**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**Analysis and Design of Algorithms**

***Submitted by***

**DHIKSHA RATHIS (1BM21CS055)**

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

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



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

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**June-2023 to September-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 **DHIKSHA RATHIS (1BM21CS055),** who is a 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 September-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.

Radhika A. D.          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:  a. Print all the nodes reachable from a given starting node in a digraph using BFS method.  b. Check whether a given graph is connected or not using DFS method. | 5 |
| 2 | Write program to obtain the Topological ordering of vertices in a given digraph. | 8 |
| 3 | Implement Johnson Trotter algorithm to generate permutations. | 10 |
| 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. | 14 |
| 5 | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 18 |
| 6 | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | 21 |
| 7 | Implement 0/1 Knapsack problem using dynamic programming. | 24 |
| 8 | Implement All Pair Shortest paths problem using Floyd’s algorithm. | 26 |
| 9 | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm. | 28 |
| 10 | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | 32 |
| 11 | Implement “N-Queens Problem” using Backtracking. | 37 |

**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. |

**EXPERIMENT 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. Check whether a given graph is connected or not using DFS method.

a.

#include<stdio.h>

int a[20][20],q[20],visited[20],n,

f=-1,r=-1;

void bfs(int v)

{

int i;

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

{

if(a[v][i] != 0 && visited[i] == 0)

{

r=r+1;

q[r]=i;

visited[i]=1;

printf("%d ",i);

}

}

f=f+1;

if(f<=r)

bfs(q[f]);

}

void main()

{

int v,i,j;

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

scanf("%d",&n);

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

{

visited[i]=0;

}

printf("\nEnter graph data in matrix form:\n");

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

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

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

printf("\nEnter the starting vertex:");

scanf("%d",&v);

f=r=0;

q[r]=v;

printf("\nBFS traversal is:\n");

visited[v]=1;

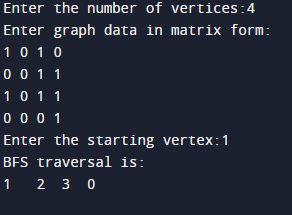
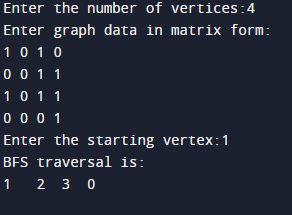
printf("%d ",v);

bfs(v);

if(r != n-1)

printf("\nBFS is not possible");

}

b.

#include<stdio.h>

int a[20][20],reach[20],n;

void dfs(int v){

int i;

reach[v]=1;

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

if(a[v][i]&&!reach[i]){

printf("\n%d->%d",v,i);

dfs(i);

}

}

int main(){

int i,j,count=0;

printf("\nEnter no of vertices : ");

scanf("%d",&n);

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

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

reach[i]=0;

a[i][j]=0;

}

printf("\nEnter adjacency matrix : \n");

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

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

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

dfs(1);

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

if(reach[i])

count++;

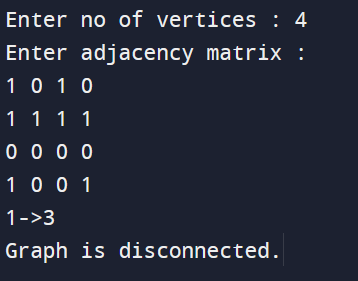
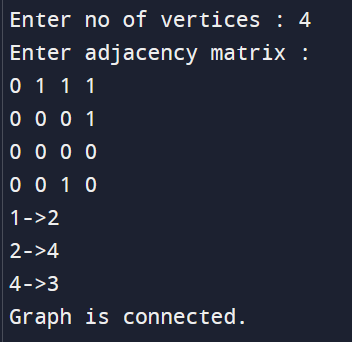
if(count==n)

printf("\nGraph is connected.");

else

printf("\nGraph is disconnected.");

return(0);

 ****

**EXPERIMENT 2:**

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

#include<stdio.h>

void main()

{

int a[20][20],rem[20],ind,n,i,j,flag=0,t=0;

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

scanf("%d",&n);

printf("\nEnter the adjacency matrix ");

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

{

rem[i]=0;

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

{

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

}

}

while(flag==0)

{

flag=1;

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

{

if(rem[i]==0)

{

ind=0;

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

{

if(!(rem[j]==1||a[j][i]==0))

{

ind=1;

break;

}

}

if(ind==0)

{

printf("%s",t==0?"\nTopological ordering is ":"");

rem[i]=1;

printf("%d ",i+1);

flag=0;

t++;

break;

}

}

}

}

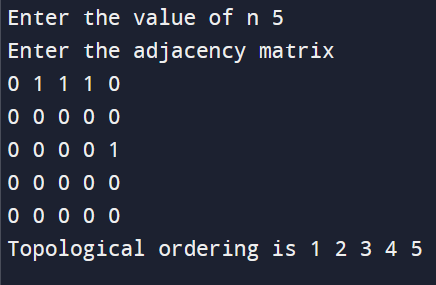
if(t!=n)

{

printf("\nTopological ordering is not possible(it can only be partially ordered)!!");

}

}



**EXPERIMENT 3:**

Implement Johnson Trotter algorithm to generate permutations.

#include <stdio.h>

#include <stdbool.h>

bool LEFT\_TO\_RIGHT = true;

bool RIGHT\_TO\_LEFT = false;

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;

}

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

{

int mobile\_prev = 0, mobile = 0;

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

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;

else

return mobile;

}

void printOnePerm(int a[], bool 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 fact(int n)

{

int res = 1;

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

res = res \* i;

return res;

}

void printPermutation(int n)

{

int a[n];

bool 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 < fact(n); i++)

printOnePerm(a, dir, n);

}

int main()

{

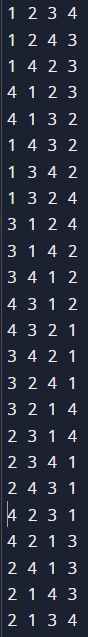
int n = 4;

printPermutation(n);

return 0;

}

OUTPUT:



**EXPERIMENT 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.

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r) {

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void printArray(int A[], int size)

{

int i;

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

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

printf("\n");

}

int main()

{

int arr[200], num, i;

clock\_t start, end;

printf("Enter no of elements in the array\t");

scanf("%d", &num);

printf("Enter the array\n");

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

{

arr[i] = rand();

}

printf("Given array is \t");

printArray(arr, num);

start = clock();

mergeSort(arr, 0, num - 1);

end = clock();

printf("\nSorted array is \n");

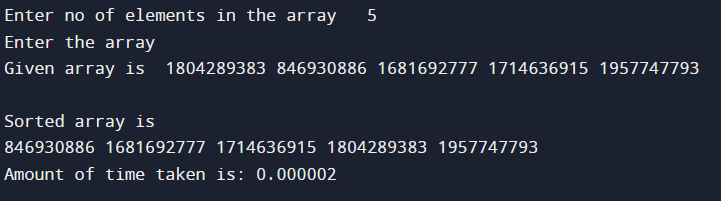
printArray(arr, num);

printf("Amount of time taken is: %f", ((double)(end-start))/CLOCKS\_

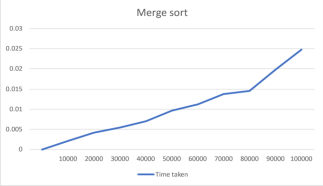
PER\_SEC);

return 0;

OUTPUT:



GRAPH:

****

**EXPERIMENT 5:**

Sort a given set of N integer elements using Quick Sort technique and

compute its time taken.

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

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

{

int key,i,j,temp;

key=a[low];

i=low+1;

j=high;

while(1)

{

while(i<high && key>=a[i])

i++;

while(key<a[j])

j--;

if(i<j)

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

else

{

temp=a[low];

a[low]=a[j];

a[j]=temp;

return j;

}

}

}

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

{

int j;

if(low<high)

{

j=partition(a,low,high);

quicksort(a,low,j-1);

quicksort(a,j+1,high);

}

}

void main()

{

int a[10000],n,t,i;

clock\_t end,start;

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

scanf("%d",&n);

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

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

{

a[i]=rand()%1000;

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

}

start=clock();

for(int j=0;j<5000000;j++)

t=900/900;

quicksort(a,0,n-1);

end=clock();

printf("Sorted array:\n");

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

{

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

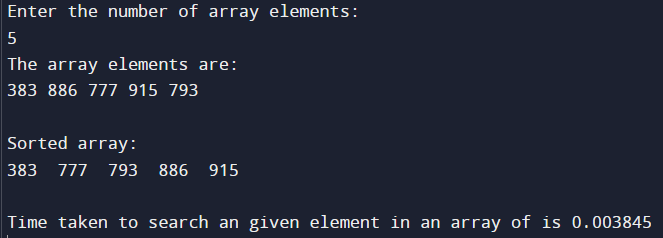
}

printf("Time taken to search an given element in an array of is

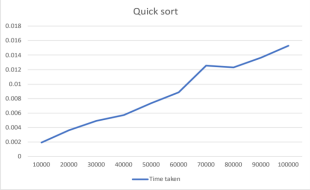
%f\n",(((double)(end-start))/CLOCKS\_PER\_SEC));

}

OUTPUT:



GRAPH:

****

**EXPERIMENT 6:**

Sort a given set of N integer elements using Heap Sort technique and

compute its time taken.

#include <stdio.h>

#include <time.h>

clock\_t start, end;

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

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

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

{

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < N && arr[left] > arr[largest])

largest = left;

if (right < N && arr[right] > arr[largest])

largest = right;

if (largest != i) {

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

heapify(arr, N, largest);

}

}

void heapSort(int arr[], int N)

{

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

heapify(arr, N, i);

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

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

heapify(arr, i, 0);

}

}

void printArray(int arr[], int N)

{

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

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

printf("\n");

}

int main()

{

int arr[1000];

int N;

printf("Enter number of elements:");

scanf("%d",&N);

printf("\nEnter elements:");

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

{arr[i]=rand();}

start=clock();

heapSort(arr, N);

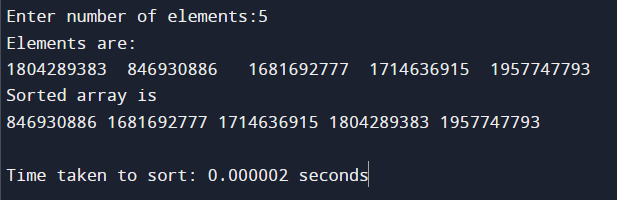
end=clock();

printf("Sorted array is\n");

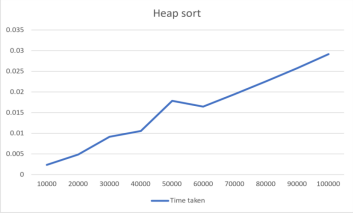
printArray(arr, N);

printf("\n Time taken to sort: %f seconds",(double)(end-start)/CLOCKS\_PER\_SEC);

}

OUTPUT: 

GRAPH:

****

**EXPERIMENT 7:**

Implement 0/1 Knapsack problem using dynamic programming.

#include <stdio.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int knapSack(int W, int wt[], int val[], int n, int selected[]) {

int i, w;

int K[n + 1][W + 1];

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

for (w = 0; w <= W; w++) {

if (i == 0 || w == 0)

K[i][w] = 0;

else if (wt[i - 1] <= w)

K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);

else

K[i][w] = K[i - 1][w];

}

}

int res = K[n][W];

w = W;

for (i = n; i > 0 && res > 0; i--) {

if (res == K[i - 1][w])

continue;

else {

selected[i - 1] = 1;

res = res - val[i - 1];

w = w - wt[i - 1];

}

}

return K[n][W];

}

int main() {

int n, W, i;

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

scanf("%d", &n);

int val[n], wt[n], selected[n];

printf("Enter the values of the items: ");

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

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

printf("Enter the weights of the items: ");

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

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

printf("Enter the capacity of the knapsack: ");

scanf("%d", &W);

int max\_profit = knapSack(W, wt, val, n, selected);

printf("The maximum profit is %d\n", max\_profit);

printf("The objects selected for the optimal solution are: ");

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

if (selected[i])

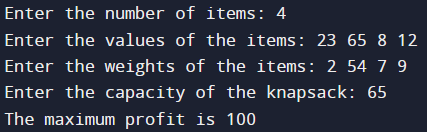
printf("%d ", i + 1);

}

return 0;

}

OUTPUT:



**EXPERIMENT 8:**

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

#include<stdio.h>

#include<conio.h>

int a[10][10],n;

void floyd();

int min(int,int);

void main()

{

int i,j;

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

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]);

}

}

floyd();

}

void floyd()

{

int i,j,k;

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

{

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

{

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

{

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

}

}

}

printf("All pair of shortest path matrix is:\n");

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

{

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

{

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

}

printf("\n\n");

}

}

int min(int x,int y)

{

if(x<y)

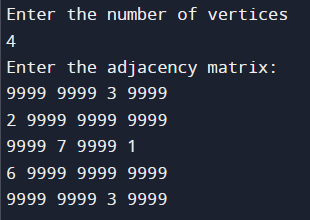
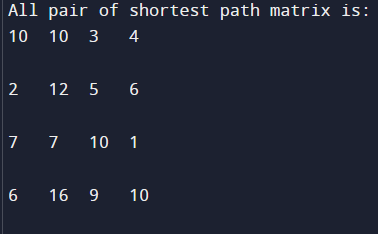
return x;

else

return y;

}

OUTPUT:

**EXPERIMENT 9:**

Find Minimum Cost Spanning Tree of a given undirected graph using

Prim’s and Kruskal’s algorithm.

1. Prim’s Algorithm

#include<stdio.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++;

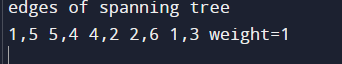
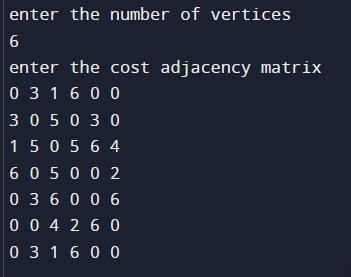
vis[v]=1;

sum=sum+min;

}

}

OUTPUT:



1. Kruskal’s Algorithm

#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++)

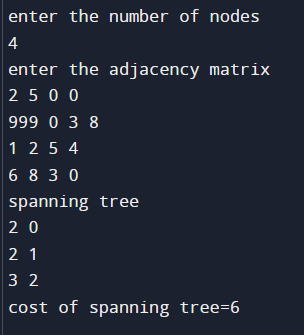
for(j=0;j<n;j++)

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

kruskal(n,a);

}

OUTPUT:



**EXPERIMENT 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 9999

#define MAX 10

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

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;

// Creating cost matrix

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++;

}

// Printing the distance

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 number of nodes:");

scanf("%d",&n);

printf("Enter matrix:");

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

{

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

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

}

}

printf("Enter initial vertex:");

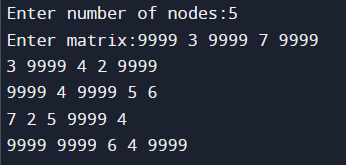
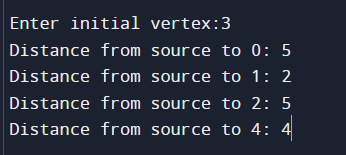
scanf("%d",&u);

4

Dijkstra(Graph, n, u);

return 0;

OUTPUT:

**EXPERIMENT 11:**

Implement “N-Queens Problem” using Backtracking.

#include<stdio.h>

#include<math.h>

int board[20],count;

int main()

{

int n,i,j;

void queen(int row,int n);

printf(" - N Queens Problem Using Backtracking -");

printf("\n\nEnter number of Queens:");

scanf("%d",&n);

queen(1,n);

return 0;

}

void print(int n)

{

int i,j;

printf("\n\nSolution %d:\n\n",++count);

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

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

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

{

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

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

{

if(board[i]==j)

printf("\tQ");

else

printf("\t-");

}

}

}

If no conflict for desired postion returns 1 otherwise returns 0\*/

int place(int row,int column)

{

int i;

for(i=1;i<=row-1;++i)

{

if(board[i]==column)

return 0;

else

if(abs(board[i]-column)==abs(

i-row))

return 0;

}

return 1; }

void queen(int row,int n)

{

int column;

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

{

if(place(row,column))

{

board[row]=column;

if(row==n)

print(n);

else

queen(row+1,n);

}

}

}

OUTPUT:

