

DAA PRACTICAL 7

Name: Ansha Lanjewar

Section: A3_B1_13

Aim: Implement Hamiltonian Cycle using Backtracking.

Code:

```
#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 x[N];
int found = 0;

void printCycle() {
    for (int i = 0; i < N; i++)
        printf("%c ", x[i] + 'A');
    printf("%c\n", x[0] + 'A');
    found = 1;
}

void NextVertex(int k) {
    while (1) {
        x[k] = (x[k] + 1) % N;
        if (x[k] == 0)
            return;
        if (G[x[k] - 1][x[k]] != 0) {
            int j;
            for (j = 0; j < k; j++)
                if (x[j] == x[k])
                    break;
            if (j == k) {
```

```

        if (k < N - 1 || (k == N - 1 && G[x[k]][x[0]] != 0))
            return;
    }
}

}

}

void Hamiltonian(int k) {
    while (1) {
        NextVertex(k);
        if (x[k] == 0)
            return;
        if (k == N - 1)
            printCycle();
        else
            Hamiltonian(k + 1);
    }
}

int main() {
    for (int i = 0; i < N; i++)
        x[i] = 0;
    x[0] = 0;
    Hamiltonian(1);
    if (found)
        printf("Hamiltonian cycle found.\n");
    else
        printf("No Hamiltonian cycle found.\n");

return 0;
}

```

OUTPUT:

```
[Running] cd "c:\Users\DT USER\Desktop\1A333333\DAA\A3
A B C D E A
A C B D E A
A E D B C A
A E D C B A
Hamiltonian cycle found.

[Done] exited with code=0 in 0.264 seconds
```

```
[Running] cd "c:\Users\DT USER\Desktop\1A333333\DAA\A333333\" && g++ DAA7.C -o DAA7 8
T U V W X T
T U W V X T
T U W X V T
T V U W X T
T V X W U T
T X V W U T
T X W U V T
T X W V U T
Hamiltonian cycle found.

[Done] exited with code=0 in 0.25 seconds
```

The screenshot displays a coding platform interface with a dark theme. On the left, the 'Output Window' shows 'Compilation Results' for a custom input file 'Y.O.G.I. (AI Bot)'. It reports 'Problem Solved Successfully' with 52/52 test cases passed, 1/1 attempts correct, 100% accuracy, 4/4 points scored, and a time taken of 0.11 seconds. Below this, there are buttons for 'Solve Next' and 'Stay Ahead With: Build 21 Projects in 21 Days'. On the right, the code editor shows a Java solution for finding a Hamiltonian cycle using Depth-First Search (DFS). The code defines a 'Solution' class with a 'check' method that iterates through all nodes and a 'dfs' method that recursively explores the graph. The 'check' method returns true if a Hamiltonian cycle is found, and false otherwise. The 'dfs' method uses a 'visited' array to track nodes and a 'count' to track the number of nodes visited. The 'check' method calls 'dfs' for each node and returns the result.

```
1 class Solution {
2     boolean check(int n, int m, ArrayList<ArrayList<Integer>> edges) {
3         ArrayList<ArrayList<Integer>> adj = new ArrayList<>();
4         for (int i = 0; i < n; i++) adj.add(new ArrayList<>());
5         for (ArrayList<Integer> edge : edges) {
6             int u = edge.get(0) - 1;
7             int v = edge.get(1) - 1;
8             adj.get(u).add(v);
9             adj.get(v).add(u);
10        }
11
12        boolean[] visited = new boolean[n];
13
14        for (int i = 0; i < n; i++) {
15            if (dfs(i, 1, visited, adj, n)) return true;
16        }
17        return false;
18    }
19
20    boolean dfs(int node, int count, boolean[] visited, ArrayList<ArrayList<Integer>> adj, int n) {
21        visited[node] = true;
22        if (count == n) return true;
23
24        for (int neighbor : adj.get(node)) {
25            if (!visited[neighbor]) {
26                if (dfs(neighbor, count + 1, visited, adj, n)) return true;
27            }
28        }
29        visited[node] = false;
30        return false;
31    }
32 }
33 }
```