# Narula Institute of Technology

**NAAC Accredited Autonomous Institution Affiliated by MAKAUT**



|  |  |
| --- | --- |
| **Paper Name:** | **Design And Analysis Of Algorithms Lab** |
| **Paper Code:** | **PCC-CS492** |
| **Faculty Name:** | **Dr. Papri Ghosh** |

**Lab Experiment Report**

**Name:** PRADHUMAN KUMAR

**Course:** B-Tech

**Semester:** 4TH

**Department:** CSE(AI&ML)

**University Roll No.:** 431022110087

**YEAR:** 2022-2023

**Session:** 2022-2025



**Index**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Program Statement** | **Page No.** |
| 1 | Write a C Program to add two numbers. | 3 |
| 2 | Write a C Program to check a number is odd or even. | 3 |
| 3 | Write a C Program to check a number is prime or not. | 4 |
| 4 | Write a C Program to calculate Factorial of a number. | 5 |
| 5 | Write a C Program to print corresponding ASCII value. | 6 |
| 6 | Write a C Program for Fractional Knapsack problem. | 7 |
| 7 | Write a C Program for Graph coloring. | 10 |
| 8 | Write a C Program for BFS. | 13 |
| 9 | Write a C Program for DFS | 16 |
| 10 | Write a C Program for spanning tree using Prisms algorithm. | 20 |
| 11 | Write a C Program for spanning tree using Kruskals algorithm. | 23 |
| 12 | Write a C Program for Matrix multiplication. | 26 |
| 13 | Write a C Program for Dijkstra algorithm. | 29 |
| 14 | Write a C Program for Floyd- Warshall algorithm**.** | 32 |
| 15 | Write a C Program for N-Queen problem**.** | 34 |
| 16 | Write a C Program for Cycle formation. | 37 |
|  |  |  |
|  |  |  |

**1. Write a C Program to add two numbers.**

**Program Code:**

#include <stdio.h>

int main()

{

int A, B, sum = 0;

printf("Enter two numbers A and B : \n");

scanf("%d%d", &A, &B);

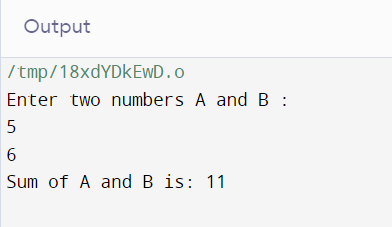
sum = A + B;

printf("Sum of A and B is: %d", sum);

return 0;

}

**Sample Output:**

****

**2.Write a C Program to check a number is odd or even.**

**Program Code:**

#include<stdio.h>

int main()

{

int a;

printf("enter number:\n");

scanf("%d",&a);

if(a%2==0)

printf("%d is Even number",a);

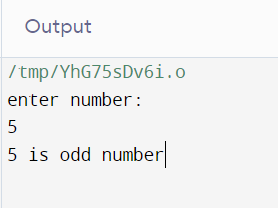
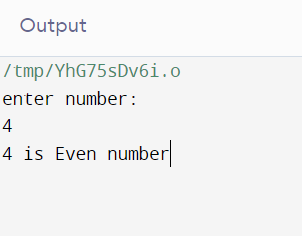
else

printf("%d is odd number",a);

return 0;

}

**Sample Output:**

** **

**3.Write a C Program to check a number is prime or not.**

**Program Code:**

#include<stdio.h>

int main()

{

int n,m;

m=0;

printf("enter number:\n");

scanf("%d",&n);

if(n<=2)

printf("%d not prime number",n);

else

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

{

if(n%i==0)

m++;

}

if(m==2)

printf("%d prime number",n);

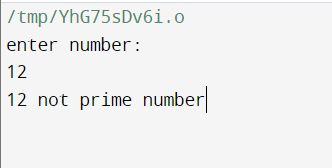
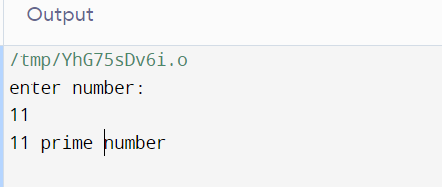
else if(m>2)

printf("%d not prime number",n);

return 0;

}

**Sample Output:**

****

**4.Write a C Program to calculate Factorial of a number.**

**Program Code:**

#include<stdio.h>

int main()

{

int n,i=1,fact=1;

printf("Enter a number to find factorial: ");

scanf("%d",&n);

while(n>1)

{

fact=fact\*n;

n--;

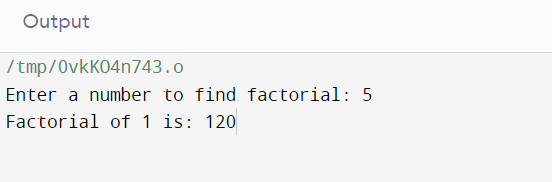
}

printf("Factorial of %d is: %d",n,fact);

return 0;

}

**Sample Output:**

****

**5.Write a C Program to print corresponding ASCII value.**

**Program Code:**

#include <stdio.h>

int main()

{

char c;

printf("Enter a character: ");

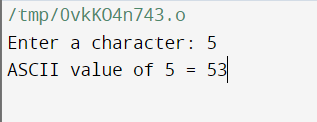
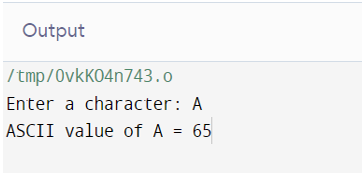
scanf("%c", &c);

printf("ASCII value of %c = %d", c, c);

return 0;

}

**Sample Output:**

****

**6. Write a C Program for Fractional Knapsack problem.**

**Program Code:**

#include <stdio.h>

void main()

{

int capacity, no\_items, cur\_weight, item;

int used[10];

float total\_profit;

int i;

int weight[10];

int value[10];

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

scanf("%d", &capacity);

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

scanf("%d", &no\_items);

printf("Enter the weight and value of %d item:\n", no\_items);

for (i = 0; i < no\_items; i++)

{

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

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

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

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

}

for (i = 0; i < no\_items; ++i)

used[i] = 0;

cur\_weight = capacity;

while (cur\_weight > 0)

{

item = -1;

for (i = 0; i < no\_items; ++i)

if ((used[i] == 0) &&

((item == -1) || ((float) value[i] / weight[i] > (float) value[item] / weight[item])))

item = i;

used[item] = 1;

cur\_weight -= weight[item];

total\_profit += value[item];

if (cur\_weight >= 0)

printf("Added object %d (%d Rs., %dKg) completely in the bag. Space left: %d.\n", item + 1, value[item], weight[item], cur\_weight);

else

{

int item\_percent = (int) ((1 + (float) cur\_weight / weight[item]) \* 100);

printf("Added %d%% (%d Rs., %dKg) of object %d in the bag.\n", item\_percent, value[item], weight[item], item + 1);

total\_profit -= value[item];

total\_profit += (1 + (float)cur\_weight / weight[item]) \* value[item];

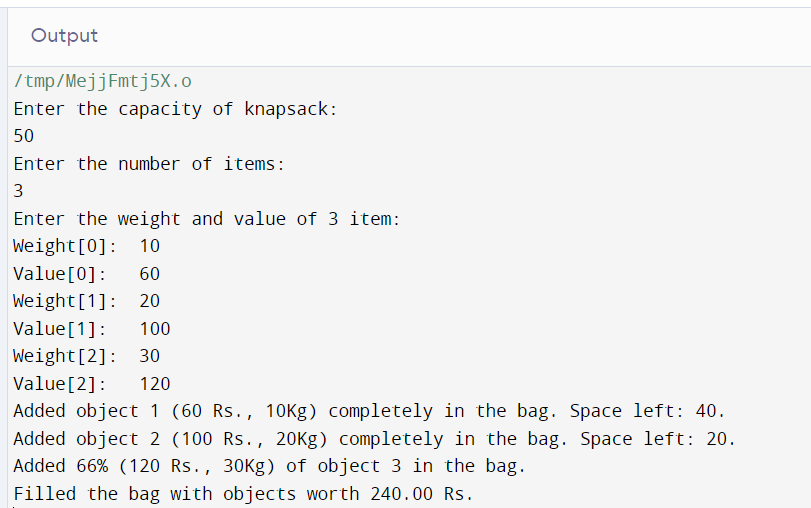
}

}

printf("Filled the bag with objects worth %.2f Rs.\n", total\_profit);

}

**Sample Output:**

****

**7. Write a C Program for Graph coloring.**

**Program Code:**

#include <stdbool.h>

#include <stdio.h>

#define V 4

void printSolution(int color[]);

bool isSafe(bool graph[V][V], int color[])

{

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

for (int j = i + 1; j < V; j++)

if (graph[i][j] && color[j] == color[i])

return false;

return true;

}

bool graphColoring(bool graph[V][V], int m, int i,

int color[V])

{

if (i == V) {

if (isSafe(graph, color)) {

// Print the solution

printSolution(color);

return true;

}

return false;

}

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

color[i] = j;

if (graphColoring(graph, m, i + 1, color))

return true;

color[i] = 0;

}

return false;

}

void printSolution(int color[])

{

printf("Solution Exists:"

" Following are the assigned colors \n");

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

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

printf("\n");

}

int main()

{

bool graph[V][V] = {

{ 0, 1, 1, 1 },

{ 1, 0, 1, 0 },

{ 1, 1, 0, 1 },

{ 1, 0, 1, 0 },

};

int m = 3;

int color[V];

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

color[i] = 0;

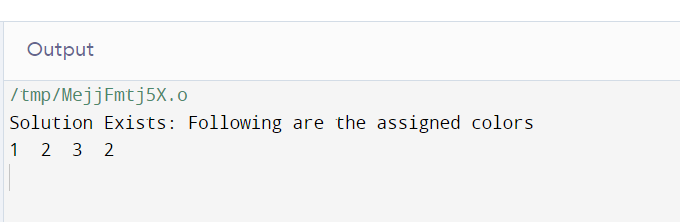
if (!graphColoring(graph, m, 0, color))

printf("Solution does not exist");

return 0;

}

**Sample Output:**

****

**8 Write a C Program for BFS.**

**Program Code:**

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 50

typedef struct Graph\_t {

int V;

bool adj[MAX\_VERTICES][MAX\_VERTICES];

} Graph;

Graph\* Graph\_create(int V)

{

Graph\* g = malloc(sizeof(Graph));

g->V = V;

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

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

g->adj[i][j] = false;

}

}

return g;

}

void Graph\_destroy(Graph\* g) {

free(g);

}

void Graph\_addEdge(Graph\* g, int v, int w)

{

// Add w to v’s list.

g->adj[v][w] = true;

}

void Graph\_BFS(Graph\* g, int s)

{

bool visited[MAX\_VERTICES];

for (int i = 0; i < g->V; i++) {

visited[i] = false;

}

int queue[MAX\_VERTICES];

int front = 0, rear = 0;

visited[s] = true;

queue[rear++] = s;

while (front != rear) {

s = queue[front++];

printf("%d ", s);

for (int adjacent = 0; adjacent < g->V;

adjacent++) {

if (g->adj[s][adjacent] && !visited[adjacent]) {

visited[adjacent] = true;

queue[rear++] = adjacent;

}

}

}

}

int main()

{

Graph\* g = Graph\_create(4);

Graph\_addEdge(g, 0, 1);

Graph\_addEdge(g, 0, 2);

Graph\_addEdge(g, 1, 2);

Graph\_addEdge(g, 2, 0);

Graph\_addEdge(g, 2, 3);

Graph\_addEdge(g, 3, 3);

printf("Following is Breadth First Traversal "

"(starting from vertex 2) \n");

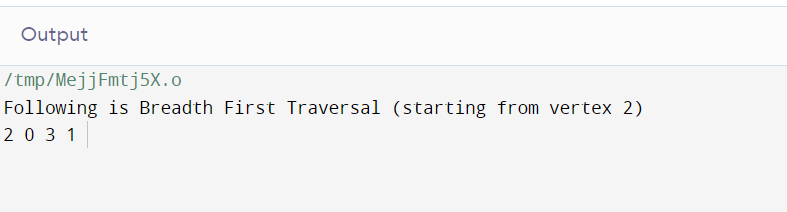
Graph\_BFS(g, 2);

Graph\_destroy(g);

return 0;

}

**Sample Output:**

****

**9. Write a C Program for DFS**

**Program Code:**

#include <stdio.h>

#include <stdlib.h>

struct node {

int vertex;

struct node\* next;

};

struct node\* createNode(int v);

struct Graph {

int numVertices;

int\* visited;

struct node\*\* adjLists;

};

void DFS(struct Graph\* graph, int vertex) {

struct node\* adjList = graph->adjLists[vertex];

struct node\* temp = adjList;

graph->visited[vertex] = 1;

printf("Visited %d \n", vertex);

while (temp != NULL) {

int connectedVertex = temp->vertex;

if (graph->visited[connectedVertex] == 0) {

DFS(graph, connectedVertex);

}

temp = temp->next;

}

}

struct node\* createNode(int v) {

struct node\* newNode = malloc(sizeof(struct node));

newNode->vertex = v;

newNode->next = NULL;

return newNode;

}

struct Graph\* createGraph(int vertices) {

struct Graph\* graph = malloc(sizeof(struct Graph));

graph->numVertices = vertices;

graph->adjLists = malloc(vertices \* sizeof(struct node\*));

graph->visited = malloc(vertices \* sizeof(int));

int i;

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

graph->adjLists[i] = NULL;

graph->visited[i] = 0;

}

return graph;

}

void addEdge(struct Graph\* graph, int src, int dest) {

struct node\* newNode = createNode(dest);

newNode->next = graph->adjLists[src];

graph->adjLists[src] = newNode;

newNode = createNode(src);

newNode->next = graph->adjLists[dest];

graph->adjLists[dest] = newNode;

}

void printGraph(struct Graph\* graph) {

int v;

for (v = 0; v < graph->numVertices; v++) {

struct node\* temp = graph->adjLists[v];

printf("\n Adjacency list of vertex %d\n ", v);

while (temp) {

printf("%d -> ", temp->vertex);

temp = temp->next;

}

printf("\n");

}

}

int main() {

struct Graph\* graph = createGraph(4);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 2);

addEdge(graph, 2, 3);

printGraph(graph);

DFS(graph, 2);

return 0;

}

**Sample Output:**

****

**10. Write a C Program for spanning tree using Prisms algorithm.**

**Program Code:**

#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

#define V 5

int minKey(int key[], bool mstSet[])

{

int min = INT\_MAX, min\_index;

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

if (mstSet[v] == false && key[v] < min)

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

return min\_index;

}

int printMST(int parent[], int graph[V][V])

{

printf("Edge \tWeight\n");

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

printf("%d - %d \t%d \n", parent[i], i,

graph[i][parent[i]]);

}

void primMST(int graph[V][V])

{

int parent[V];

int key[V];

bool mstSet[V];

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

key[i] = INT\_MAX, mstSet[i] = false;

key[0] = 0;

parent[0] = -1;

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

int u = minKey(key, mstSet);

mstSet[u] = true;

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

if (graph[u][v] && mstSet[v] == false

&& graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

printMST(parent, graph);

}

int main()

{

int graph[V][V] = { { 0, 2, 0, 6, 0 },

{ 2, 0, 3, 8, 5 },

{ 0, 3, 0, 0, 7 },

{ 6, 8, 0, 0, 9 },

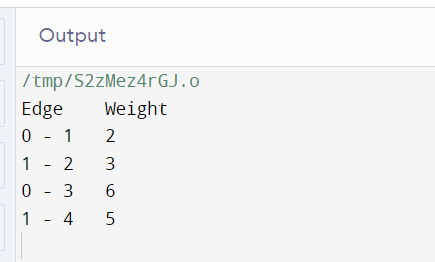
{ 0, 5, 7, 9, 0 } };

primMST(graph);

return 0;

}

**Sample Output:**

****

**11. Write a C Program for spanning tree using Kruskals algorithm.**

**Program Code:**

#include <stdio.h>

#include <stdlib.h>

int comparator(const void\* p1, const void\* p2)

{

const int(\*x)[3] = p1;

const int(\*y)[3] = p2;

return (\*x)[2] - (\*y)[2];

}

void makeSet(int parent[], int rank[], int n)

{

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

parent[i] = i;

rank[i] = 0;

}

}

int findParent(int parent[], int component)

{

if (parent[component] == component)

return component;

return parent[component]

= findParent(parent, parent[component]);

}

void unionSet(int u, int v, int parent[], int rank[], int n)

{

u = findParent(parent, u);

v = findParent(parent, v);

if (rank[u] < rank[v]) {

parent[u] = v;

}

else if (rank[u] > rank[v]) {

parent[v] = u;

}

else {

parent[v] = u;

rank[u]++;

}

}

void kruskalAlgo(int n, int edge[n][3])

{

qsort(edge, n, sizeof(edge[0]), comparator);

int parent[n];

int rank[n];

makeSet(parent, rank, n);

int minCost = 0;

printf(

"Following are the edges in the constructed MST\n");

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

int v1 = findParent(parent, edge[i][0]);

int v2 = findParent(parent, edge[i][1]);

int wt = edge[i][2];

if (v1 != v2) {

unionSet(v1, v2, parent, rank, n);

minCost += wt;

printf("%d -- %d == %d\n", edge[i][0],

edge[i][1], wt);

}

}

printf("Minimum Cost Spanning Tree: %d\n", minCost);

}

int main()

{

int edge[5][3] = { { 0, 1, 10 },

{ 0, 2, 6 },

{ 0, 3, 5 },

{ 1, 3, 15 },

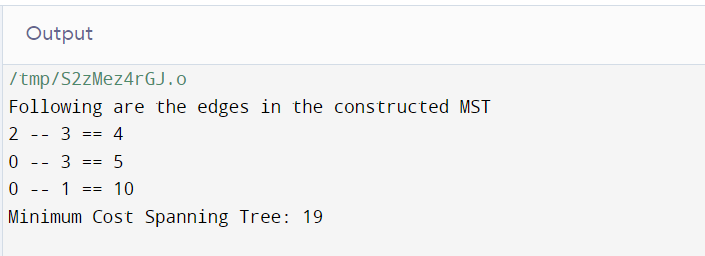
{ 2, 3, 4 } };

kruskalAlgo(5, edge);

return 0;

}

**Sample Output:**

****

**12. Write a C Program for Matrix multiplication.**

**Program Code:**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int a[10][10], b[10][10], mul[10][10], r, c, i, j, k;

system("cls");

printf("enter the number of row=");

scanf("%d", &r);

printf("enter the number of column=");

scanf("%d", &c);

printf("enter the first matrix element=\n");

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

{

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

{

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

}

}

printf("enter the second matrix element=\n");

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

{

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

{

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

}

}

printf("multiply of the matrix=\n");

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

{

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

{

mul[i][j] = 0;

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

{

mul[i][j] += a[i][k] \* b[k][j];

}

}

}

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

{

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

{

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

}

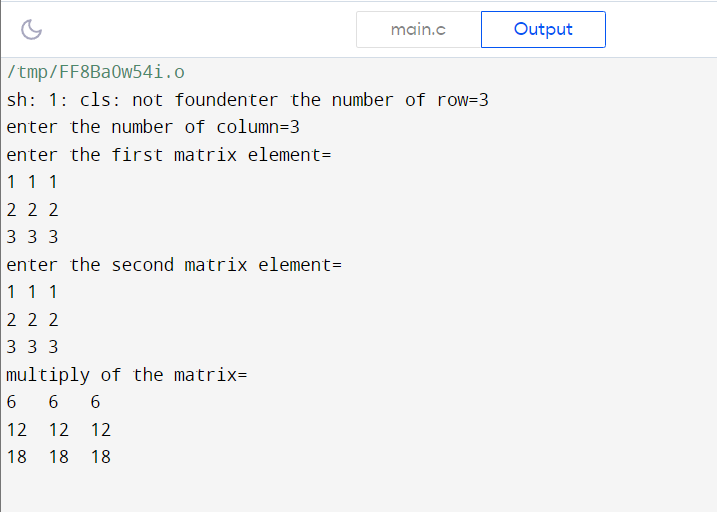
printf("\n");

}

return 0;

}

**Sample Output:**

****

**13. Write a C Program for Dijkstra algorithm.**

**Program Code:**

#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

#define V 9

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

{

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[])

{

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 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);

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);

}

int main()

{

int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

{ 4, 0, 8, 0, 0, 0, 0, 11, 0 },

{ 0, 8, 0, 7, 0, 4, 0, 0, 2 },

{ 0, 0, 7, 0, 9, 14, 0, 0, 0 },

{ 0, 0, 0, 9, 0, 10, 0, 0, 0 },

{ 0, 0, 4, 14, 10, 0, 2, 0, 0 },

{ 0, 0, 0, 0, 0, 2, 0, 1, 6 },

{ 8, 11, 0, 0, 0, 0, 1, 0, 7 },

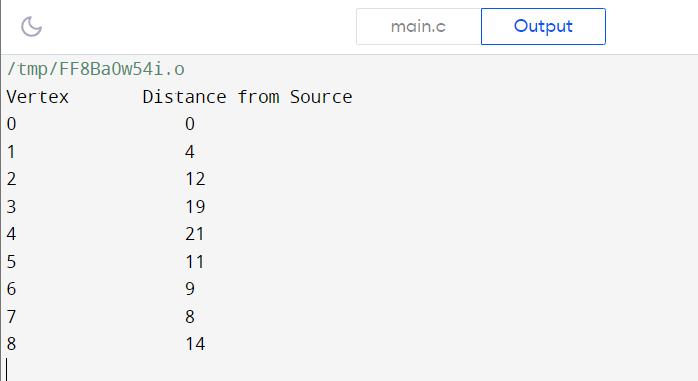
{ 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

dijkstra(graph, 0);

return 0;

}

**Sample Output:**

****

**14. Write a C Program for Floyd- Warshall algorithm.**

**Program Code:**

#include <stdio.h>

#define nV 4

#define INF 999

void printMatrix(int matrix[][nV]);

void floydWarshall(int graph[][nV]) {

int matrix[nV][nV], i, j, k;

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

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

matrix[i][j] = graph[i][j];

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

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

for (j = 0; j < nV; j++) {

if (matrix[i][k] + matrix[k][j] < matrix[i][j])

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

}

}

}

printMatrix(matrix);

}

void printMatrix(int matrix[][nV]) {

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

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

if (matrix[i][j] == INF)

printf("%4s", "INF");

else

printf("%4d", matrix[i][j]);

}

printf("\n");

}

}

int main() {

int graph[nV][nV] = {{0, 3, INF, 5},

{2, 0, INF, 4},

{INF, 1, 0, INF},

{INF, INF, 2, 0}};

floydWarshall(graph);

}

**Sample Output:**

****

**15. Write a C Program for N-Queen problem.**

**Program Code:**

#define N 4

#include <stdbool.h>

#include <stdio.h>

void printSolution(int board[N][N])

{

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

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

if(board[i][j])

printf("Q ");

else

printf(". ");

}

printf("\n");

}

}

bool isSafe(int board[N][N], int row, int col)

{

int i, j;

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

if (board[row][i])

return false;

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

if (board[i][j])

return false;

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

if (board[i][j])

return false;

return true;

}

bool solveNQUtil(int board[N][N], int col)

{

if (col >= N)

return true;

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

if (isSafe(board, i, col)) {

board[i][col] = 1;

if (solveNQUtil(board, col + 1))

return true;

board[i][col] = 0; // BACKTRACK

}

}

return false;

}

bool solveNQ()

{

int board[N][N] = { { 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 },

{ 0, 0, 0, 0 } };

if (solveNQUtil(board, 0) == false) {

printf("Solution does not exist");

return false;

}

printSolution(board);

return true;

}

int main()

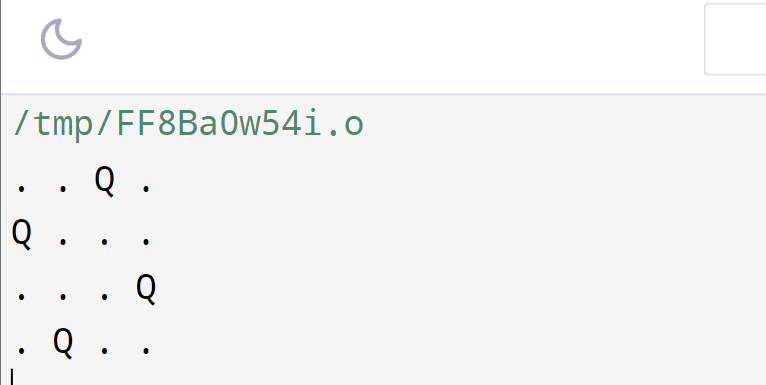
{

solveNQ();

return 0;

}

**Sample Output:**

****

**16. Write a C Program for Cycle formation.**

**Program Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_VERTICES 100

typedef struct {

int vertices[MAX\_VERTICES];

int edges[MAX\_VERTICES][MAX\_VERTICES];

int numVertices;

int numEdges;

} Graph;

void initializeGraph(Graph\* graph, int numVertices) {

graph->numVertices = numVertices;

graph->numEdges = 0;

for (int i = 0; i < MAX\_VERTICES; i++) {

graph->vertices[i] = 0;

for (int j = 0; j < MAX\_VERTICES; j++) {

graph->edges[i][j] = 0;

}

}

}

void addEdge(Graph\* graph, int src, int dest) {

graph->edges[src][dest] = 1;

graph->numEdges++;

}

bool DFS(Graph\* graph, int vertex, bool visited[], bool recStack[]) {

visited[vertex] = true;

recStack[vertex] = true;

for (int i = 0; i < graph->numVertices; i++) {

if (graph->edges[vertex][i]) {

if (!visited[i]) {

if (DFS(graph, i, visited, recStack))

return true;

} else if (recStack[i]) {

return true;

}

}

}

recStack[vertex] = false;

return false;

}

bool hasCycle(Graph\* graph) {

bool visited[MAX\_VERTICES] = { false };

bool recStack[MAX\_VERTICES] = { false };

for (int i = 0; i < graph->numVertices; i++) {

if (!visited[i]) {

if (DFS(graph, i, visited, recStack))

return true;

}

}

return false;

}

// Example usage

int main()

{

Graph graph;

int numVertices = 4;

initializeGraph(&graph, numVertices);

addEdge(&graph, 0, 1);

addEdge(&graph, 1, 2);

addEdge(&graph, 2, 0);

if (hasCycle(&graph))

printf("Graph contains a cycle.\n");

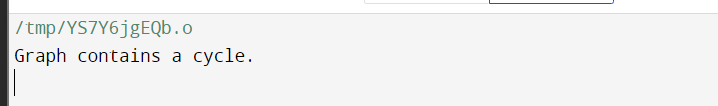
else

printf("Graph does not contain a cycle.\n");

return 0;

}

**Sample Output:**

****