

Name: Sriram Gopalkrishnan Iyer

Section: A3

Batch: B3

Roll No: 38

Ramdeobaba University, Nagpur
Department of Computer Science and Engineering
Session: 2024-2025

Design and Analysis of Algorithms Lab

III Semester

PRACTICAL NO. 7

Aim: Implement Hamiltonian Cycle using Backtracking.

Problem Statement:

The Smart City Transportation Department is designing a night-patrol route for security vehicles.

Each area of the city is represented as a vertex in a graph, and a road between two areas is represented as an edge.

The goal is to find a route that starts from the main headquarters (Area A), visits each area exactly once, and returns back to the headquarters — forming a Hamiltonian Cycle.

If such a route is not possible, display a suitable message.

Code:

```
#include <stdio.h>

int n;
int G[10][10];
int x[10];
int found = 0;

void NextVertex(int k) {
    while(1) {
        x[k] = (x[k] + 1) % (n + 1);
        if(x[k] == 0)
            return;

        if (G[x[k - 1]][x[k]] != 0) {
            int j;
            for (j = 1; j < k; j++)
                if (x[j] == x[k])
                    break;

            if(j == k) {
                if ((k < n) || (k == n && G[x[n]][x[1]] != 0))
                    return;
            }
        }
    }
}

void Hamiltonian(int k) {
    while(1) {
        NextVertex(k);
```

```

        if(x[k] == 0)
            return;

        if (k == n) {
            found = 1;
            printf("\nHamiltonian Cycle: ");
            for (int i = 1; i <= n; i++)
                printf("%d ", x[i]);
            printf("%d", x[1]);
        } else {
            Hamiltonian(k + 1);
        }
    }
}

int main() {
    printf("Enter the Number of Vertices: ");
    scanf("%d", &n);

    printf("Enter the Adjacency Matrix: ");
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) {
            scanf("%d", &G[i][j]);
        }
    }

    for(int i = 1; i <= n; i++)
        x[i] = 0;

    x[1] = 1;
    Hamiltonian(2);
}

```

```

    if(!found)
        printf("\nNo Hamiltonian Cycle exists.\n");

    return 0;
}

```

Output:

1) Adjacency Matrix

	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	1	0
C	1	1	0	1	0
D	0	1	1	0	1
E	1	0	0	1	0

```

Enter the Number of Vertices: 5
Enter the Adjacency Matrix: 0 1 1 0 1
1 0 1 1 0
1 1 0 1 0
0 1 1 0 1
1 0 0 1 0

Hamiltonian Cycle: 1 2 3 4 5 1
Hamiltonian Cycle: 1 3 2 4 5 1
Hamiltonian Cycle: 1 5 4 2 3 1
Hamiltonian Cycle: 1 5 4 3 2 1

```

1) Adjacency Matrix

	T	M	S	H	C
T	0	1	1	0	1
M	1	0	1	1	0
S	1	1	0	1	1
H	0	1	1	0	1
C	1	0	1	1	0

```

Enter the Number of Vertices: 5
Enter the Adjacency Matrix: 0 1 1 0 1
1 0 1 1 0
1 1 0 1 1
0 1 1 0 1
1 0 1 1 0

Hamiltonian Cycle: 1 2 3 4 5 1
Hamiltonian Cycle: 1 2 4 3 5 1
Hamiltonian Cycle: 1 2 4 5 3 1
Hamiltonian Cycle: 1 3 2 4 5 1
Hamiltonian Cycle: 1 3 5 4 2 1
Hamiltonian Cycle: 1 5 3 4 2 1
Hamiltonian Cycle: 1 5 4 2 3 1
Hamiltonian Cycle: 1 5 4 3 2 1%

```

GFG Output:

Problem

Editorial

Submissions

Comments

Output Window

Compilation Results

Custom Input

Y.O.G.I. (AI Bot)

Problem Solved Successfully

Suggest Feedback

Test Cases Passed

52 / 52

Attempts: Correct / Total

1 / 1

Accuracy : 100%

Points Scored

4 / 4

Your Total Score: 12

Time Taken

0.04


Solve Next

Number of Provinces

Number of Distinct Islands

Number of Good Components

Stay Ahead With:



Build 21 Projects in 21 Days
 Build real-world ML, Deep Learning & Gen AI projects
[Register Now](#)

Python3

Start Timer

```

1- class Solution:
2-     def check(self, n, m, edges):
3-         adj = [[] for _ in range(n + 1)]
4-         for u, v in edges:
5-             adj[u].append(v)
6-             adj[v].append(u)
7-
8-     def dfs(node, visited_count, visited):
9-         if visited_count == n:
10-             return True
11-
12-         for nei in adj[node]:
13-             if not visited[nei]:
14-                 visited[nei] = True
15-                 if dfs(nei, visited_count + 1, visited):
16-                     return True
17-                 visited[nei] = False
18-             return False
19-
20-     for start in range(1, n + 1):
21-         visited = [False] * (n + 1)
22-         visited[start] = True
23-         if dfs(start, 1, visited):
24-             return 1
25-
26-     return 0

```

Custom Input

Compile & Run

Submit