Name: Nachiket Jagdale

Class: A3-B1

Roll no.-02

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.

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

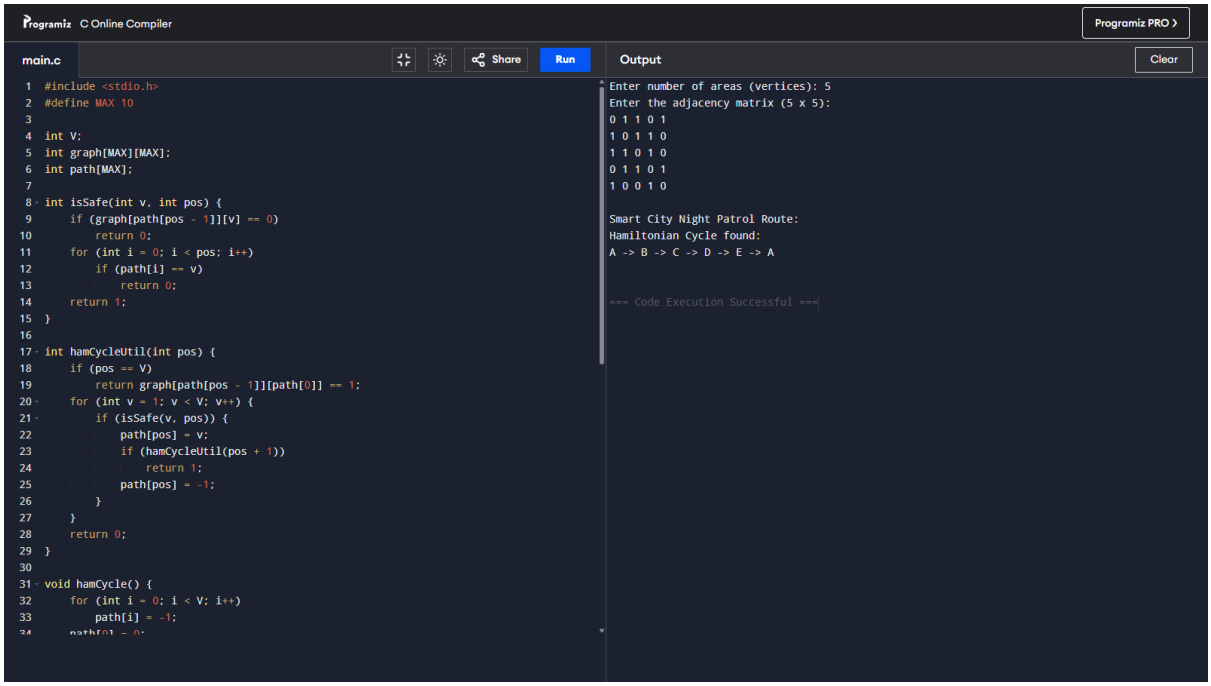
CODE:

```
#include <stdio.h>
#define MAX 10

int V;
int graph[MAX][MAX];
int path[MAX];

int isSafe(int v, int pos) {
    if (graph[path[pos - 1]][v] == 0)
        return 0;
    for (int i = 0; i < pos; i++)
        if (path[i] == v)
            return 0;
    return 1;
}

int hamCycleUtil(int pos) {
    if (pos == V)
        return graph[path[pos - 1]][path[0]] == 1;
    for (int v = 1; v < V; v++) {
        if (isSafe(v, pos)) {
            path[pos] = v;
            if (hamCycleUtil(pos + 1))
```

```
                return 1;
            path[pos] = -1;
        }
    }
    return 0;
}

void hamCycle() {
    for (int i = 0; i < V; i++)
        path[i] = -1;
    path[0] = 0;

    if (!hamCycleUtil(1)) {
        printf("No Hamiltonian Cycle exists.\n");
        return;
    }

    printf("Hamiltonian Cycle found:\n");
    for (int i = 0; i < V; i++)
        printf("%c -> ", 'A' + path[i]);
    printf("%c\n", 'A' + path[0]);
}

int main() {
    printf("Enter number of areas (vertices): ");
```

```
    scanf("%d", &V);

    printf("Enter the adjacency matrix (%d x %d):\n", V, V);
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &graph[i][j]);

    printf("\nSmart City Night Patrol Route:\n");
    hamCycle();

    return 0;
}
```

OUTPUT:



2) 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

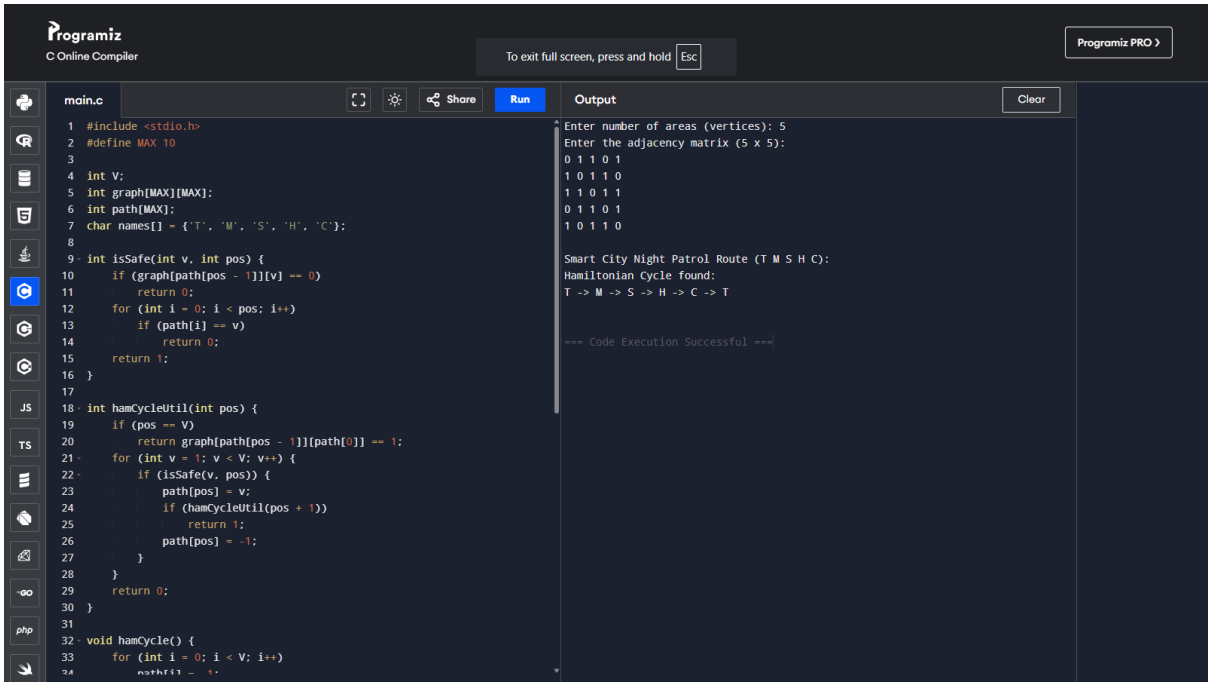
CODE:

```
#include <stdio.h>
#define MAX 10

int V;
int graph[MAX][MAX];
int path[MAX];
char names[] = {'T', 'M', 'S', 'H', 'C'};

int isSafe(int v, int pos) {
    if (graph[path[pos - 1]][v] == 0)
        return 0;
    for (int i = 0; i < pos; i++)
        if (path[i] == v)
            return 0;
    return 1;
}
```

```
int hamCycleUtil(int pos) {
    if (pos == V)
        return graph[path[pos - 1]][path[0]] == 1;
    for (int v = 1; v < V; v++) {
        if (isSafe(v, pos)) {
            path[pos] = v;
            if (hamCycleUtil(pos + 1))
                return 1;
            path[pos] = -1;
        }
    }
    return 0;
}

void hamCycle() {
    for (int i = 0; i < V; i++)
        path[i] = -1;
    path[0] = 0;

    if (!hamCycleUtil(1)) {
        printf("No Hamiltonian Cycle exists.\n");
        return;
    }

    printf("Hamiltonian Cycle found:\n");
```

```
    for (int i = 0; i < V; i++)
        printf("%c -> ", names[path[i]]);
    printf("%c\n", names[path[0]]);
}

int main() {
    printf("Enter number of areas (vertices): ");
    scanf("%d", &V);

    printf("Enter the adjacency matrix (%d x %d):\n", V, V);
    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            scanf("%d", &graph[i][j]);

    printf("\nSmart City Night Patrol Route (T M S H C):\n");
    hamCycle();

    return 0;
}
```

OUTPUT:



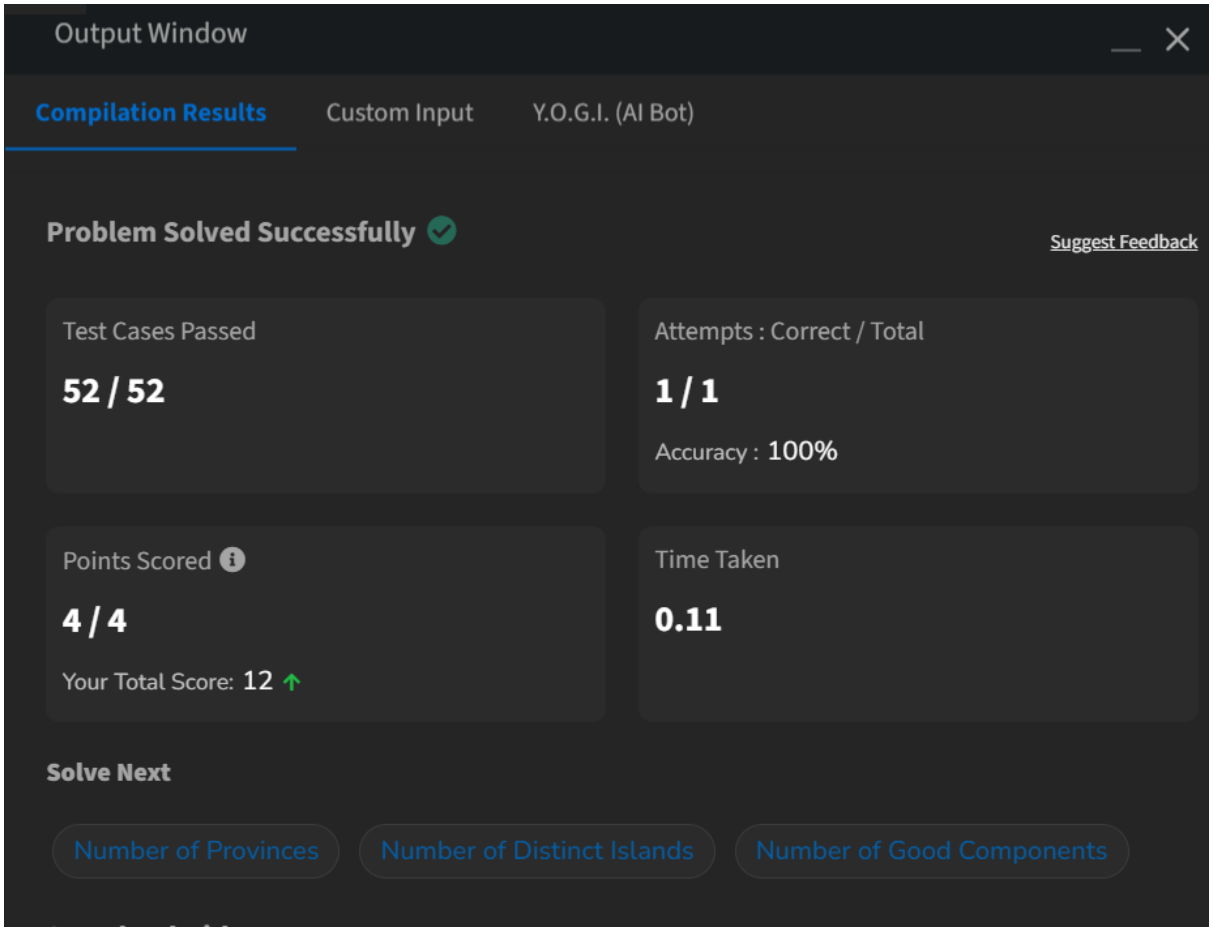
Competitive Coding Link:
https://www.geeksforgeeks.org/problems/hamiltonian-path2522/1

Code:

```
import java.util.*;

class Solution {

    boolean check(int n, int m, ArrayList<ArrayList<Integer>>
edges) {

        boolean[][] graph = new boolean[n + 1][n + 1];

        for (ArrayList<Integer> e : edges) {

            int u = e.get(0), v = e.get(1);

            graph[u][v] = true;

            graph[v][u] = true;

        }
```

```
        boolean[] visited = new boolean[n + 1];
        for (int i = 1; i <= n; i++)
            if (dfs(i, graph, visited, 1, n)) return true;
        return false;
    }
    boolean dfs(int node, boolean[][] graph, boolean[] visited, int
count, int n) {
        visited[node] = true;
        if (count == n) return true;
        for (int v = 1; v <= n; v++) {
            if (graph[node][v] && !visited[v])
                if (dfs(v, graph, visited, count + 1, n)) return true;
        }
        visited[node] = false;
        return false;
    }
}
```

OUTPUT:


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
0.11
Solve Next
Number of Provinces
Number of Distinct Islands
Number of Good Components