Name:Areeb Ansari Roll No:A3-B1-01

Practical 7

Aim: Implement Hamiltonian Cycle using Backtracking.

Problem Statement:

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

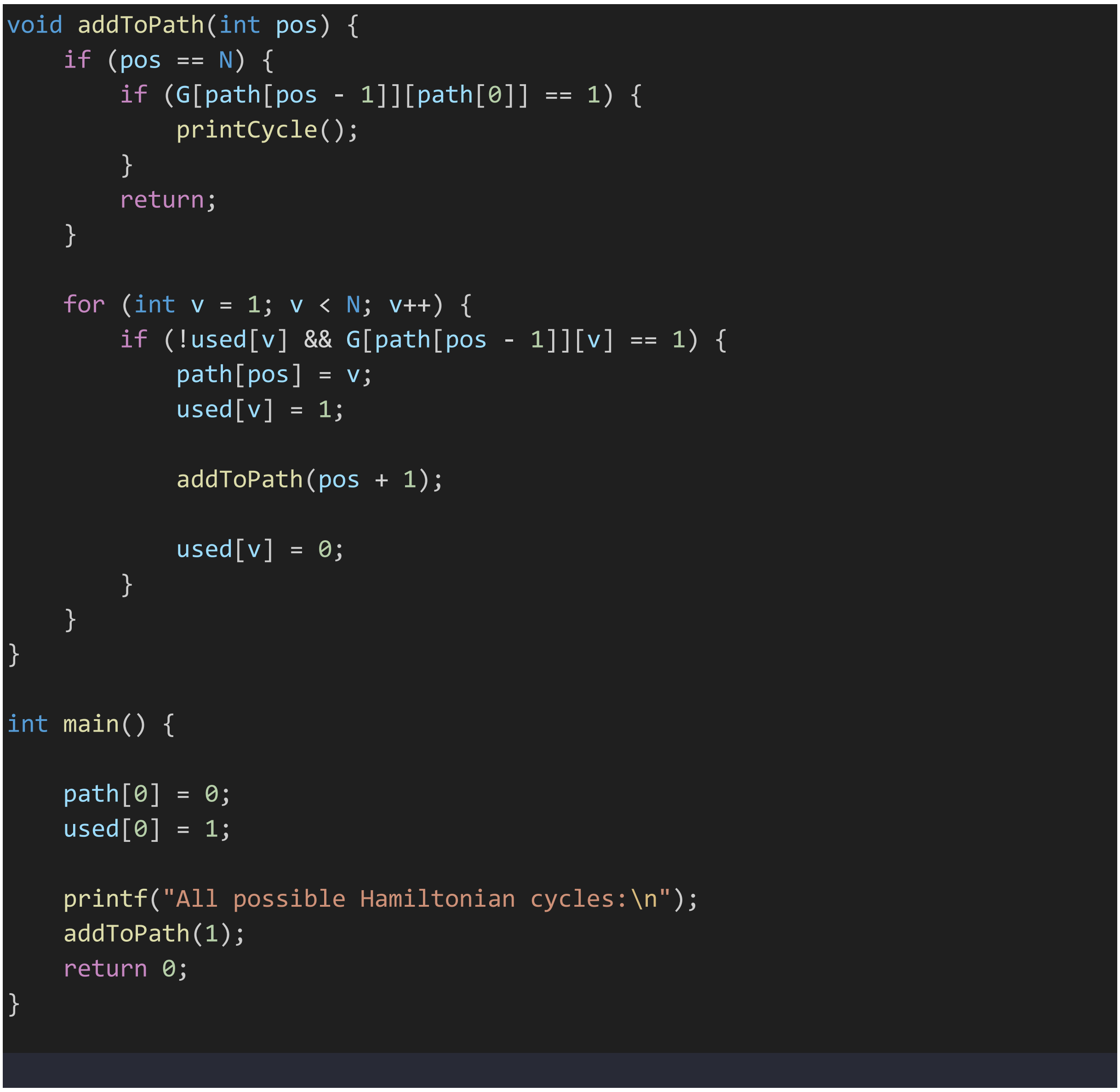
Each area of the city is represented as a vertex in a graph, and a road between two areas is represented as an edge.

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.

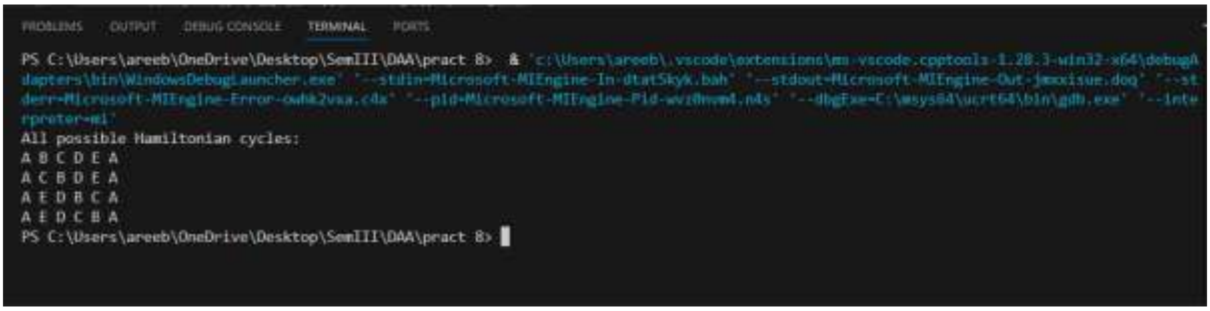
Code:

```c
#include <stdio.h>

#define N 5

int G[N][N] = {
    {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}
};

int path[N];
int used[N] = {0};

void printCycle() {
    for (int i = 0; i < N; i++) {
        printf("%c ", 'A' + path[i]);
    }
    printf("%c\n", 'A' + path[0]);
}
```



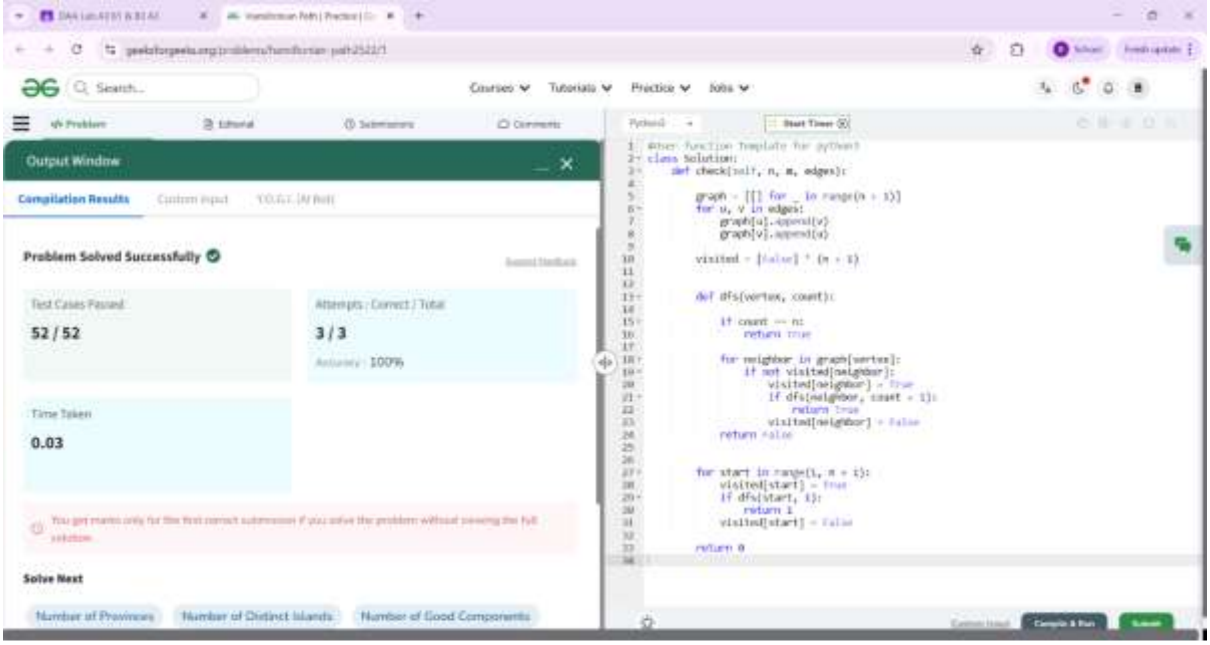
```c
void addToPath(int pos) {
    if (pos == N) {
        if (G[path[pos - 1]][path[0]] == 1) {
            printCycle();
        }
        return;
    }

    for (int v = 1; v < N; v++) {
        if (!used[v] && G[path[pos - 1]][v] == 1) {
            path[pos] = v;
            used[v] = 1;

            addToPath(pos + 1);

            used[v] = 0;
        }
    }
}

int main() {

    path[0] = 0;
    used[0] = 1;

    printf("All possible Hamiltonian cycles:\n");
    addToPath(1);
    return 0;
}
```

Output:



```
PS C:\Users\areeb\OneDrive\Desktop\SemIII\DAA\pract 8>  & 'c:\Users\areeb\.vscode\extensions\ms-vscode.cpptools-1.28.3-win32-x64\debugA
dapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-dtatSkyk.bah' '--stdout=Microsoft-MIEngine-Out-jmxxisue.doq' '--st
derr=Microsoft-MIEngine-Error-owhk2vaa.c4x' '--pid=Microsoft-MIEngine-Pid-wvi0nvm4.n4s' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--inte
rpreter=mi'
All possible Hamiltonian cycles:
A B C D E A
A C B D E A
A E D B C A
A E D C B A
PS C:\Users\areeb\OneDrive\Desktop\SemIII\DAA\pract 8> 
```

GFG problems



Problem Solved Successfully ✓

Test Cases Passed

52 / 52

Attempts : Correct / Total

3 / 3

Accuracy : 100%

Time Taken

0.03

You get marks only for the first correct submission if you solve the problem without viewing the full solution.

```python
#User function Template for python3
class Solution:
    def check(self, n, m, edges):

        graph = [[] for _ in range(n + 1)]
        for u, v in edges:
            graph[u].append(v)
            graph[v].append(u)

        visited = [False] * (n + 1)


        def dfs(vertex, count):

            if count == n:
                return True

            for neighbor in graph[vertex]:
                if not visited[neighbor]:
                    visited[neighbor] = True
                    if dfs(neighbor, count + 1):
                        return True
                    visited[neighbor] = False
            return False


        for start in range(1, n + 1):
            visited[start] = True
            if dfs(start, 1):
                return 1
            visited[start] = False

        return 0
```