

```

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] - 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 shows a programming competition interface with the following details:

- Problem:** Problem 48
- Editorial:** Editorial
- Submissions:** Submissions
- Comments:** Comments
- Output Window:** Output Window
- Compilation Results:** Custom Input, Y.O.G.I. (AI Bot)
- Status:** Problem Solved Successfully (green checkmark)
- Test Cases Passed:** 52 / 52
- Attempts:** Correct / Total: 1 / 1
- Accuracy:** 100%
- Points Scored:** 4 / 4
- Time Taken:** 0.04
- Your Total Score:** 4
- Solve Next:** Solve Next
- Stay Ahead With:** Build 21 Projects in 21 Days (with a small icon of a computer monitor)
- Code Editor (Python3):**

```
1 #User Function Template for python3  
2 class Solution:  
3     def check(self, n, m, edges):  
4         adj = [[0] for _ in range(n)]  
5         for u, v, w in edges:  
6             adj[u-1].append(v-1)  
7             adj[v-1].append(u-1)  
8         def backtrack(path, visited):  
9             if len(path) == n:  
10                 return True  
11             last = path[-1]  
12             for nei in adj[last]:  
13                 if not visited[nei]:  
14                     visited[nei] = True  
15                     path.append(nei)  
16                     if backtrack(path, visited):  
17                         return True  
18                     path.pop()  
19                     visited[nei] = False  
20             return False  
21  
22         for start in range(n):  
23             visited = [False]*n  
24             visited[start] = True  
25             if backtrack([start], visited):  
26                 return 1  
27         return 0
```
- Advertisement:** Kick start your career with GFG 160!