Name- Anjali Lanjewar

Section- A4-B2

Roll no- 25

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

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

C code-

```
#include <stdio.h>

#define N 5

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

void hamUtil(int graph[][N], char names[], int path[], int used[], int
pos, int *found) {     if (pos == N) {
       if (graph[path[N-1]][path[0]] == 1) {
printf("Hamiltonian Cycle Found: ");
printCycle(names, path, N);
```

```
            (*found)++;
        }
        return;
    }

    for (int v = 1; v < N; ++v) {
        if (!used[v] && graph[path[pos-1]][v]) {
path[pos] = v;            used[v] = 1;
            hamUtil(graph, names, path, used, pos + 1, found);
used[v] = 0;                path[pos] = -1;
        }
    }
}

int hamCycleAll(int graph[][N], char names[], char *label) {
int path[N];    int used[N];    for (int i = 0; i < N; ++i) {
path[i] = -1;       used[i] = 0;

    }

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

```
    int found = 0;
    printf("\n=== %s ===\n", label);
    hamUtil(graph, names, path, used, 1, &found);

    if (!found) {
        printf("No Hamiltonian Cycle exists for this graph.\n");
    } else {
        printf("Total cycles found (starting at %c): %d\n", names[0],
found);
    }
    return found;
}

int main() {
    // Graph 1: A B C D E
int graph1[N][N] = {
/*A B C D E*/
        {0,1,1,0,1}, /*A*/
        {1,0,1,1,0}, /*B*/
{1,1,0,1,0}, /*C*/
        {0,1,1,0,1}, /*D*/
        {1,0,0,1,0}  /*E*/
    };
```

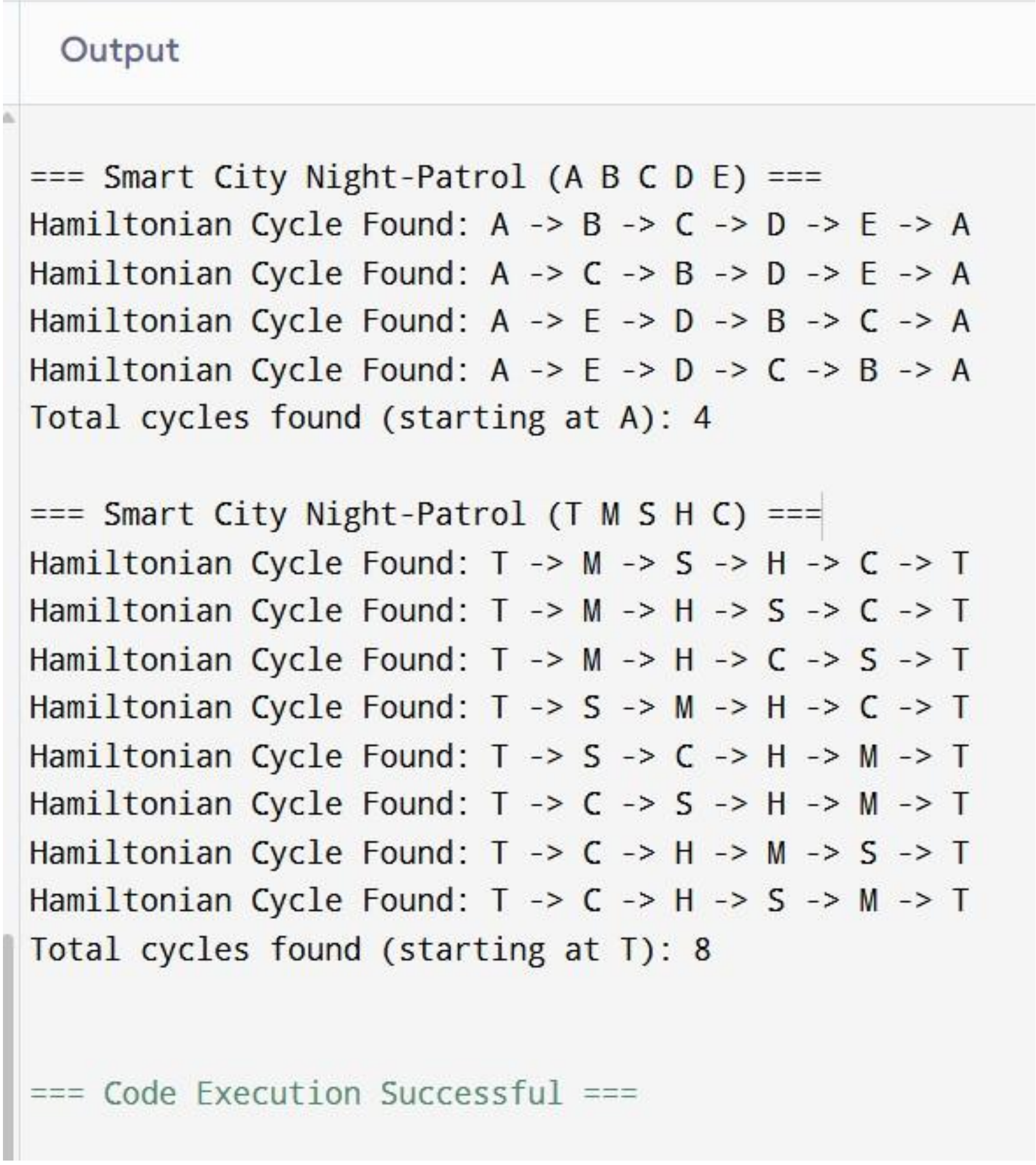
```
   char names1[N] = {'A','B','C','D','E'};


   // Graph 2: T M S H C
int graph2[N][N] = {
/*T M S H C*/
      {0,1,1,0,1}, /*T*/
      {1,0,1,1,0}, /*M*/
      {1,1,0,1,1}, /*S*/
      {0,1,1,0,1}, /*H*/
      {1,0,1,1,0}  /*C*/
   };
   char names2[N] = {'T','M','S','H','C'};


   hamCycleAll(graph1, names1, "Smart City Night-Patrol (A B C D E)");
   hamCycleAll(graph2, names2, "Smart City Night-Patrol (T M S H C)");


   return 0;
}
```

Output-



Output

```
=== Smart City Night-Patrol (A B C D E) ===
Hamiltonian Cycle Found: A -> B -> C -> D -> E -> A
Hamiltonian Cycle Found: A -> C -> B -> D -> E -> A
Hamiltonian Cycle Found: A -> E -> D -> B -> C -> A
Hamiltonian Cycle Found: A -> E -> D -> C -> B -> A
Total cycles found (starting at A): 4

=== Smart City Night-Patrol (T M S H C) ===
Hamiltonian Cycle Found: T -> M -> S -> H -> C -> T
Hamiltonian Cycle Found: T -> M -> H -> S -> C -> T
Hamiltonian Cycle Found: T -> M -> H -> C -> S -> T
Hamiltonian Cycle Found: T -> S -> M -> H -> C -> T
Hamiltonian Cycle Found: T -> S -> C -> H -> M -> T
Hamiltonian Cycle Found: T -> C -> S -> H -> M -> T
Hamiltonian Cycle Found: T -> C -> H -> M -> S -> T
Hamiltonian Cycle Found: T -> C -> H -> S -> M -> T
Total cycles found (starting at T): 8


=== Code Execution Successful ===
```