SISTER NIVEDITA UNIVERSITY

DESIGN AND ANALYSIS OF ALGORITHMS LAB FILE

SUBMITTED BY: PIYUSH CHANDRA CHANDRA

DEPT- BTECH (CSE), ROLL- 1027

SUBMITTED TO: SWARUP KUMAR GHOSH

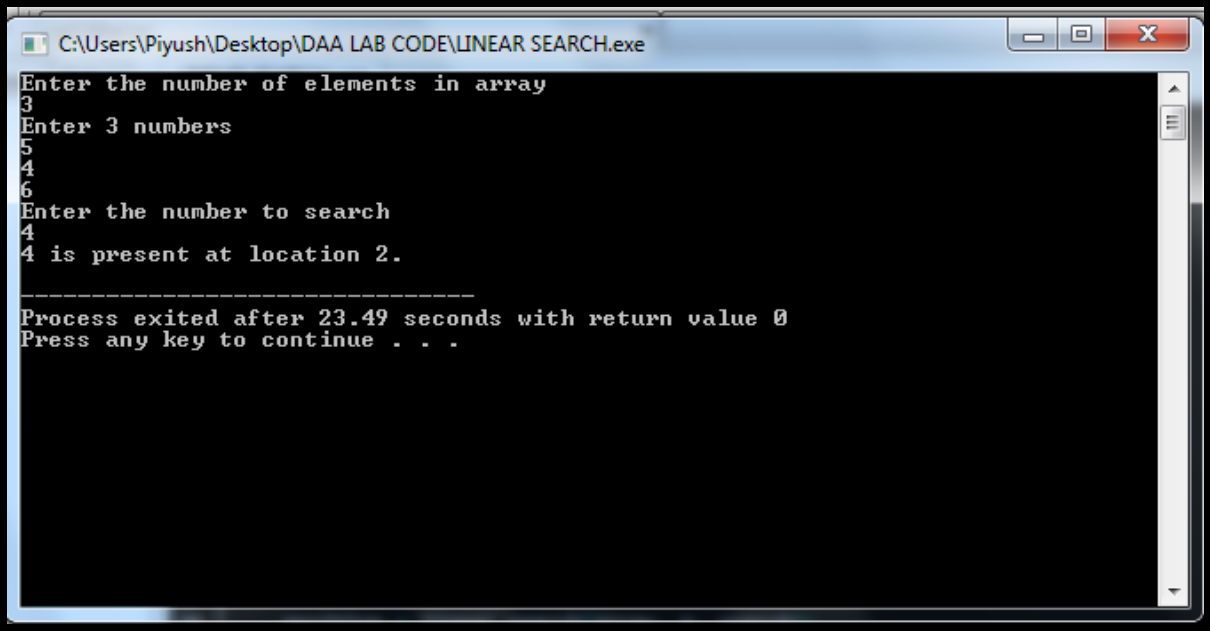
INDEX

1 Write a C program to perform Linear search.

```
#include<stdio.h>
Int  linear_search(int*, int, int);
main()
{
  int array[100], search, c, n, position;
  printf("Enter the number of elements in array\n");
  scanf("%d",&n);
  printf("Enter %d numbers\n", n);
  for ( c = 0 ; c < n ; c++ )
    scanf("%d",&array[c]);
  printf("Enter the number to search\n");
  scanf("%d",&search);
  position = linear_search(array, n, search);
  if ( position == -1 )
    printf("%d is not present in array.\n", search);
  else
    printf("%d is present at location %d.\n", search, position+1);
  return 0;
}
int linear_search(int *pointer, int n, int find)
{
  int c;
  for ( c = 0 ; c < n ; c++ )
  {
```

```
    if ( *(pointer+c) == find )
      return c;
   }
   return -1;
  }
```

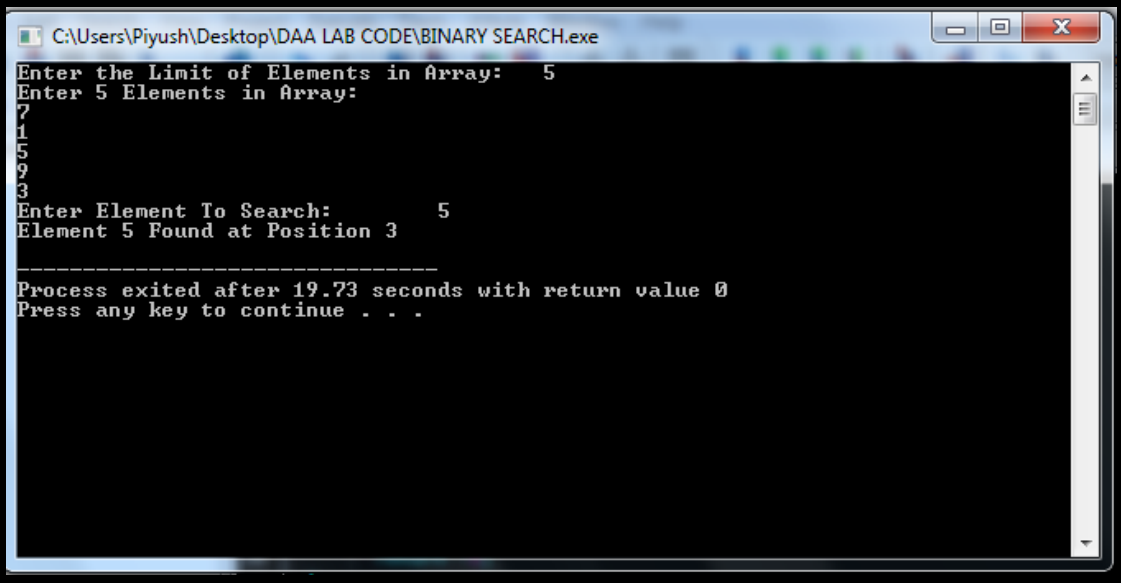
OUTPUT:



2 Write a C program to perform Binary search using divide and conquer approach.

```
#include <stdio.h>

int BinarySearching(int arr[], int max, int element)
{
    int low = 0, high = max - 1, middle;
    while(low <= high)
    {
        middle = (low + high) / 2;
        if(element > arr[middle])
            low = middle + 1;
        else if(element < arr[middle])
            high = middle - 1;
        else
            return middle;
    }
    return -1;
}

int main()
{
    int count, element, limit, arr[50], position;
    printf("Enter the Limit of Elements in Array:\t");
    scanf("%d", &limit);
    printf("Enter %d Elements in Array: \n", limit);
    for(count = 0; count < limit; count++)
    {
        scanf("%d", &arr[count]);
    }
    printf("Enter Element To Search:\t");
    scanf("%d", &element);
    position = BinarySearching(arr, limit, element);
    if(position == -1)
    {
        printf("Element %d Not Found\n", element);
    }
    else
    {
        printf("Element %d Found at Position %d\n", element, position + 1);
    }
    return 0;
}
```

OUTPUT:


C:\Users\Piyush\Desktop\DAA LAB CODE\BINARY SEARCH.exe
Enter the Limit of Elements in Array: 5
Enter 5 Elements in Array:
7
1
5
9
3
Enter Element To Search: 5
Element 5 Found at Position 3

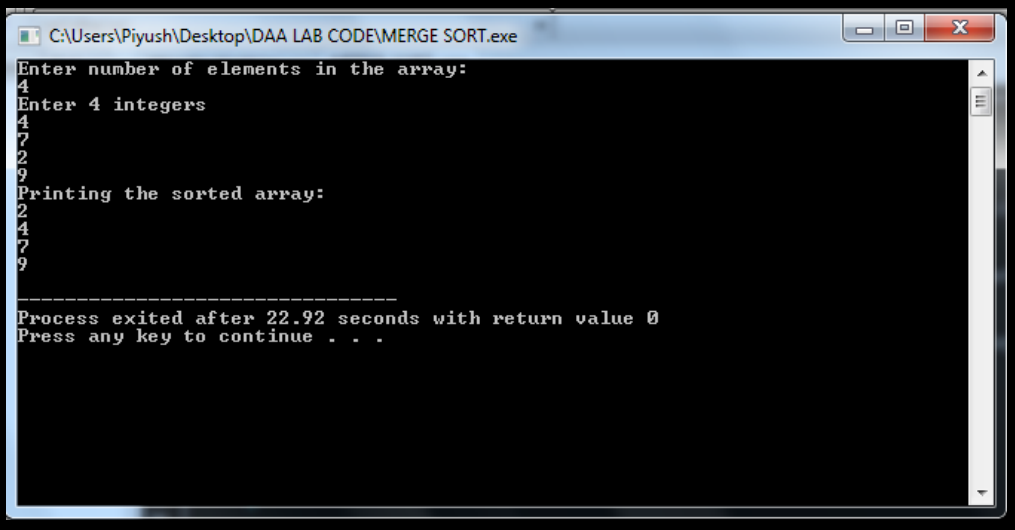
Process exited after 19.73 seconds with return value 0
Press any key to continue . . .

3 Write a C program to perform Merge Sort using divide and conquer approach.

```
#include <stdio.h>
void merge_sort(int i, int j, int a[], int aux[]) {
  if (j <= i) {
    return;    // the subsection is empty or a single element
  }
  int mid = (i + j) / 2;
  merge_sort(i, mid, a, aux);
  merge_sort(mid + 1, j, a, aux);
  int pointer_left = i
  int pointer_right = mid + 1
  int k
  for (k = i; k <= j; k++) {
    if (pointer_left == mid + 1) {
      aux[k] = a[pointer_right];
      pointer_right++;
    } else if (pointer_right == j + 1) {
      aux[k] = a[pointer_left];
      pointer_left++;
    } else if (a[pointer_left] < a[pointer_right]) {
      aux[k] = a[pointer_left];
      pointer_left++;
    } else {
      aux[k] = a[pointer_right];
      pointer_right++;
    }
```

```
    }
    for (k = i; k <= j; k++) {
        a[k] = aux[k];
    }
  }
}
int main() {
 int a[100], aux[100], n, i, d, swap;
 printf("Enter number of elements in the array:\n");
 scanf("%d", &n);
 printf("Enter %d integers\n", n);
 for (i = 0; i < n; i++)
  scanf("%d", &a[i]);
 merge_sort(0, n - 1, a, aux);
 printf("Printing the sorted array:\n");
 for (i = 0; i < n; i++)
   printf("%d\n", a[i]);
 return 0;
}
```

OUTPUT:

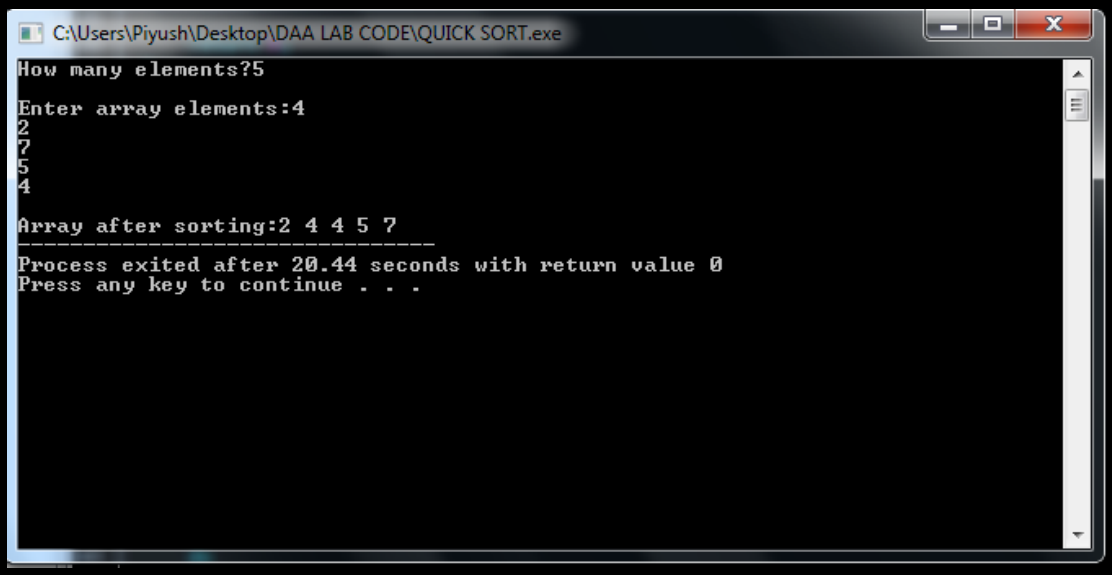


4 Write a C program to perform Quick Sort using divide and conquer approach.

```
#include <stdio.h>
void quick_sort(int[],int,int);
int partition(int[],int,int);
int main()
{
        int a[50],n,i;
        printf("How many elements?");
        scanf("%d",&n);
        printf("\nEnter array elements:");

        for(i=0;i<n;i++)
                scanf("%d",&a[i]);

        quick_sort(a,0,n-1);
        printf("\nArray after sorting:");

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

        return 0;
}

void quick_sort(int a[],int l,int u)
{
        int j;
        if(l<u)
        {
                j=partition(a,l,u);
                quick_sort(a,l,j-1);
                quick_sort(a,j+1,u);
        }
}
int partition(int a[],int l,int u)
{
        int v,i,j,temp;
        v=a[l];
        i=l;
        j=u+1;

        do
        {
                do
                        i++;
                while(a[i]<v&&i<=u);

                do
```

```
                j--;
            while(v<a[j]);

            if(i<j)
            {
                temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }while(i<j);
        a[l]=a[j];
        a[j]=v;
        return(j);
    }
```

OUTPUT:



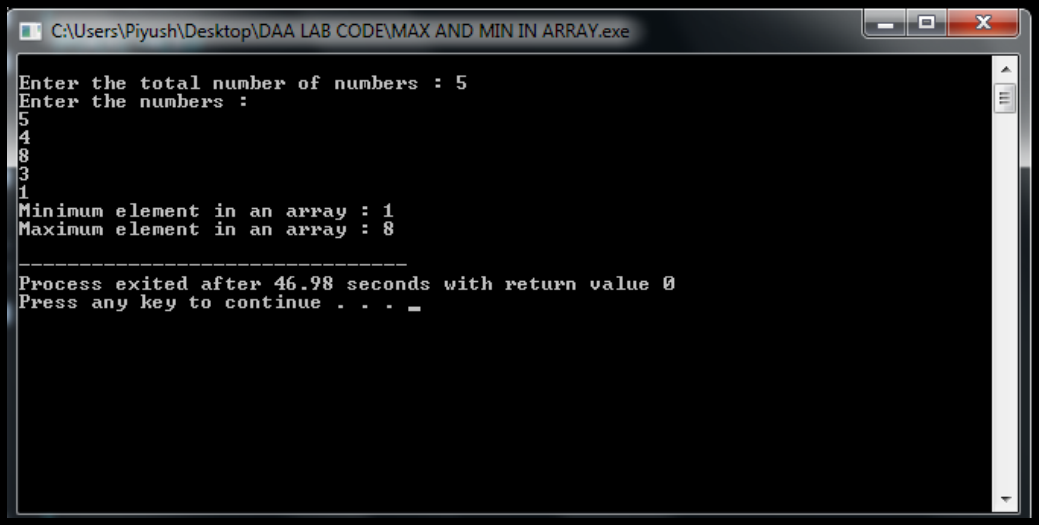
5 Write a program in C to Find Maximum and Minimum element from a array of integer using divide and conquer approach.

```
#include<stdio.h>
#include<stdio.h>
int max, min;
int a[100];
void maxmin(int i, int j)
{
 int max1, min1, mid;
 if(i==j)
 {
  max = min = a[i];
 }
 else
 {
  if(i == j-1)
  {
   if(a[i] <a[j])
   {
    max = a[j];
    min = a[i];
   }
   else
   {
    max = a[i];
    min = a[j];
   }
```

```
 }
 else
 {
  mid = (i+j)/2;
  maxmin(i, mid);
  max1 = max; min1 = min;
  maxmin(mid+1, j);
  if(max <max1)
   max = max1;
  if(min > min1)
   min = min1;
 }
}
}
int main ()
{
 int i, num;
 printf ("\nEnter the total number of numbers : ");
 scanf ("%d",&num);
 printf ("Enter the numbers : \n");
 for (i=1;i<=num;i++)
  scanf ("%d",&a[i]);
 max = a[0];
 min = a[0];
 maxmin(1, num);
 printf ("Minimum element in an array : %d\n", min);
 printf ("Maximum element in an array : %d\n", max);
```

```
 return 0;
}
```

OUTPUT:

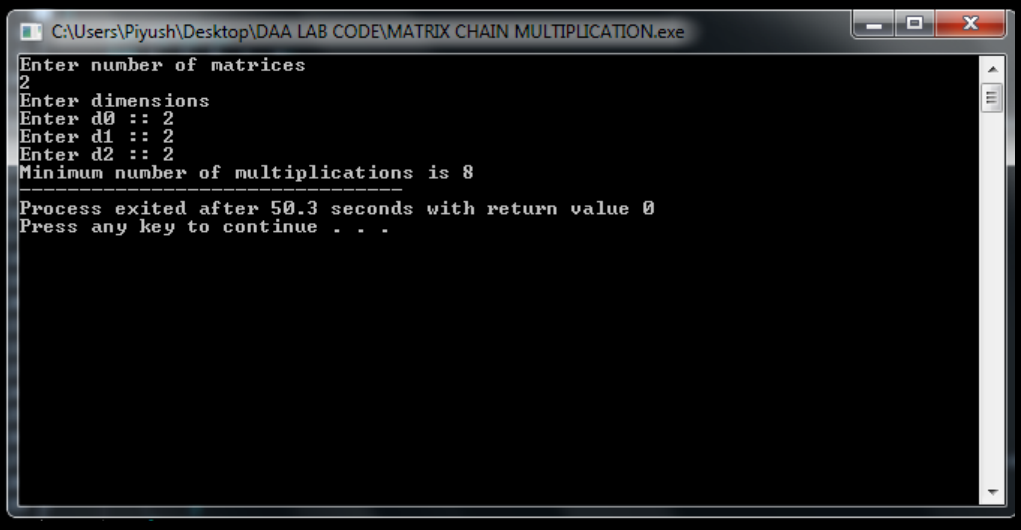


6. Write a C program to Implement matrix chain multiplication.

```
#include<stdio.h>
#include<limits.h>
int MatrixChainMultiplication(int p[], int n)
{
   int m[n][n];
   int i, j, k, L, q;
   for (i=1; i<n; i++)
      m[i][i] = 0
   for (L=2; L<n; L++)
   {
      for (i=1; i<n-L+1; i++)
      {
         j = i+L-1;
         m[i][j] = INT_MAX;
         for (k=i; k<=j-1; k++)
         {
            q = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
            if (q < m[i][j])
            {
               m[i][j] = q;
            }
         }
      }
   }

   return m[1][n-1];
}
int main()
{
   int n,i;
   printf("Enter number of matrices\n");
```

```
    scanf("%d",&n);
    n++;
    int arr[n];
    printf("Enter dimensions \n");
    for(i=0;i<n;i++)
    {
      printf("Enter d%d :: ",i);
      scanf("%d",&arr[i]);
    }
    int size = sizeof(arr)/sizeof(arr[0]);
    printf("Minimum number of multiplications is %d ",
  MatrixChainMultiplication(arr, size));
    return 0;
  }
```

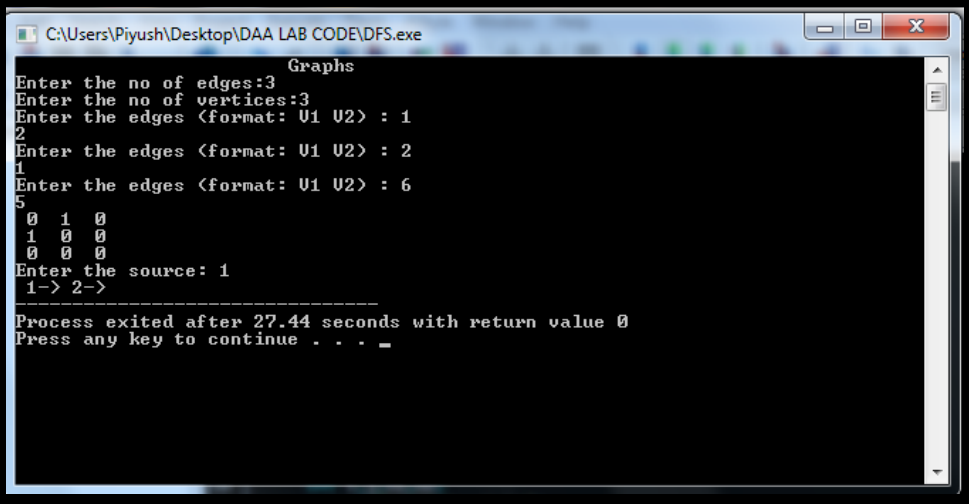
OUTPUT:



7. Write a C program to Implement Depth First Search (DFS) Algorithm.

```
#include <stdio.h>
#include <stdlib.h>
int source,V,E,time,visited[20],G[20][20];
void DFS(int i)
{
  int j;
  visited[i]=1;
  printf(" %d->",i+1);
  for(j=0;j<V;j++)
  {
    if(G[i][j]==1&&visited[j]==0)
      DFS(j);
  }
}
int main()
{
  int i,j,v1,v2;
  printf("\t\t\tGraphs\n");
  printf("Enter the no of edges:");
  scanf("%d",&E);
  printf("Enter the no of vertices:");
  scanf("%d",&V);
  for(i=0;i<V;i++)
  {
    for(j=0;j<V;j++)
```

```
            G[i][j]=0;
        }
        for(i=0;i<E;i++)
        {
            printf("Enter the edges (format: V1 V2) : ");
            scanf("%d%d",&v1,&v2);
            G[v1-1][v2-1]=1;
        }
        for(i=0;i<V;i++)
        {
            for(j=0;j<V;j++)
                printf(" %d ",G[i][j]);
            printf("\n");
        }
        printf("Enter the source: ");
        scanf("%d",&source);
            DFS(source-1);
        return 0;
    }
```

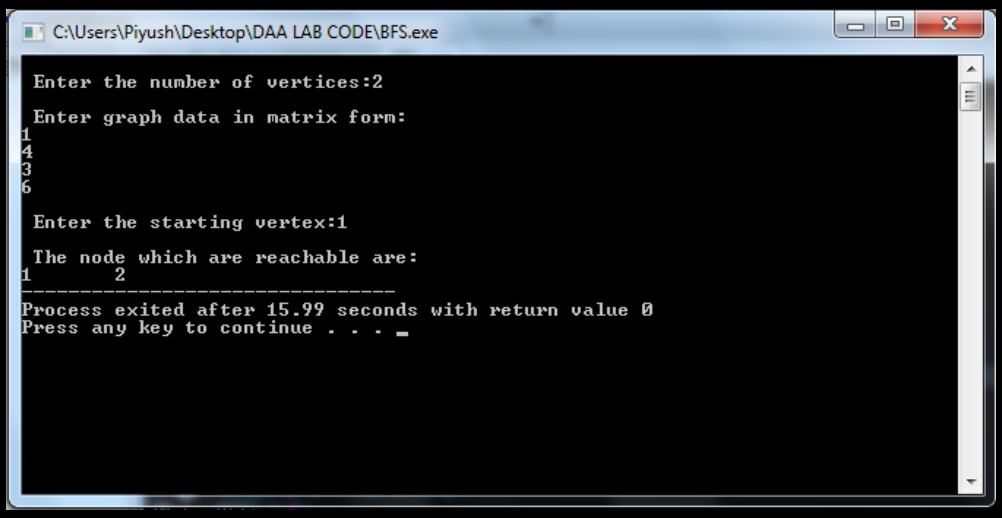


8. Write a C program to Implement Breadth First Search (BFS) Algorithm.

```
#include<stdio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;
void bfs(int v) {
 for(i = 1; i <= n; i++)
 if(a[v][i] && !visited[i])
 q[++r] = i;
 if(f <= r) {
 visited[q[f]] = 1;
 bfs(q[f++]);
 }
}
int main() {
 int v;
 printf("\n Enter the number of vertices:");
 scanf("%d", &n);
 for(i=1; i <= n; i++) {
 q[i] = 0;
 visited[i] = 0;
 }
 printf("\n Enter graph data in matrix form:\n");
 for(i=1; i<=n; i++) {
 for(j=1;j<=n;j++) {
 scanf("%d", &a[i][j]);
 }
 }
```

```
printf("\n Enter the starting vertex:");
scanf("%d", &v);
bfs(v);
printf("\n The node which are reachable are:\n");
for(i=1; i <= n; i++) {
if(visited[i])
printf("%d\t", i);
else {
printf("\n Bfs is not possible. Not all nodes are reachable");
break;
}
}
}
```

OUTPUT:



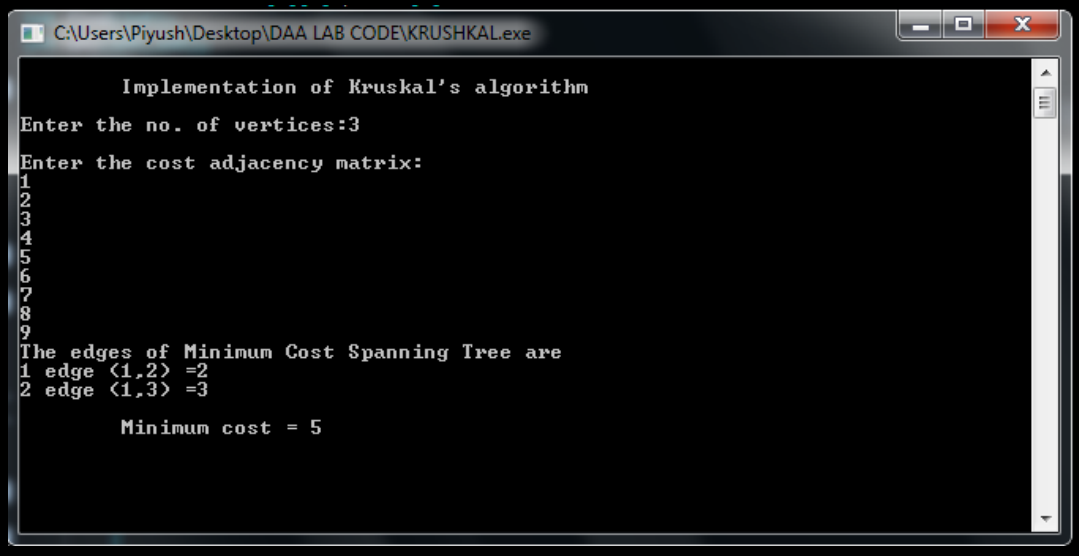
9. Write a C program to Implement Minimum Cost Spanning Tree by Kruskal's Algorithm

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);
int main()
{
printf("\n\tImplementation of Kruskal's algorithm\n");
printf("\nEnter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter 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");
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;
        }
       }
}
u=find(u);
v=find(v);
if(uni(u,v))
{
printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
mincost +=min;
}
cost[a][b]=cost[b][a]=999;
}
printf("\n\tMinimum cost = %d\n",mincost);
getch();
}
int find(int i)
{
while(parent[i])
```

```
i=parent[i];
return i;
}
int uni(int i,int j)
{
if(i!=j)
{
parent[j]=i;
return 1;
}
return 0;
}
```

OUTPUT:



10. Write a C program to Implement Minimum Cost Spanning Tree by Prim's Algorithm

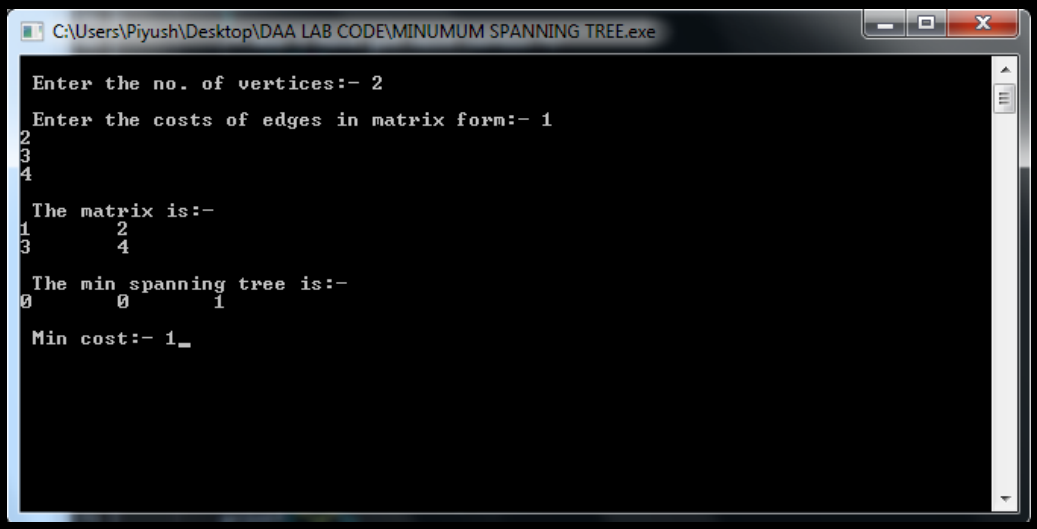
```
#include<stdio.h>
#include<conio.h>
int n, cost[10][10];
void prim()
{
int i,j,k,l,x,nr[10],temp,min_cost=0,tree[10][3];
temp=cost[0][0];
for(i=0;i < n;i++)
{
for(j=0;j < n;j++)
{
if(temp > cost[i][j])
{
temp=cost[i][j];
k=i;
l=j;
} } }
tree[0][0]=k;
tree[0][1]=l;
tree[0][2]=temp;
min_cost=temp;
for(i=0;i< n;i++)
{
if(cost[i][k]< cost[i][l])
nr[i]=k;
```

```
else
nr[i]=l;
}
nr[k]=100;
nr[l]=100;
temp=99;
for(i=1;i< n-1;i++)
{
for(j=0;j< n;j++)
{
if(nr[j]!=100 && cost[j][nr[j]] < temp)
{
temp=cost[j][nr[j]];
x=j;
}
}
tree[i][0]=x;
tree[i][1]=nr[x];
tree[i][2]=cost[x][nr[x]];
min_cost=min_cost+cost[x][nr[x]];
nr[x]=100;
for(j=0;j< n;j++)
{
if(nr[j]!=100 && cost[j][nr[j]] > cost[j][x])
nr[j]=x;
}
temp=99;
```

```
}
printf("\n The min spanning tree is:- \n");
for(i=0;i < n-1;i++)
{
for(j=0;j < 3;j++)
printf("%d\t", tree[i][j]);
printf("\n");
}
printf("\n Min cost:- %d", min_cost);
}
int main()
{
int i,j;
printf("\n Enter the no. of vertices:- ");
scanf("%d", &n);
printf ("\n Enter the costs of edges in matrix form:- ");
for(i=0;i< n;i++)
for(j=0;j< n;j++)
scanf("%d",&cost[i][j]);
printf("\n The matrix is:- \n");
for(i=0;i< n;i++)
{
for(j=0;j < n;j++)
printf("%d\t",cost[i][j]);
printf("\n");
}
prim();
```

```
getch();
}
```

OUTPUT:



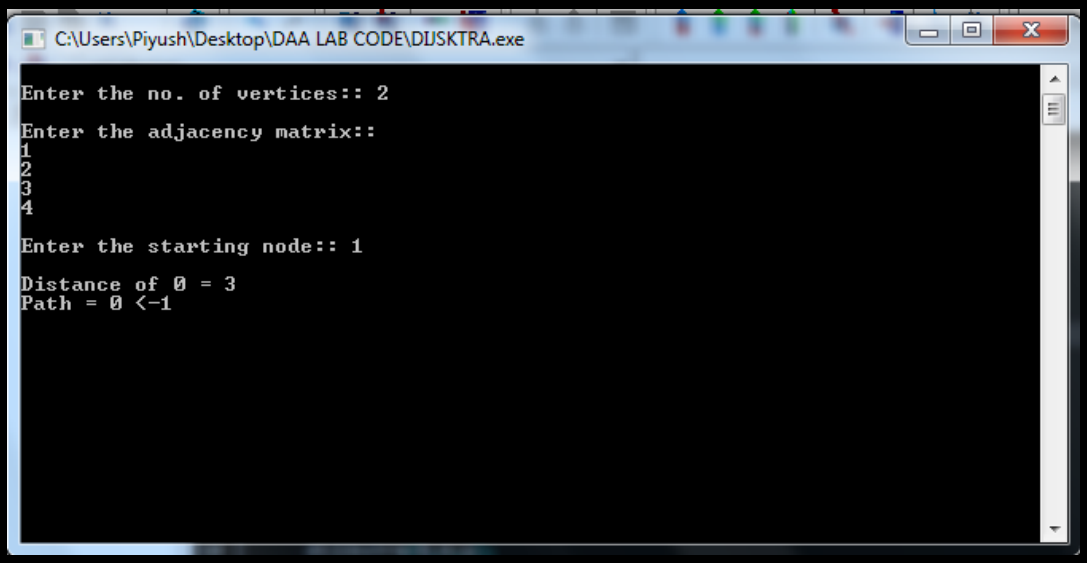
11. Write a C program to Implement single source shortest path for a graph (Dijkstra Algorithm).

```
#include<stdio.h>
#include<conio.h>
#define INFINITY 9999
#define MAX 10

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

void main(){
      int G[MAX][MAX], i, j, n, u;
      clrscr();
      printf("\nEnter the no. of vertices:: ");
      scanf("%d", &n);
      printf("\nEnter the adjacency matrix::\n");
      for(i=0;i < n;i++)
            for(j=0;j < n;j++)
                  scanf("%d", &G[i][j]);
      printf("\nEnter the starting node:: ");
      scanf("%d", &u);
      dijikstra(G,n,u);
      getch();
}
void dijikstra(int G[MAX][MAX], int n, int startnode)
{
```

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

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

      for(i=0;i < n;i++)
            if(i!=startnode)
            {
            printf("\nDistance of %d = %d", i, distance[i]);
                  printf("\nPath = %d", i);
                  j=i;
                  do
                  {
                        j=pred[j];
                        printf(" <-%d", j);
                  }
                  while(j!=startnode);
            }
}
```

OUTPUT:


C:\Users\Piyush\Desktop\DAA LAB CODE\DIJSKTRA.exe
Enter the no. of vertices:: 2
Enter the adjacency matrix::
1
2
3
4
Enter the starting node:: 1
Distance of 0 = 3
Path = 0 <-1

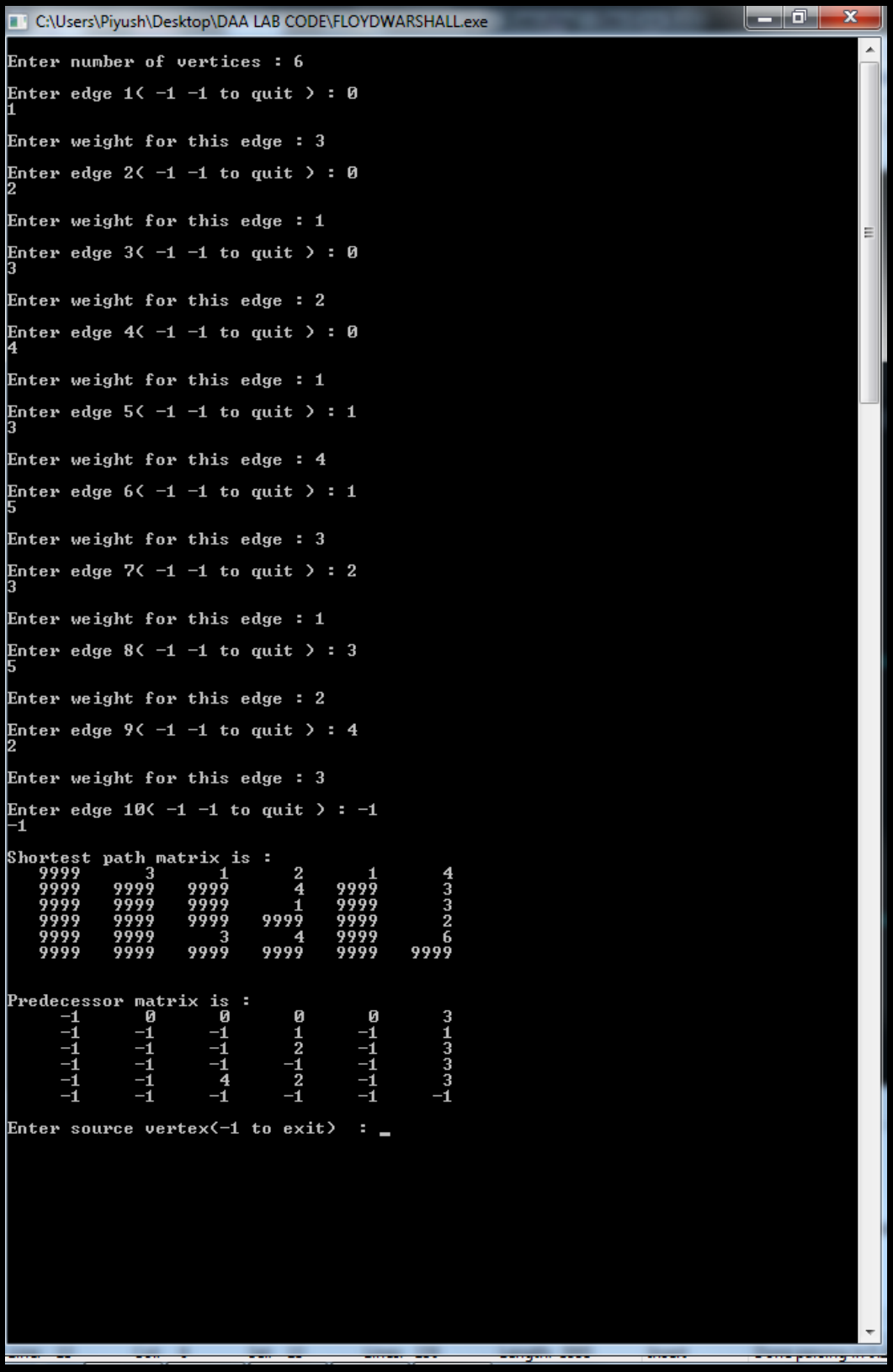
12. Write a C program to implement all pair shortest path using Floyd's algorithm.

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

#define infinity 9999
#define MAX 100

int n;
 int adj[MAX][MAX];
int D[MAX][MAX];
 int Pred[MAX][MAX];
void create_graph();
void FloydWarshalls( );
void findPath(int s, int d);
void display(int matrix[MAX][MAX], int n);

int main()
{
	int s, d;
	create_graph();
	FloydWarshalls();
	while(1)
	{
		printf("\nEnter source vertex(-1 to exit)  : ");
		scanf("%d",&s);
		if(s == -1)
			break;
		printf("\nEnter destination vertex : ");
		scanf("%d",&d);
		if( s < 0 || s>n-1 || d<0 || d>n-1)
		{
			printf("\nEnter valid vertices \n\n");
			continue;
		}
		printf("\nShortest path is : ");
		findPath(s,d);
		printf("\nLength of this path is %d\n",D[s][d]);
	}
}

void FloydWarshalls()
{
	int i,j,k;

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

```
                if(adj[i][j] == 0)
                {
                        D[i][j] = infinity;
                        Pred[i][j] = -1;
                }
                else
                {
                        D[i][j] = adj[i][j];
                        Pred[i][j] = i;
                }
        }

        for(k=0; k<n; k++)
        {
                for(i=0; i<n; i++)
                  for(j=0; j<n; j++)
                        if( D[i][k] + D[k][j] < D[i][j] )
                        {
                                D[i][j] = D[i][k] + D[k][j];
                                Pred[i][j] = Pred[k][j];
                        }
        }

        printf("\nShortest path matrix is :\n");
        display(D,n);

        printf("\n\nPredecessor matrix is :\n");
        display(Pred,n);

        for(i=0;i<n;i++)
                if(D[i][i]<0)
                {
                        printf("\nError :       negative cycle\n");
                        exit(1);
                }

}
void findPath(int s, int d)
{
        int i, path[MAX], count;

        if(D[s][d] == infinity)
        {
                printf("\nNo path \n");
                return;
        }

        count = -1;
        do
        {
```

```
            path[++count] = d;
            d = Pred[s][d];
        }while(d!=s);

        path[++count] = s;

        for(i=count; i>=0; i--)
            printf("%d ",path[i]);
        printf("\n");
}


void display(int matrix[MAX][MAX],int n )
{
        int i,j;
        for(i=0;i<n;i++)
        {
            for(j=0; j<n; j++)
                printf("%7d",matrix[i][j]);
            printf("\n");
        }
}
void create_graph()
{
        int i,max_edges,origin,destin, wt;

        printf("\nEnter number of vertices : ");
        scanf("%d",&n);
        max_edges = n*(n-1);

        for(i=1; i<=max_edges; i++)
        {
            printf("\nEnter edge %d( -1 -1 to quit ) : ",i);
            scanf("%d %d",&origin,&destin);

            if( (origin == -1) && (destin == -1) )
                break;

            printf("\nEnter weight for this edge : ");
            scanf("%d",&wt);

            if( origin >= n || destin >= n || origin<0 || destin<0)
            {
                printf("\nInvalid edge!\n");
                i--;
            }
            else
                adj[origin][destin] = wt;
        }
}
```

OUTPUT:


C:\Users\Piyush\Desktop\DAA LAB CODE\FLOYDWARSHALL.exe
Enter number of vertices : 6
Enter edge 1(-1 -1 to quit) : 0
1
Enter weight for this edge : 3
Enter edge 2(-1 -1 to quit) : 0
2
Enter weight for this edge : 1
Enter edge 3(-1 -1 to quit) : 0
3
Enter weight for this edge : 2
Enter edge 4(-1 -1 to quit) : 0
4
Enter weight for this edge : 1
Enter edge 5(-1 -1 to quit) : 1
3
Enter weight for this edge : 4
Enter edge 6(-1 -1 to quit) : 1
5
Enter weight for this edge : 3
Enter edge 7(-1 -1 to quit) : 2
3
Enter weight for this edge : 1
Enter edge 8(-1 -1 to quit) : 3
5
Enter weight for this edge : 2
Enter edge 9(-1 -1 to quit) : 4
2
Enter weight for this edge : 3
Enter edge 10(-1 -1 to quit) : -1
-1
Shortest path matrix is :
9999 3 1 2 1 4
9999 9999 9999 4 9999 3
9999 9999 9999 1 9999 3
9999 9999 9999 9999 9999 2
9999 9999 3 4 9999 6
9999 9999 9999 9999 9999 9999
Predecessor matrix is :
-1 0 0 0 0 3
-1 -1 -1 1 -1 1
-1 -1 -1 2 -1 3
-1 -1 -1 -1 -1 3
-1 -1 4 2 -1 3
-1 -1 -1 -1 -1 -1
Enter source vertex(-1 to exit) :

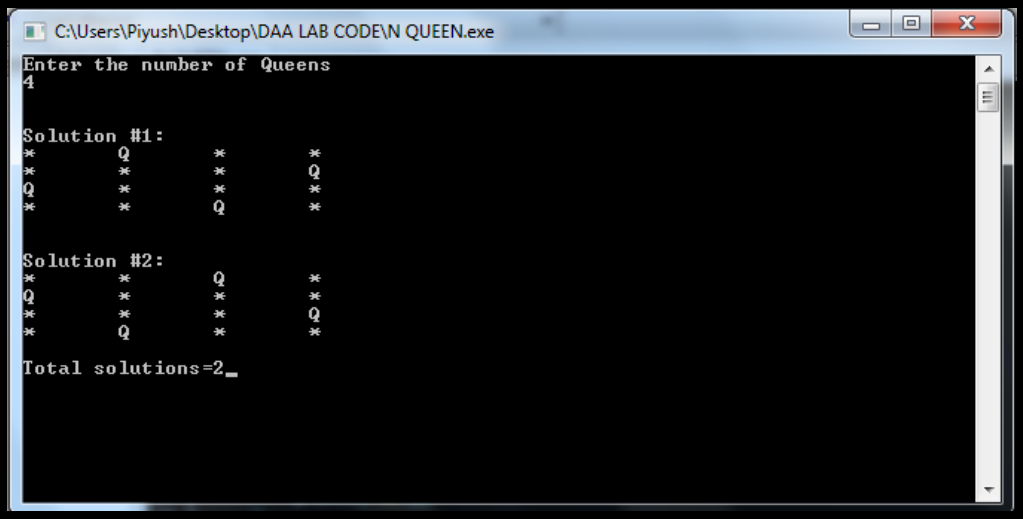
13. Write a C program to Implement N Queen problem

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
int a[30],count=0;
int place(int pos) {
      int i;
      for (i=1;i<pos;i++) {
            if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
               return 0;
      }
      return 1;
}
void print_sol(int n) {
      int i,j;
      count++;
      printf("\n\nSolution #%d:\n",count);
      for (i=1;i<=n;i++) {
            for (j=1;j<=n;j++) {
                  if(a[i]==j)
                        printf("Q\t"); else
                        printf("*\t");
            }
            printf("\n");
      }
```

```
}
void queen(int n) {
      int k=1;
      a[k]=0;
      while(k!=0) {
            a[k]=a[k]+1;
            while((a[k]<=n)&&!place(k))
               a[k]++;
            if(a[k]<=n) {
                  if(k==n)
                      print_sol(n); else {
                       k++;
                       a[k]=0;
                  }
            } else
               k--;
      }
}
void main() {
      int i,n;
      clrscr();
      printf("Enter the number of Queens\n");
      scanf("%d",&n);
      queen(n);
      printf("\nTotal solutions=%d",count);
```

```
        getch();
    }
```

OUTPUT:



14. Write a C program to Implement Traveling Salesman Problem.

```
#include <stdio.h>
int matrix[25][25], visited_cities[10], limit, cost = 0;
int tsp(int c)
{
 int count, nearest_city = 999;
 int minimum = 999, temp;
 for(count = 0; count < limit; count++)
 {
 if((matrix[c][count] != 0) && (visited_cities[count] == 0))
 {
 if(matrix[c][count] < minimum)
 {
 minimum = matrix[count][0] + matrix[c][count];
 }
 temp = matrix[c][count];
 nearest_city = count;
 }
 }
 if(minimum != 999)
 {
 cost = cost + temp;
 }
 return nearest_city;
}
```

```
void minimum_cost(int city)
{
 int nearest_city;
 visited_cities[city] = 1;
 printf("%d ", city + 1);
 nearest_city = tsp(city);
 if(nearest_city == 999)
 {
 nearest_city = 0;
 printf("%d", nearest_city + 1);
 cost = cost + matrix[city][nearest_city];
 return;
 }
 minimum_cost(nearest_city);
}
int main()
{
 int i, j;
 printf("Enter Total Number of Cities:\t");
 scanf("%d", &limit);
 printf("\nEnter Cost Matrix\n");
 for(i = 0; i < limit; i++)
 {
 printf("\nEnter %d Elements in Row[%d]\n", limit, i + 1);
 for(j = 0; j < limit; j++)
 {
 scanf("%d", &matrix[i][j]);
```

```
}
visited_cities[i] = 0;
}
printf("\nEntered Cost Matrix\n");
for(i = 0; i < limit; i++)
{
printf("\n");
for(j = 0; j < limit; j++)
{
printf("%d ", matrix[i][j]);
}
}
printf("\n\nPath:\t");
minimum_cost(0);
printf("\n\nMinimum Cost: \t");
printf("%d\n", cost);
return 0;
}
```

OUTPUT: