**VISVESVARAYATECHNOLOGICALUNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**



**LAB REPORT on**

**Analysis and Design of Algorithms**

***Submitted by***

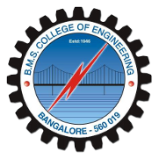
**ARCHIT MEHROTRA (1BM21CS031)**

***in partial fulfillment for the award of the degree of***

# BACHELOROFENGINEERING

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**June-2023 to Sep-2023**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

## Department of Computer Science and Engineering



**CERTIFICATE**

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by

**ARCHIT MEHROTRA (1BM21CS031),** who is bonafide student of **B.M.S. College of**

**Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester June-2023 to Sep-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

|  |  |
| --- | --- |
|  | **Dr. Jyothi S Nayak** |
| Assistant Professor | Professor and Head |
| Department of CSE | Department of CSE |
| BMSCE, Bengaluru | BMSCE, Bengaluru |

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Lab**  **Program No.** | **Program Details** | **Page No.** |
| 1 | Write program to do the following:   1. Print all the nodes reachable from a given starting node in a digraph using BFS method. 2. Check whether a given graph is connected or not using DFS method. | 1-4 |
| 2 | Write program to obtain the Topological ordering of vertices in a given digraph. | 5-8 |
| 3 | Implement Johnson Trotter algorithm to generate permutations. | 9-18 |
| 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. | 19-23 |
| 5 | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 24-28 |
| 6 | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | 29-30 |
| 7 | Implement 0/1 Knapsack problem using dynamic programming. | 31-36 |
| 8  9 | Implement All Pair Shortest paths problem using Floyd’s algorithm. | 37-41 |
| Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm. | 42-49 |
| 10 | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | 50-52 |
| 11 | Implement “N-Queens Problem” using Backtracking. | 53-58 |

**Course Outcome**

|  |  |
| --- | --- |
| CO1 | Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations. |
| CO2 | Apply various design techniques for the given problem. |
| CO3 | Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete |
| CO4 | Design efficient algorithms and conduct practical experiments to solve problems. |

## PROGRAM-1

**Write program to do the following:**

1. **Print all the nodes reachable from a given starting node in a digraph using BFS method.**
2. **Check whether a given graph is connected or not using DFS method.**

### DFS

#include<stdio.h>

#include<conio.h>

void DFS(int);

int a[10][10],vis[10],n;

void DFS(int v)

{

int i;

vis[v]=1;

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

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

{

if(a[v][i]==1 && vis[i]==0)

{

DFS(i);

}

}

}

void check()

{

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

{

if(vis[i]!=1)

{

printf("\nNot connected");

return;

}

}

printf("\nConnected");

}

void main()

{

int i,j;

printf("Enter the number of vertices ");

scanf("%d",&n);

printf("Enter the Adjacency Matrix\n");

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

{

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

{

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

}

}

printf("DFS Traversal\n");

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

{

if(vis[i]==0)

{

DFS(i);

}

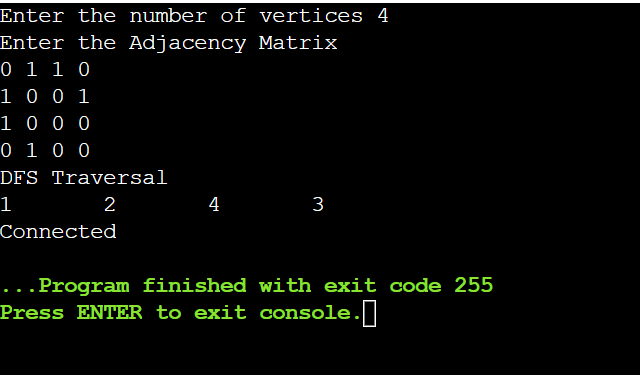
}

check();

getch();

}

### OUTPUT:



### BFS

#include <stdio.h>

#include<conio.h>

int queue[10];

int vis[10];

int matrix[10][10];

int front=1,rear=0;

void push(int a){

queue[rear]=a;

rear++;

}

int pop(){

return queue[front++];

}

void bfs(int n, int size){

for(int i=1;i<=size;i++){

if(matrix[n][i]==1 && vis[i]==0){

push(i);

printf("%d\t",i);

vis[i]=1;

}

}

int m=pop();

bfs(m,size);

};

int main(){

printf("enter the number of vertices\n");

int size;

scanf("%d",&size);

printf("enter the adjacency matrix\n");

for(int i =1;i<=size;i++){

for(int j =1;j<=size;j++){

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

}

}

printf("BFS Traversal\n");

for(int i=1;i<=size;i++){

if(vis[i]==0){

printf("%d\t",i);

vis[i]=1;

push(i);

bfs(i,size);

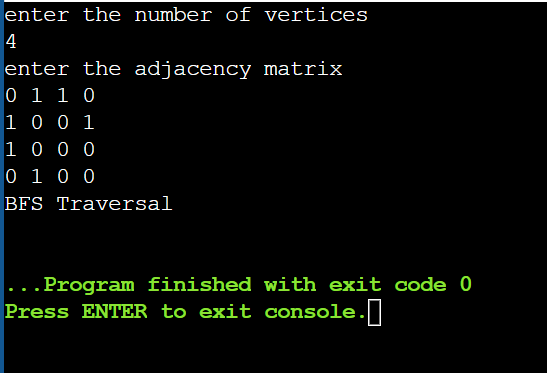
}

}

return 0;

}

**OUTPUT:**



## PROGRAM-2

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

#include<stdio.h>

#include<conio.h>

int a[10][10],vis[10],E[10],n,J=0;

void dfs(int v);

void main(){

int m,c,d;

printf("Enter the number of vertices");

scanf("%d",&n);

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

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

a[i][j]=0;

}

}

printf("Enter the number of edges");

scanf("%d",&m);

for(int i=1;i<=m;i++){

printf("Enter the edges");

scanf("%d%d",&c,&d);

a[c][d]=1;

}

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

if(vis[i]==0){

dfs(i);

}

}

printf("Topological Order\n");

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

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

}

}

void dfs(int v){

vis[v]=1;

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

if(a[v][i]==1 && vis[i]==0){

dfs(i);

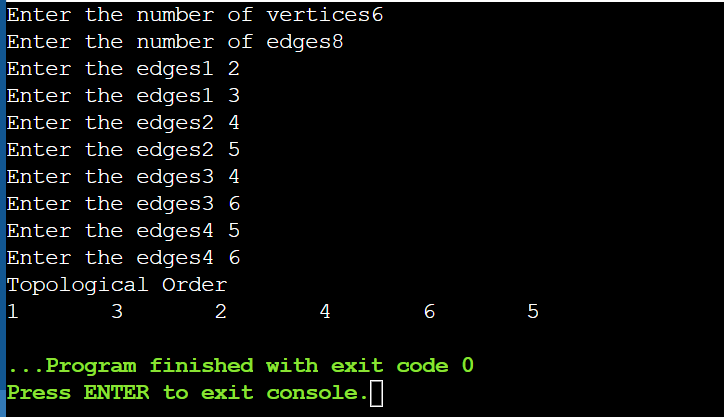
}

}

E[J++]=v;

}

**OUTPUT:**



## PROGRAM-3

**Implement Johnson Trotter algorithm to generate permutations.**

#include<stdio.h>

#include<conio.h>

int i;

int gme(int a[],int b[],int n) // greatest moving elememt

{

int max=0;

int ctr=0;

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

{

if(b[i]==0)

{

if(i!=0)

{

if(a[i] > a[i-1]&&a[i]>max)

{

max =a[i];

ctr++;

}

}

}

else

{

if(i!=n-1)

{

if(a[i]>a[i+1]&&a[i]>max)

{max =a[i];

ctr++;}

}

}

}

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

{

if(a[i]==max)

{max=i;

break;

}

}

if(ctr!=0)

return max;

else

return (n+1);

}

void swap(int g, int a[], int b[], int n)

{

int temp;

if(b[g]==0)

{

temp = a[g];

a[g] = a[g-1];

a[g-1]= temp;

temp = b[g];

b[g] = b[g-1];

b[g-1]= temp;

g=g-1;

}

else

{

temp = a[g];

a[g] = a[g+1];

a[g+1]= temp;

temp = b[g];

b[g] = b[g+1];

b[g+1]= temp;

g=g+1;

}

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

{

if(a[i]>a[g])

{

if(b[i]==0)

b[i]=1;

else

b[i]=0;

}

}

}

void main()

{

int g;

int n;

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

scanf("%d",&n);

int a[n];

int b[n]; //0-left 1- right

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

{

b[i]=0;

a[i]=i+1;

}

int ctr=0;

while(g!=n+1)

{

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

{

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

}

printf("\n");

g= gme(a,b,n);

if(g==n+1)

printf("Task Completed \n");

else

swap(g,a,b,n);

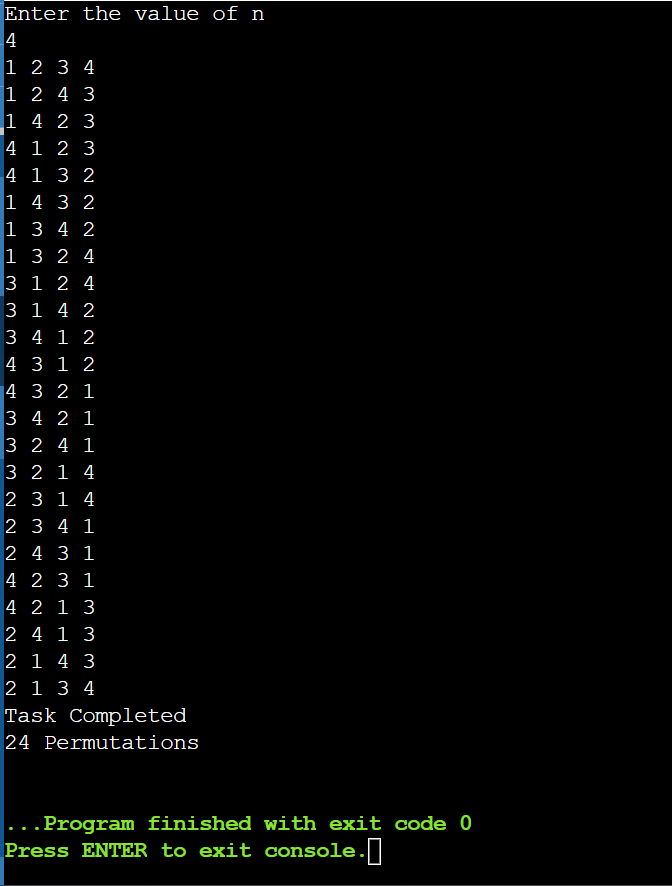
ctr++;

}

printf("%d Permutations \n",ctr);

}

**OUTPUT:**



## PROGRAM-4

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

#include <stdio.h>

void merge(int a[], int low, int high, int mid) {

int c[high - low + 1];

int k = 0;

int i = low;

int j = mid + 1;

while (i <= mid && j <= high) {

if (a[i] < a[j]) {

c[k++] = a[i++];

} else {

c[k++] = a[j++];

}

}

while (i <= mid) {

c[k++] = a[i++];

}

while (j <= high) {

c[k++] = a[j++];

}

for (int t = low; t <= high; t++) {

a[t] = c[t - low];

}

}

void mergesort(int a[], int low, int high) {

if (low < high) {

int mid = (low + high) / 2;

mergesort(a, low, mid);

mergesort(a, mid + 1, high);

merge(a, low, high, mid);

}

}

int main() {

int n;

printf("Enter the size of the array: ");

scanf("%d", &n);

int a[n];

printf("Enter the elements of the list:\n");

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

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

}

mergesort(a, 0, n - 1);

printf("Sorted array: ");

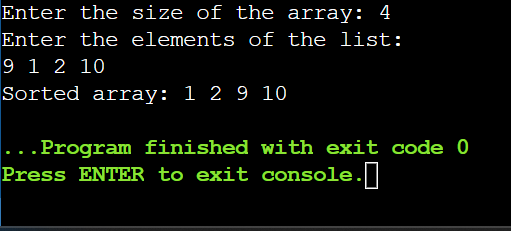
for (int i = 0; i < n; i++) {

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

}

return 0; }

**OUTPUT:**



## PROGRAM-5

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

# include <stdio.h>

void QuickSort(int a[], int low,int high)

{

int mid;

if(low<high)

{

mid=partition(a,low,high);

QuickSort(a,low,mid);

QuickSort(a,mid+1,high);

}

}

int partition(int a[],int low,int high)

{

int i=low+1;

int j=high;

int temp;

int pivot=a[low];

while(i<=j)

{

while(a[i]<pivot)

{

i++;

}

while(a[j]>pivot)

{

j--;

}

if(i<j)

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

}

temp=a[low];

a[low]=a[j];

a[j]=temp;

return j;

}

int main()

{

int n;

printf("Enter the size of the array: ");

scanf("%d", &n);

int a[n];

printf("Enter the elements of the list:\n");

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

{

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

}

QuickSort(a, 0, n-1);

printf("Sorted array: ");

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

{

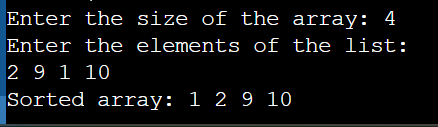
printf("%d ", a[i]);

}

return 0;

}

**OUTPUT:**



## PROGRAM-6

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

#include <stdio.h>

#include<conio.h>

void heapify(int arr[], int i, int size)

{

int left = 2 \* i + 1;

int right = 2 \* i + 2;

int maxIdx = i;

if (left < size && arr[left] > arr[maxIdx])

maxIdx = left;

if (right < size && arr[right] > arr[maxIdx])

maxIdx = right;

if (maxIdx != i)

{

int temp = arr[maxIdx];

arr[maxIdx] = arr[i];

arr[i] = temp;

heapify(arr, maxIdx, size);

}

}

void heapSort(int arr[], int n)

{

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

heapify(arr, i, n);

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

{

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, 0, i);

}

}

int main()

{

int n;

printf("\nEnter the number of elements: ");

scanf("%d", &n);

int arr[n];

printf("Enter array elements: ");

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

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

heapSort(arr, n);

printf("Sorted Array:\n");

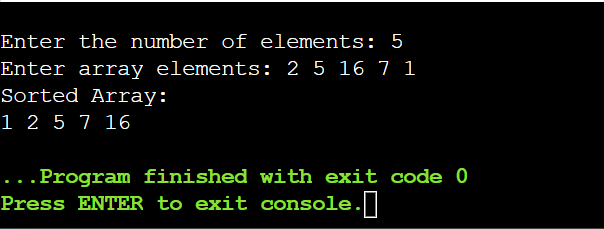
for (int i = 0; i < n; i++)

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

return 0;

}

**OUTPUT:**



## PROGRAM-7

**Implement 0/1 Knapsack problem using dynamic programming.**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int calc(int weight[],int profit[],int wt, int i, int j,int b[][wt])

{

int a;

int l;

if(j>=weight[i])

l=(b[i-1][j-weight[i]])+profit[i];

else

l=(b[i][j-1]);

a=fmax(b[i-1][j],l);

return a;

}

int main ()

{

int n;

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

scanf ("%d", &n);

int item[n+1], weight[n+1],profit[n+1],wt = 0;

int i, j;

printf ("Enter the items \n");

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

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

printf ("Enter the weights \n");

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

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

printf ("Enter the profits \n");

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

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

item[0]=0;

weight[0]=0;

profit[0]=0;

// finding maximum weight

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

if (weight[i] > wt)

wt = weight[i];

int b[n+1][wt+1];

//initialising all values to 0

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

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

b[i][j]=0;

//printing the grid

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

{

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

{

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

printf ("%d \t",b[i][j]);

else

{

b[i][j] = calc(weight,profit,wt+1,i,j,b);

printf ("%d \t", b[i][j]);

}

}

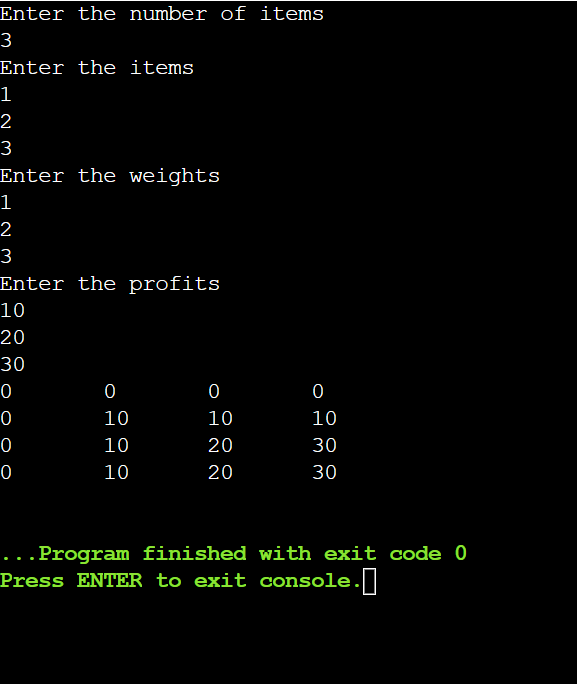
printf ("\n");

}

return 0;

}

**OUTPUT:**



## PROGRAM-8

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

#include<stdio.h>

int min(int a,int b)

{

if(a<b) return(a);

else return(b);

}

void floyds(int p[10][10],int n)

{

int i,j,k;

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

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

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

p[i][j]=min(p[i][j],p[i][k]+p[k][j]);

}

int main()

{

int p[10][10],w,n,e,u,v,i,j;

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

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

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

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

if(i==j)

p[i][j]=0;

else

p[i][j]=999;

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

{

printf("\nEnter the end vertices of edge %d with its weight:\n",i);

scanf("%d %d %d",&u,&v,&w);

p[u][v]=w; }

printf("\n Matrix of input data:\n");

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

{

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

{

if(p[i][j]==999)

printf("INF \t");

else

printf("%d \t",p[i][j]);

}

printf("\n");

}

floyds(p,n);

printf("\n Transitive closure:\n");

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

{

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

if(p[i][j]==999)

printf("INF \t");

else

printf("%d \t",p[i][j]);

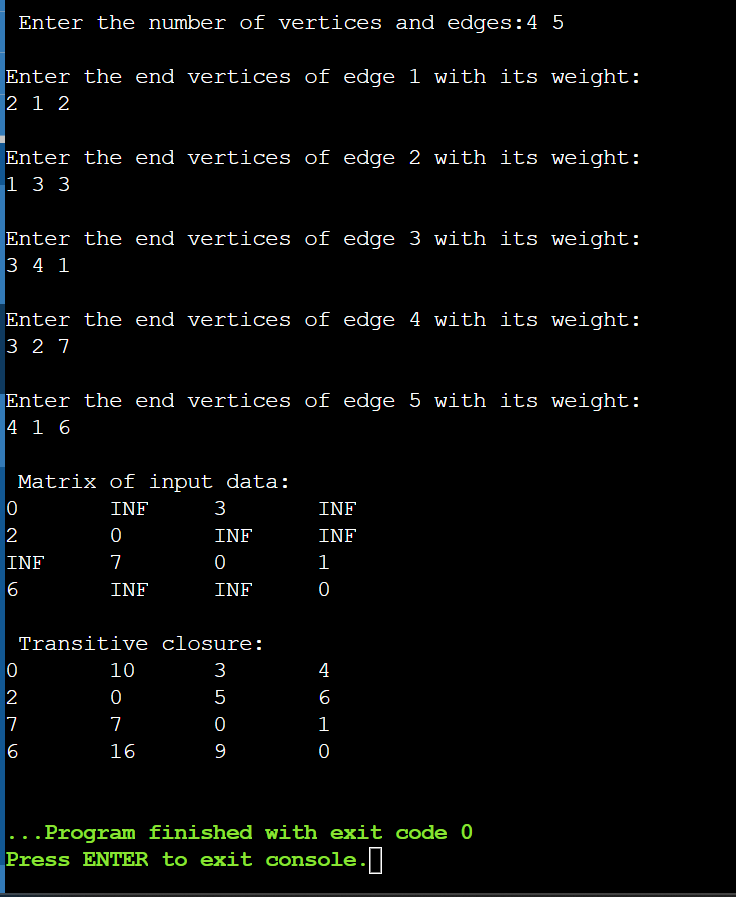
printf("\n");

}

return 0;

}

**OUTPUT:**



## PROGRAM-9

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

#include <stdio.h>

#include <stdbool.h>

#include <string.h>

#define INF 999

#define V 5

int n;

int G[V][V];

int main()

{

int sum=0;

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

scanf("%d", &n);

printf("enter the cost matrix:\n");

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

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

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

int no\_edge;

int selected[V];

memset(selected, false, sizeof(selected));

no\_edge = 0;

selected[0] = true;

int x;

int y;

printf("Edge : Weight\n");

while (no\_edge < V - 1)

{

int min = INF;

x = 0;

y = 0;

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

if (selected[i])

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

if (!selected[j] && G[i][j])

if (min > G[i][j])

{

min = G[i][j];

x = i;

y = j;

}

printf("%d - %d : %d\n", x, y, G[x][y]);

sum+=G[x][y];

selected[y] = true;

no\_edge++;

}

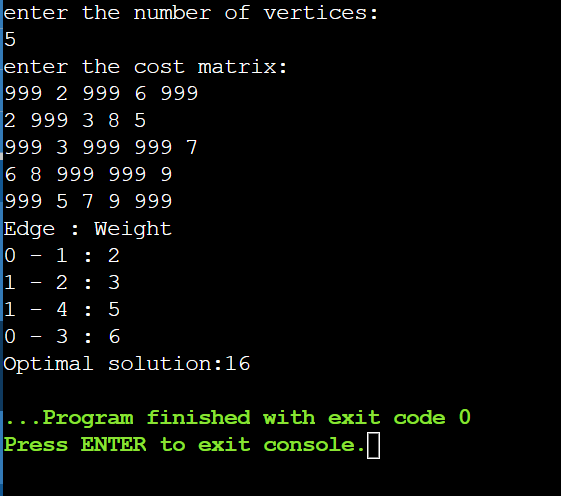
printf("Optimal solution:%d",sum);

return 0;

}

### Prim’s algorithm

**OUTPUT:**



### Kruskal’s algorithm

#include <stdio.h>

int parent[10];

int a[10][10];

int n;

int t[10][2];

int find(int a)

{

while(parent[a]!=a)

a=parent[a];

return a;

}

void unionn(int a,int b)

{

if(a<b)

parent[b]=a;

else

parent[a]=b;

}

void kruskals(int cost[10][10])

{

int count=0;

int k=0;

int u,v;

int i,j,sum=0;

while(count!=n-1)

{

int min=999;

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

parent[i]=i;

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

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

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

{

min=cost[i][j];

u=i;

v=j;

}

i=find(u);

j=find(v);

if(i!=j)

{

t[k][0]=u;

t[k][1]=v;

k++;

count++;

sum+=cost[i][j];

unionn(i,j);

}

cost[u][v]=cost[v][u]=999;

}

printf("the minimal spannning tree is:\n");

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

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

printf("optimal solution: %d",sum);

}

int main()

{

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

scanf("%d",&n);

printf("enter the cost matrix:\n");

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

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

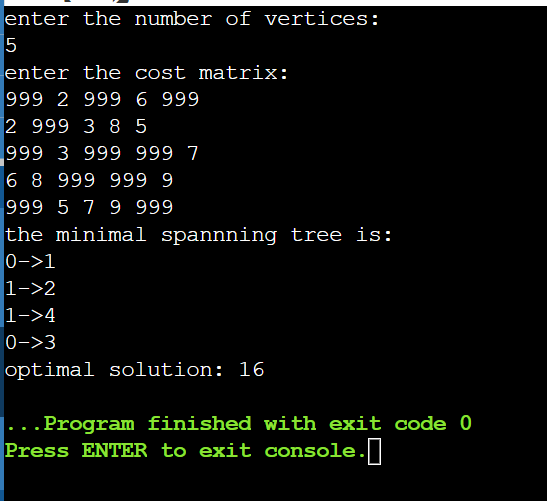
scanf("%d",&a[i][j]);

kruskals(a);

return 0;

}

**OUTPUT:**



## PROGRAM-10

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

#include <stdio.h>

#define INFINITY 999

#define MAX 10

void Dijkstra(int Graph[MAX][MAX], int n, int start)

{

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 (Graph[i][j] == 0)

cost[i][j] = INFINITY;

else

cost[i][j] = Graph[i][j];

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

{

distance[i] = cost[start][i];

pred[i] = start;

visited[i] = 0;

}

distance[start] = 0;

visited[start] = 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 != start)

printf("\nDistance from source to %d: %d", i, distance[i]);

}

int main()

{

int Graph[MAX][MAX], i, j, n, u;

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

scanf("%d",&n);

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

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

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

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

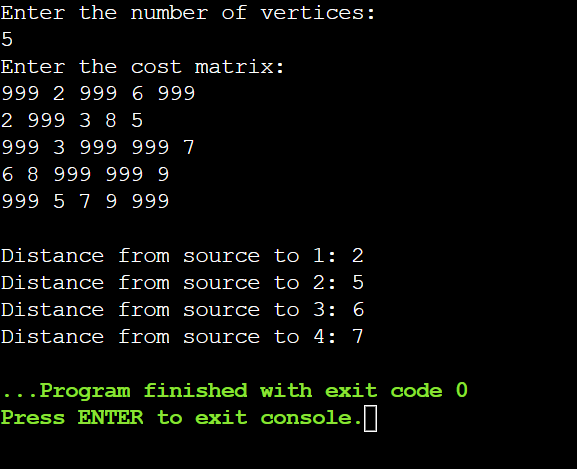
u = 0;

Dijkstra(Graph, n, u);

return 0;

}

**OUTPUT:**



## PROGRAM-11

**Implement “N-Queens Problem” using Backtracking.**

#include <stdio.h> int n, count=0; int isSafe(char board[n][n], int row, int col) #include <stdio.h>

#include <conio.h>

int n, count=0;

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

{

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

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

return 0;

for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)

if (board[i][j] == 'Q')

return 0;

for (int i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++)

if (board[i][j] == 'Q')

return 0;

return 1;

}

void printBoard(char board[][n])

{

printf("\nCHESS BORAD\n");

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

{

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

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

printf("\n");

}

}

void nQueens(char board[n][n], int row)

{

if (row == n)

{

printBoard(board);

count++;

return;

}

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

{

if (isSafe(board, row, j) == 1)

{

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

nQueens(board, row + 1);

board[row][j] = 'X';

}

}

}

int main()

{

printf("Enter the size of the board: ");

scanf("%d", &n);

char board[n][n];

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

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

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

nQueens(board, 0);

printf("\nTotal Possible Solution: %d ",count);

}

{ for (int i = row - 1; i >= 0; i--)

{ if (board[i][col] == 'Q')

{ return 0; } } for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)

{ if (board[i][j] == 'Q') { return 0; } } for (int i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++)

if (board[i][j] == 'Q') { return 0; } } return 1; } void printBoard(char board[][n])

{ printf("\nChess Board\n"); for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++) { printf("%c ", board[i][j]);

}

printf("\n");

} } void nQueens(char board[n][n], int row)

{ if (row == n) printBoard(board); count++; return;

} for (int j = 0; j < n; j++) { if (isSafe(board, row, j) == 1) { board[row][j] = 'Q'; nQueens(board, row + 1); board[row][j] = 'X';

}

} } int main() { printf("Enter the size of the board: "); scanf("%d", &n); char board[n][n]; for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++)

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

} } nQueens(board, 0); printf("\nTotal Possible Solution: %d ",count);

}

**OUTPUT:**

