INDEX

|  |  |  |
| --- | --- | --- |
| Sl.No | Experiment Title | Page No |
| 1.1 | Implementation of BFS in a digraph | 4-6 |
| 1.2 | Implementation of DFS in a digraph | 7-9 |
| 3 | Implementation of topological sort in a digraph | 10-12 |
| 4 | Implementation of Jhonson-Trotter algorithm | 13-15 |
| 5 |  | 16-20 |
| 6 | Delete operations in singly linked list | 21-26 |
| 7.1 | Sorting a singly linked list | 27-29 |
| 7.2 | Reversing a singly linked list | 30-32 |
| 7.3 | Concatenation of two singly linked lists | 32-35 |
| 8 | Implementation of stacks and queues using singly linked list | 36-41 |
| 9 | Implementation of doubly linked list | 42-47 |
| 10 | Implementation of binary search tree | 48-52 |

**PROGRAM 1**

Write program to do the following:

a. Print all the nodes reachable from a given starting node in a digraph using BFS method.

b. Print the DFS traversal of a given graph.

**1.1 BFS Traversal**

**1.1.1 CODE**

#include<stdio.h>

#include<conio.h>

void insert\_rear(int q[],int \*r, int item, int size)

{

if(\*r==size)

printf("Queue overflow!\n");

else

{

\*r=\*r+1;

q[\*r]=item;

}

}

int delete\_front(int q[],int \*r, int \*f)

{

int del\_item=-1;

\*f=\*f+1;

del\_item=q[\*f];

return del\_item;

}

int isEmpty(int q[], int \*r, int \*f)

{

if(\*r==-1 || \*r==\*f)

return 1;

else

return 0;

}

void main()

{

int n,i,j,r=-1,f=-1;

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

scanf("%d",&n);

printf("Enter the adjacency matrix representing the graph:\n");

int graph[n][n];

int vis[n],q[n];

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

{

for(int j=0;j<n;j++)

{

scanf("%d",&graph[i][j]);

}

}

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

{

vis[i]=0;

}

printf(“The BFS travsersal is:\n”);

int k=0;

printf("%d ",k); // print the first node

vis[k]=1; //Make the first node visited

insert\_rear(q,&r,k,n); // Insert the node in the queue

while(isEmpty(q,&r,&f)==0) // if queue is not empty

{

int node=delete\_front(q,&r,&f); //remove node from queue

for(j=0;j<n;j++)

{

if(graph[node][j]==1 && vis[j]==0) /\*if the child of node removed exits and is not visited, make it visited.

1.print the child

2.make the node visited.

3.insert the child into the queue\*/

{

printf("%d ",j);

vis[j]=1;

insert\_rear(q,&r,j,n);

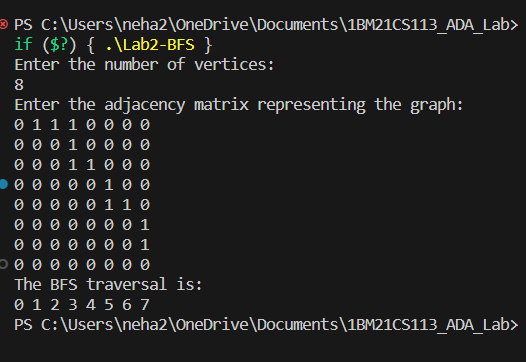
}

}

}

}

**1.1.2 OUTPUT**

****

**1.2 DFS Traversal**

**1.2.1 CODE**

#include<stdio.h>

int graph[20][20];

void DFS(int i,int vis[],int n)

{

int j;

printf("%d ",i); // print the source node

vis[i]=1; // make the source node visited

for(j=0;j<n;j++)

{

if(graph[i][j]==1 && vis[j]==0) // for every adjacent vertex that is not visited

{

DFS(j,vis,n); // recursive call to DFS- because we need to print the nodes depth wise

}

}

}

void main()

{

int n,i,j,top=-1;

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

scanf("%d",&n);

printf("Enter the adjacency matrix representing the graph:\n");

int vis[n],st[n];

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

{

for(int j=0;j<n;j++)

{

scanf("%d",&graph[i][j]);

}

}

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

{

vis[i]=0;

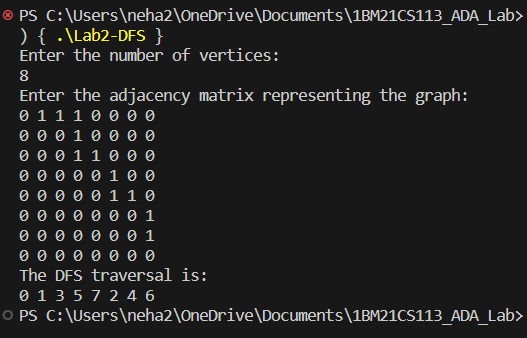
}

printf("The DFS traversal is:\n");

DFS(0,vis,n);

}

**1.2.2 OUTPUT**

****

**PROGRAM 2**

Write program to obtain the Topological ordering of vertices in a given digraph.

**2.1 CODE**

#include<stdio.h>

#include<stdlib.h>

int s[100], j, res[100];

void AdjacencyMatrix(int a[][100], int n)

{

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

{

for(int j=0;j<n;j++)

{

scanf("%d",&a[i][j]);

}

}

return;

}

void dfs(int u, int n, int a[][100])

{

int v;

s[u] = 1;

for (v = 0; v < n ; v++) {

if (a[u][v] == 1 && s[v] == 0) {

dfs(v, n, a);

}

}

j += 1;

res[j] = u; // Store every dead node in the array

}

void topological\_order(int n, int a[][100])

{

int i, u;

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

s[i] = 0;

}

j = 0;

for (u = 0; u < n; u++) {

if (s[u] == 0) {

dfs(u, n, a);

}

}

}

void main() {

int a[100][100], n, i, j;

printf("Enter number of vertices:\n"); /\* READ NUMBER OF VERTICES \*/

scanf("%d", &n);

printf("Enter the adjacency matrix:\n");

AdjacencyMatrix(a, n);

topological\_order(n, a);

printf("The topological sort order is:\n");

for (i = n; i >=1; i--) /\*Inside the array 'res', we are adding the nodes that become dead from first to last.

But topological sort is the reverse order.So we are printing the array backwards.\*/

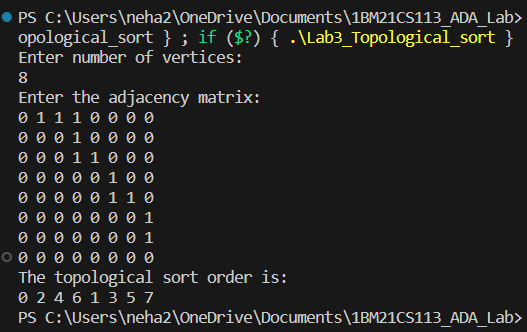
{

printf("%d ", res[i]);

}

}

**2.2 OUTPUT**

****

**PROGRAM 3**

Implement Johnson Trotter algorithm to generate permutations.

**3.1 CODE**

#include <stdio.h>

#define RIGHT\_TO\_LEFT 0

#define LEFT\_TO\_RIGHT 1

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int searchArr(int a[], int n, int mobile)

{

for (int i = 0; i < n; i++) {

if (a[i] == mobile) {

return i + 1;

}

}

return -1; // Mobile not found

}

int getMobile(int a[], int dir[], int n)

{

int mobile\_prev = 0, mobile = 0;

for (int i = 0; i < n; i++) {

// Direction 0 represents RIGHT TO LEFT.

if (dir[a[i] - 1] == RIGHT\_TO\_LEFT && i != 0)

{

if (a[i] > a[i - 1] && a[i] > mobile\_prev)

{

mobile = a[i];

mobile\_prev = mobile;

}

}

if (dir[a[i] - 1] == LEFT\_TO\_RIGHT && i != n - 1)

{

if (a[i] > a[i + 1] && a[i] > mobile\_prev)

{

mobile = a[i];

mobile\_prev = mobile;

}

}

}

if (mobile == 0 && mobile\_prev == 0)

{

return 0; // No mobile element found

} else {

return mobile;

}

}

void printOnePerm(int a[], int dir[], int n)

{

int mobile = getMobile(a, dir, n);

int pos = searchArr(a, n, mobile);

if (dir[a[pos - 1] - 1] == RIGHT\_TO\_LEFT)

{

swap(&a[pos - 1], &a[pos - 2]);

} else if (dir[a[pos - 1] - 1] == LEFT\_TO\_RIGHT)

{

swap(&a[pos], &a[pos - 1]);

}

for (int i = 0; i < n; i++) {

if (a[i] > mobile) {

if (dir[a[i] - 1] == LEFT\_TO\_RIGHT)

{

dir[a[i] - 1] = RIGHT\_TO\_LEFT;

} else if (dir[a[i] - 1] == RIGHT\_TO\_LEFT)

{

dir[a[i] - 1] = LEFT\_TO\_RIGHT;

}

}

}

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

{

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

}

printf("\n");

}

int factorial(int n)

{

int res = 1;

for (int i = 1; i <= n; i++)

{

res = res \* i;

}

return res;

}

void printPermutation(int n)

{

int a[n];

int dir[n];

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

{

a[i] = i + 1;

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

}

printf("\n");

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

{

dir[i] = RIGHT\_TO\_LEFT;

}

for (int i = 1; i < factorial(n); i++)

{

printOnePerm(a, dir, n);

}

}

int main()

{

int n;

printf("Enter the value of n: ");

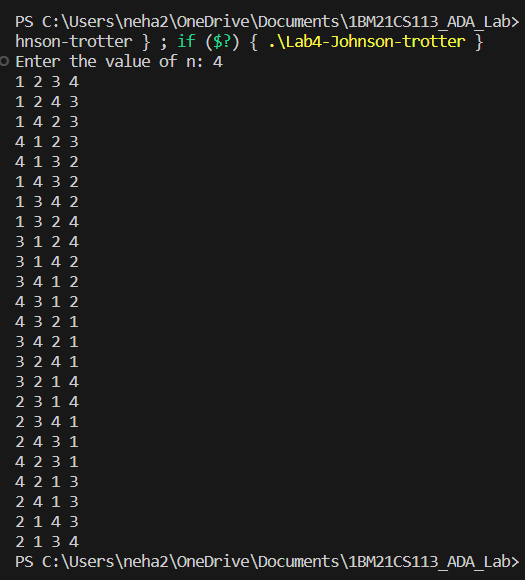
scanf("%d", &n);

printPermutation(n);

return 0;

}

**3.2 OUTPUT**

****

**PROGRAM 4**

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

**4.1 CODE**

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

void conquor(int arr[], int s,int mid,int e)

{

int merged[e-s+1];

int i1=s;

int i2=mid+1;

int x=0;

while(i1<=mid && i2<=e)

{

if(arr[i1]<=arr[i2])

{

merged[x]=arr[i1];

x++;

i1++;

}

else

{

merged[x]=arr[i2];

x++;

i2++;

}

}

while(i1<=mid)

{

merged[x]=arr[i1];

x++;

i1++;

}

while(i2<=e)

{

merged[x]=arr[i2];

x++;

i2++;

}

for(int i=0;i<x;i++)

{

arr[i+s]=merged[i];

}

}

void divide(int arr[], int s, int e)

{

if(s>=e)

return;

int mid=s+(e-s)/2;

divide(arr,s,mid);

divide(arr,mid+1,e);

conquor(arr,s,mid,e);

}

void main()

{

clock\_t st,et;

double ts;

int n= rand()%100+50; // General formala is: rand()%range+min

printf("Size of array:%d\n",n);

int arr[n];

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

{

arr[i]=rand()%100+1;//random number from 1 to 100

}

/\*printf("Original array:\n");

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

{

printf("%d ",arr[i]);

}

printf("\n");\*/

st=clock();

divide(arr,0,n-1);

et=clock();

ts=(double)((et-st)/CLOCKS\_PER\_SEC);

printf("Sorted array:\n");

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

{

printf("%d ",arr[i]);

}

printf("\nThe time taken for merge sort is: %f\n",ts);

}

**PROGRAM 5**

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

**5.1 CODE**

#include<stdio.h>

#include<time.h>

int partition(int arr[], int low, int high) {

int pivot = arr[low]; // Use the first element as the pivot

int i = low;

int j = high + 1;

int temp;

while (1)

{

do

{

i++;

}while (arr[i] <= pivot && i <= high);

do

{

j--;

}while (arr[j] > pivot && j >= low);

if (i >= j) {

break;

}

// Swap arr[i] and arr[j]

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

// Swap arr[low] (pivot) and arr[j]

temp = arr[low];

arr[low] = arr[j];

arr[j] = temp;

return j;

}

void quickSort(int arr[], int low, int high) {

if (low < high) {

int pidx = partition(arr, low, high);

quickSort(arr, low, pidx - 1);

quickSort(arr, pidx + 1, high);

}

}

int main() {

int n;

clock\_t st,et;

double ts;

printf("Enter the number of elements in the array:\n");

scanf("%d", &n);

int arr[n];

printf("Enter %d elements of array:\n", n);

for (int i = 0; i < n; i++) {

scanf("%d", &arr[i]);

}

st=clock();

quickSort(arr, 0, n - 1);

et=clock();

ts=(double)((et-st)/CLOCKS\_PER\_SEC);

printf("The sorted array is:\n");

for (int i = 0; i < n; i++) {

printf("%d ", arr[i]);

}

printf("\n");

printf("Time taken for quick sort:%f\n",ts);

return 0;

}

**PROGRAM 6**

Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

**6.1 CODE**

#include <stdio.h>

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void heapify(int a[], int n, int i)

{

int largest = i;

int l = 2 \* i + 1;

int r = 2 \* i + 2;

if (l < n && a[l] > a[largest])

{

largest = l;

}

if (r < n && a[r] > a[largest])

{

largest = r;

}

if (largest != i)

{

swap(&a[i], &a[largest]);

heapify(a, n, largest);

}

}

void heapSort(int a[], int n)

{

for (int i = n / 2 - 1; i >= 0; i--)

{

heapify(a, n, i);

}

for (int i = n - 1; i > 0; i--)

{

swap(&a[0], &a[i]);

heapify(a, i, 0);

}

}

int main()

{

int n;

printf("Enter the number of elements in the array:\n");

scanf("%d",&n);

int arr[n];

printf("Enter %d elements:\n",n);

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

{

scanf("%d",&arr[i]);

}

printf("Original array: ");

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

{

printf("%d ", arr[i]);

}

printf("\n");

heapSort(arr, n);

printf("Sorted array: ");

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

{

printf("%d ", arr[i]);

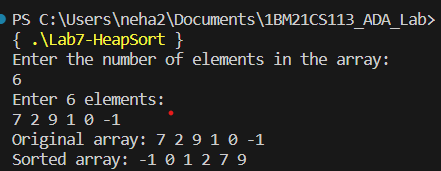
}

printf("\n");

return 0;

}

**6.2 OUTPUT**

****

**PROGRAM 7**

Implement 0/1 Knapsack problem using dynamic programming.

**7.1 CODE**

#include<stdio.h>

void main()

{

int n,cap;

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

scanf("%d",&n);

int wt[n],pft[n];

printf("Enter the weights of %d items:\n",n);

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

{

scanf("%d",&wt[i]);

}

printf("Enter the profit of %d items:\n",n);

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

{

scanf("%d",&pft[i]);

}

printf("Enter the capacity of the sack:\n");

scanf("%d",&cap);

int v[n+1][cap+1];

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

{

for(int j=0;j<=cap;j++)

{

if(i==0 || j==0)

v[i][j]=0;

else if(j<wt[i-1])

{

v[i][j]=v[i-1][j];

}

else if(j>=wt[i-1])

{

if(v[i-1][j]>=(v[i-1][j-wt[i-1]]+pft[i-1]))

v[i][j]=v[i-1][j];

else

v[i][j]=v[i-1][j-wt[i-1]]+pft[i-1];

}

}

}

printf("The table is:\n");

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

{

for(int j=0;j<=cap;j++)

{

printf("%d ",v[i][j]);

}

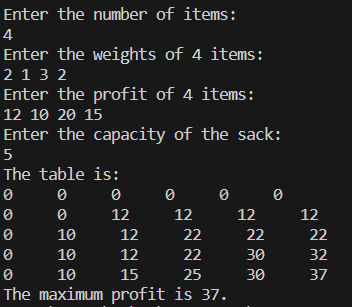
printf("\n");

}

printf("The maximum profit is %d.\n",v[n][cap]);

}

**7.2 OUTPUT**

****

**PROGRAM 8**

Implement All Pair Shortest paths problem using Floyd’s algorithm.

**8.1 CODE**

#include<stdio.h>

void main()

{

int n,i,j,k;

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

scanf("%d",&n);

int graph[n][n];

printf("Enter the cost matrix of the graph:\n");

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

{

for(j=0;j<n;j++)

{

scanf("%d",&graph[i][j]);

}

}

for(k=0;k<n;k++)

{

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

{

for(j=0;j<n;j++)

{

if(graph[i][j]>graph[i][k]+graph[k][j])

graph[i][j]=graph[i][k]+graph[k][j];

}

}

}

printf(“The floyds matrix is:\n”);

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

{

for(j=0;j<n;j++)

{

printf("%d ",graph[i][j]);

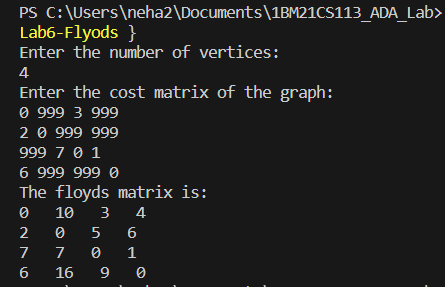
}

printf("\n");

}

}

**8.2 OUTPUT**

****

**PROGRAM 9**

Find Minimum Cost Spanning Tree of a given undirected graph using Prim/Kruskal’s algorithm.

**9.1 Prims algorithm**

**9.1.1 CODE**

#include<stdio.h>

#include<conio.h>

int cost[10][10],vt[10],et[10][10],vis[10],j,n;

int sum=0;

int x=1;

int e=0;

void prims();

void main()

{

int i;

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

}

vis[i]=0;

}

prims();

printf("Edges of spanning tree\n");

for(i=1;i<=e;i++)

{

printf("%d,%d\t",et[i][0],et[i][1]);

}

printf("Weight=%d\n",sum);

}

void prims()

{

int s,min,m,k,u,v;

vt[x]=1;

vis[x]=1;

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

{

j=x;

min=999;

while(j>0)

{

k=vt[j];

for(m=2;m<=n;m++)

{

if(vis[m]==0)

{

if(cost[k][m]<min)

{

min=cost[k][m];

u=k;

v=m;

}

}

}

j--;

}

vt[++x]=v;

et[s][0]=u;

et[s][1]=v;

e++;

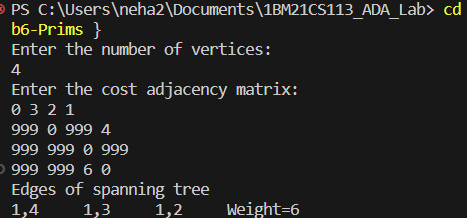
vis[v]=1;

sum=sum+min;

}

}

**9.1.2 OUTPUT**

****

**9.2 Kruskals Algorithm**

**9.2.1 CODE**

#include<stdio.h>

int find(int v,int parent[10])

{

while(parent[v]!=v)

{

v=parent[v];

}

return v;

}

void union1(int i,int j,int parent[10])

{

if(i<j)

parent[j]=i;

else

parent[i]=j;

}

void kruskal(int n,int a[10][10])

{

int count,k,min,sum,i,j,t[10][10],u,v,parent[10];

count=0;

k=0;

sum=0;

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

parent[i]=i;

while(count!=n-1)

{

min=999;

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

{

for(j=0;j<n;j++)

{

if(a[i][j]<min && a[i][j]!=0)

{

min=a[i][j];

u=i;

v=j;

}

}

}

i=find(u,parent);

j=find(v,parent);

if(i!=j)

{

union1(i,j,parent);

t[k][0]=u;

t[k][1]=v;

k++;

count++;

sum=sum+a[u][v];

}

a[u][v]=a[v][u]=999;

}

if(count==n-1)

{

printf("Spanning tree\n");

for(i=0;i<n-1;i++)

{

printf("%d %d\n",t[i][0],t[i][1]);

}

printf("Cost of spanning tree=%d\n",sum);

}

else

printf("Spanning tree does not exist\n");

}

void main()

{

int n,i,j,a[10][10];

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

scanf("%d",&n);

printf("Enter the adjacency matrix:\n");

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

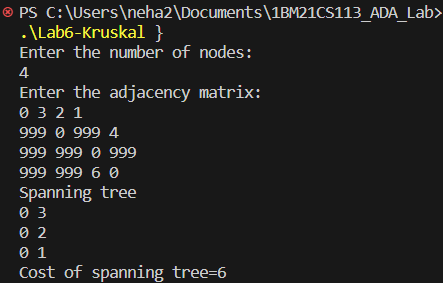
for(j=0;j<n;j++)

scanf("%d",&a[i][j]);

kruskal(n,a);

}

**9.2.2 OUTPUT**

****

**PROGRAM 10**

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.

**10.1 CODE**

#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

int minDistance(int dist[], bool sptSet[],int V)

{

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

void printSolution(int dist[], int V)

{

printf("Vertex \t\t Distance from Source\n");

for (int i = 0; i < V; i++)

printf("%d \t\t\t\t %d\n", i, dist[i]);

}

void dijkstra(int V,int graph[V][V], int src)

{

int dist[V];

bool sptSet[V

for (int i = 0; i < V; i++)

{

dist[i] = INT\_MAX, sptSet[i] = false;

}

dist[src] = 0;

for (int count = 0; count < V - 1; count++)

{

int u = minDistance(dist, sptSet,V);

sptSet[u] = true;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INT\_MAX && dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist,V);

}

void main()

{

int n;

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

scanf("%d",&n);

int graph[n][n];

printf("Enter the cost matrix of the graph:\n");

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

{

for(int j=0;j<n;j++)

{

scanf("%d",&graph[i][j]);

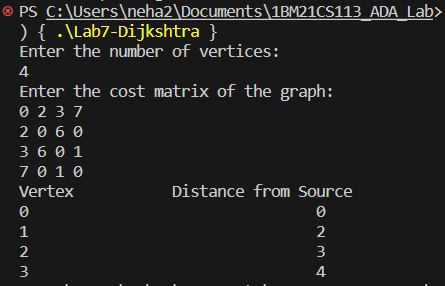
}

}

dijkstra(n,graph, 0);

}

**10.2 OUTPUT**

****

**PROGRAM 11**

Implement “N-Queens Problem” using Backtracking.

**11.1 CODE**

#include <stdio.h>

#include <stdlib.h>

void displayBoard(char board[][4], int n) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

printf("%c ", board[i][j]);

}

printf("\n");

}

}

int isSafe(int row, int col, char board[][4], int n)

{

int duprow = row;

int dupcol = col;

// Checking left side

while (col >= 0)

{

if (board[row][col] == 'Q')

return 0;

col--;

}

row = duprow;

col = dupcol;

// Checking Left Upper diagonal

while (row >= 0 && col >= 0)

{

if (board[row][col] == 'Q')

return 0;

row--;

col--;

}

row = duprow;

col = dupcol;

// Checking Left Lower diagonal

while (row < n && col >= 0)

{

if (board[row][col] == 'Q')

return 0;

row++;

col--;

}

return 1;

}

void solve(int col, char board[][4], int n)

{

if(col == n)

{

displayBoard(board,n);

printf("\n"); //For next combination of board

return;

}

for(int row =0; row<n;row++)

{

if(isSafe(row,col,board,n))

{

board[row][col] ='Q';

solve(col+1,board,n);

board[row][col] = '.'; //Backtracking step

}

}

}

int main()

{

int n;

printf("Enter the dimension of chessBoard:\n");

scanf("%d", &n);

if(n==2 || n==3)

{

printf("No solution exists!\n");

exit(0);

}

char board[n][n];

// Initialising board with No queen

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

{

for (int j = 0; j < n; j++)

{

board[i][j] = '.';

}

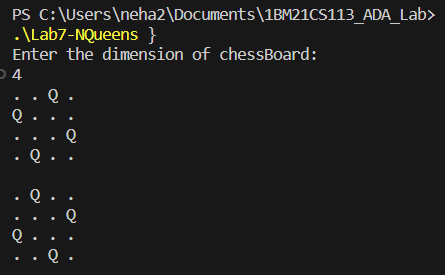
}

solve(0, board,n); // 0th col is called

return 0;

}

**11.2 OUTPUT**

****