and Searching (DFS and BFS)

Breadth First search:

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#define size 20
#define   TRUE 1
#defineFALSE     0
int    g[size][size];
int    visit[size];
int    Q[size];
int    front, rear;
int n;
void main ()
{
     int v1, v2;
     char ans ='y';
     void create(),bfs(int v1);
     clrscr();
     create();
     clrscr();
     printf("The Adjacency Matrix for the graph is \n");
     for ( v1 = 0; v1 < n; v1++)
     {
          for ( v2 = 0; v2 < n; v2++)
               printf(" %d ",g[v1][v2]);
          printf("\n");
     }
     getch();
     do
     {
          for ( v1 = 0; v1 < n; v1++)
               visit[v1] = FALSE;
          clrscr();
          printf("Enter the Vertex from which you want to
                    traverse: ");
          scanf("%d", &v1);
          if ( v1 >= n )
```

```
                printf("Invalid Vertex\n");
            else
            {
                printf("The Breadth First Search of the Graph
                        is\n");
                bfs(v1);
                getch();
            }
            printf("\nDo you want to traverse from any other
                        node?");
            ans=getche();
        }while(ans=='y');
        exit(0);
}
void  create()
{
      int v1, v2;
      char ans='y';
      printf("\n\t\t This is a Program To Create a Graph");
      printf("\n\t\t The Display Is In Breadth First
           Manner");
      printf("\nEnter no. of nodes");
      scanf("%d",&n);
      for ( v1 = 0; v1 < n; v1++)
       for ( v2 = 0; v2 < n; v2++)
           g[v1][v2] = FALSE;
       printf("\nEnter the vertices no. starting from 0: ");
       do
       {
           printf("\nEnter the vertices v1 & v2: ");
           scanf("%d%d", &v1, &v2);
           if ( v1 >= n || v2 >= n)
                printf("Invalid Vertex Value\n " );
           else
           {
                g[v1][v2] = TRUE;
                g[v2][v1] = TRUE;
           }
           printf("\n\nAdd more edges??(y/n) ");
           ans=getche();
       }while(ans==’y’);
}

void bfs(int v1)
{
```

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

Depth First search:

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

#define size   20
#define TRUE   1
#define FALSE 0

int g[size][size];
int visit[size];

int    Q[size];
int    front, rear;
int n;
void main ()
{
      int v1, v2,flag;
      void create();
      void Dfs(int v1);
      int isconn();
      clrscr();
      create();
      clrscr();
```

```
     printf("The Adjacency Matrix for the graph is \n");
     for ( v1 = 0; v1 < n; v1++)
     {
          for ( v2 = 0; v2 < n; v2++)
               printf(" %d ", g[v1][v2]);
          printf("\n");
     }
     getch();
     for ( v1 = 0; v1 < n; v1++)
      visit[v1] = FALSE;
     Dfs(0);
      flag=isconn();
      if(flag==1)
       printf("\n The Roads are Connected to different cities");

}

void  create()
{
     int  v1, v2,dist;
     printf("\nEnter no. of nodes ");
     scanf("%d",&n);
     for ( v1 = 0; v1 < n; v1++)
          for ( v2 = 0; v2 < n; v2++)
               g[v1][v2] = FALSE;
     printf("\n Enter the distance between JPNAGAR and JAYANAGAR
");
     scanf("%d",&dist);
     g[0][1] = dist;
     printf("\n Enter the distance between JAYANAGAR and BTM ");
     scanf("%d",&dist);
     g[1][2] = dist;
     printf("\n Enter the distance between BTM to V.V. PURAM ");
     scanf("%d",&dist);
     g[2][3] = dist;
     printf("\n Enter the distance between V.V. PURAM to
JAYANAGAR ");
     scanf("%d",&dist);
     g[3][1] = dist;
     printf("\n Enter the distance between BTM to JPNAGAR ");
     scanf("%d",&dist);
     g[2][0] = dist;

}
```

```
void Dfs(int v1)
{
  int v2;
  printf("\nThe road is connected to:");
  switch(v1)
  {
    case 0:printf(" JPNAGAR ");
        break;
    case 1:printf(" JAYANAGAR ");
        break;
    case 2:printf(" BTM ");
        break;
    case 3:printf(" V.V.PURAM ");
        break;
    case 4:printf(" K.R.PURAM ");
        break;

  }
  visit[v1] = TRUE;
  for ( v2 = 0; v2 < size; v2++)
    if ( g[v1][v2] != 0 &&  visit[v2] == FALSE )
      Dfs(v2);
}
int isconn()
{
 int i;
 for(i=0;i<n;i++)
 {
  if(visit[i]==FALSE)
  {
    printf("\n The Road to K.R.PURAM is not Connected");
    return 0;
  }
 }
 return 1;

}
```

EXPERIMENT - 12

AIM: Implement prim's algorithm

Input:

```
# include<stdio.h>
# include<conio.h>
# define SIZE 20
# define INFINITY 32767

/* This function finds the minimal spanning tree by Prim's
Algorithm  */

void Prim(int G[][SIZE], int nodes)

{
      int select[SIZE], i, j, k;
      int min_dist, v1, v2,total=0;

      for (i=0 ; i<nodes ; i++)   // Initialize the selected
vertices list
                select[i] = 0;

      printf("\n\n The Minimal Spanning Tree Is :\n");
      select[0] = 1;
      for (k=1 ; k<nodes ; k++)
      {
               min_dist = INFINITY;
               for (i=0 ; i<nodes ; i++) // Select an edge such
that one vertex is
               {                         // selected and other
is not and the edge
      for (j=0 ; j<nodes ; j++) // has the least weight.
      {
          if (G[i][j] && ((select[i] && !select[j]) ||
(!select[i] && select[j])))
               {
                     if (G[i][j] < min_dist)//obtained edge with
minimum wt
                     {
                         min_dist = G[i][j];
```

```
                        v1 = i;
                        v2 = j;   //picking up those vertices
                    }
                }
            }
        }
        printf("\n Edge (%d %d )and weight = %d",v1,v2,min_dist);
        select[v1] = select[v2] = 1;
        total =total+min_dist;
        }
     printf("\n\n\t Total Path Length Is = %d",total);
}

void main()
{
        int G[SIZE][SIZE], nodes;
        int v1, v2, length, i, j, n;
        clrscr();
        printf("\n\t Prim'S Algorithm\n");
        nodes=4;
        n=4;

        for (i=0 ; i<nodes ; i++)        // Initialize the graph
            for (j=0 ; j<nodes ; j++)
                 G[i][j] = 0;
        G[0][1]=G[1][0]=1;
        G[0][2]=G[2][0]=5;
        G[0][3]=G[3][0]=2;
        G[2][3]=G[3][2]=3;
        printf("\n Graph is created");
        getch();
        printf("\n\t");
        Prim(G,nodes);
        getch();
}
```

EXPERIMENT - 13

AIM: Implement Kruskal's algorithm

Input:

```
#include<stdio.h>
#define INFINITY 999
typedef struct Graph
{
 int v1;
 int v2;
 int cost;
}GR;
GR G[20];
int tot_edges,tot_nodes;
void create();
void spanning_tree();
int Minimum(int);

void main()
{
 printf("\n\t Graph Creation by adjacency matrix ");
 create();
 spanning_tree();
}
void create()
{
 int k;
 printf("\n Enter Total number of nodes: ");
 scanf("%d",&tot_nodes);
 printf("\n Enter Total number of edges: ");
 scanf("%d",&tot_edges);

 for(k=0;k<tot_edges;k++)
 {

          printf("\n Enter Edge in (V1 V2)form ");
          scanf("%d%d",&G[k].v1,&G[k].v2);
          printf("\n Enter Corresponding Cost ");
          scanf("%d",&G[k].cost);
 }
}
void spanning_tree()
```

```
{
 int count,k,v1,v2,i,j,tree[10][10],pos,parent[10];
 int sum;
 int Find(int v2,int parent[]);
 void Union(int i,int j,int parent[]);
 count=0;
 k=0;
 sum=0;
 for(i=0;i<tot_nodes;i++)
          parent[i]=i;
 while(count!=tot_nodes-1)
 {
          pos=Minimum(tot_edges);//finding the minimum cost edge
          if(pos==-1)//Perhaps no node in the graph
                break;
          v1=G[pos].v1;
          v2=G[pos].v2;
          i=Find(v1,parent);
          j=Find(v2,parent);
          if(i!=j)
          {
                tree[k][0]=v1;//storing the minimum edge in
array tree[]
                tree[k][1]=v2;
                k++;
                count++;
                sum+=G[pos].cost;//accumulating the total cost
of MST
                Union(i,j,parent);
          }
          G[pos].cost=INFINITY;
       }
          if(count==tot_nodes-1)
          {
           printf("\n Spanning tree is...");
           printf("\n ------------------------\n");
           for(i=0;i<tot_nodes-1;i++)
           {
                printf("[%d",tree[i][0]);
                printf(" - ");
                printf("%d",tree[i][1]);
                printf("]");
           }
           printf("\n ------------------------");
           printf("\nCost of Spanning Tree is = %d",sum);
```

```
            }
            else
             {
                  printf("There is no Spanning Tree");
             }
}
int Minimum(int n)
{
 int i,small,pos;
 small=INFINITY;
 pos=-1;
 for(i=0;i<n;i++)
 {
            if(G[i].cost<small)
            {
                 small=G[i].cost;
                 pos=i;
            }
 }
 return pos;
}
int Find(int v2,int parent[])
{
 while(parent[v2]!=v2)
 {
            v2=parent[v2];
       }
       return v2;
}
void Union(int i,int j,int parent[])
{
 if(i<j)
             parent[j]=i;
 else
             parent[i]=j;
}
```

EXPERIMENT - 14

AIM: ***Implement Travelling Salesman problem.***

***Input*:**

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

typedef struct
{
  int nodes[MAX];
  int vertex;
  int min;
}Path_node;          // structure to store the path

Path_node TSP(int source, Path_node list, int Element[][MAX],
int max_no_cities)
{
 int i, j;
 Path_node new_list, new_path, new_min;
 if(list.vertex == 0)
 {
   new_min.min = Element[source][1];
   new_min.nodes[max_no_cities-1] = source; // store the vertex
from in the list
   new_min.vertex = max_no_cities;
   return new_min;
 }
 for(i=0;i<list.vertex;i++)   //going through all the vertices
 {
   new_list.vertex = 0;
   for(j = 0; j < list.vertex; j++)
   if(i != j)
     new_list.nodes[new_list.vertex++] = list.nodes[j];
   new_path = TSP(list.nodes[i], new_list, Element,
max_no_cities);
                            // call recursively
   new_path.min = Element[source][list.nodes[i]] + new_path.min;
                            // updating new path
   new_path.nodes[max_no_cities - list.vertex -1] = source;
```

```
   if(i == 0)       // if the new_path is for the 1st node then
that is the
             // current minimum path
   new_min = new_path;
   else              // else check for better path
      if(new_path.min < new_min.min)
     new_min = new_path;
 }
 return new_min;    // return the newly obtained min path
}                  // This is the minimum path
void display(Path_node Path)
{
 int i;
 printf("\n\nThe minimum cost is %d\n", Path.min);
 printf("\n The path is...\n");
 for(i = 0; i < Path.vertex;i++)
   printf("%d - - ", Path.nodes[i]);
 printf("%d", Path.nodes[0]);//returning to original node
}
main( )
{
 int i, j, Element[MAX][MAX], max_no_cities;
 Path_node Graph, Path;
 clrscr( );
 printf("\n How Many Number of Cities are there? ");
 scanf("%d", &max_no_cities);
     // accept the no. of vertices
 if(max_no_cities==0)
 {
  printf("Error:There is no city for processing the TSP");
 }
 else
 {
  for(i = 1; i <= max_no_cities; i++)
  {
   for(j = 1; j <= max_no_cities; j++)
   if(i == j)
     Element[i][i] = 0;//self referancing path is set to 0
   else
   {
    printf("Enter distace from city %d to %d ? ", i, j);
    scanf("%d", &Element[i][j]);//create graph for finding TSP
   }
   if(i > 1)
    Graph.nodes[i-2] = i;
```

```
  }
  Graph.vertex = max_no_cities - 1;//total number of cities
  Path = TSP(1, Graph, Element, max_no_cities);
  display(Path);
 }
 getch( );
 return 1;
}
```