

```

def Hamiltonian(n, G):
    x = [0] * (n + 1)
    x[1] = 1

    def NextVertex(k):
        while True:
            x[k] = (x[k] + 1) % (n + 1)
            if x[k] == 0:
                return
            if G[x[k] - 1][x[k] - 1] != 0:
                for j in range(1, k):
                    if x[j] == x[k]:
                        break
            else:
                if k < n or (k == n and G[x[n] - 1][x[1] - 1] != 0):
                    return

    def HamiltonianCycle(k):
        while True:
            NextVertex(k)
            if x[k] == 0:
                return False
            if k == n:
                if G[x[n] - 1][x[1] - 1] != 0:
                    return x[1:n + 1]
            else:
                result = HamiltonianCycle(k + 1)
                if result:
                    return result

    cycle = HamiltonianCycle(2)
    if cycle:
        print("Hamiltonian Cycle:", cycle)
    else:
        print("No Hamiltonian Cycle exists.")

G1 = [
    [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]
]
G2 = [

```

```

[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]
]

```

```

print("Graph 1:")
Hamiltonian(5, G1)
print("\nGraph 2:")
Hamiltonian(5, G2)

```

The screenshot displays the LeetCode interface for a problem. On the left, the 'Output Window' shows 'Problem Solved Successfully' with 52/52 test cases passed, 1/1 attempts correct, and an accuracy of 100%. The points scored are 4/4, and the time taken is 0.04 seconds. Below this, there are buttons for 'Solve Next' and a section titled 'Stay Ahead With:' featuring a promotion for 'Build 21 Projects in 21 Days'.

On the right, the code editor shows the following Python code:

```

1 # Better function Template for python3
2 class Solution:
3     def check(self, n, m, edges):
4         adj = [[] for _ in range(n)]
5         for u, v in edges:
6             adj[u-1].append(v-1)
7             adj[v-1].append(u-1)
8
9     def backtrack(path, visited):
10        if len(path) == n:
11            return True
12        last = path[-1]
13        for nei in adj[last]:
14            if not visited[nei]:
15                visited[nei] = True
16                path.append(nei)
17                if backtrack(path, visited):
18                    return True
19                path.pop()
20                visited[nei] = False
21        return False
22
23    def hamiltonian(self, n, edges):
24        visited = [False] * n
25        visited[0] = True
26        if backtrack([0], visited):
27            return 1
28        return 0

```