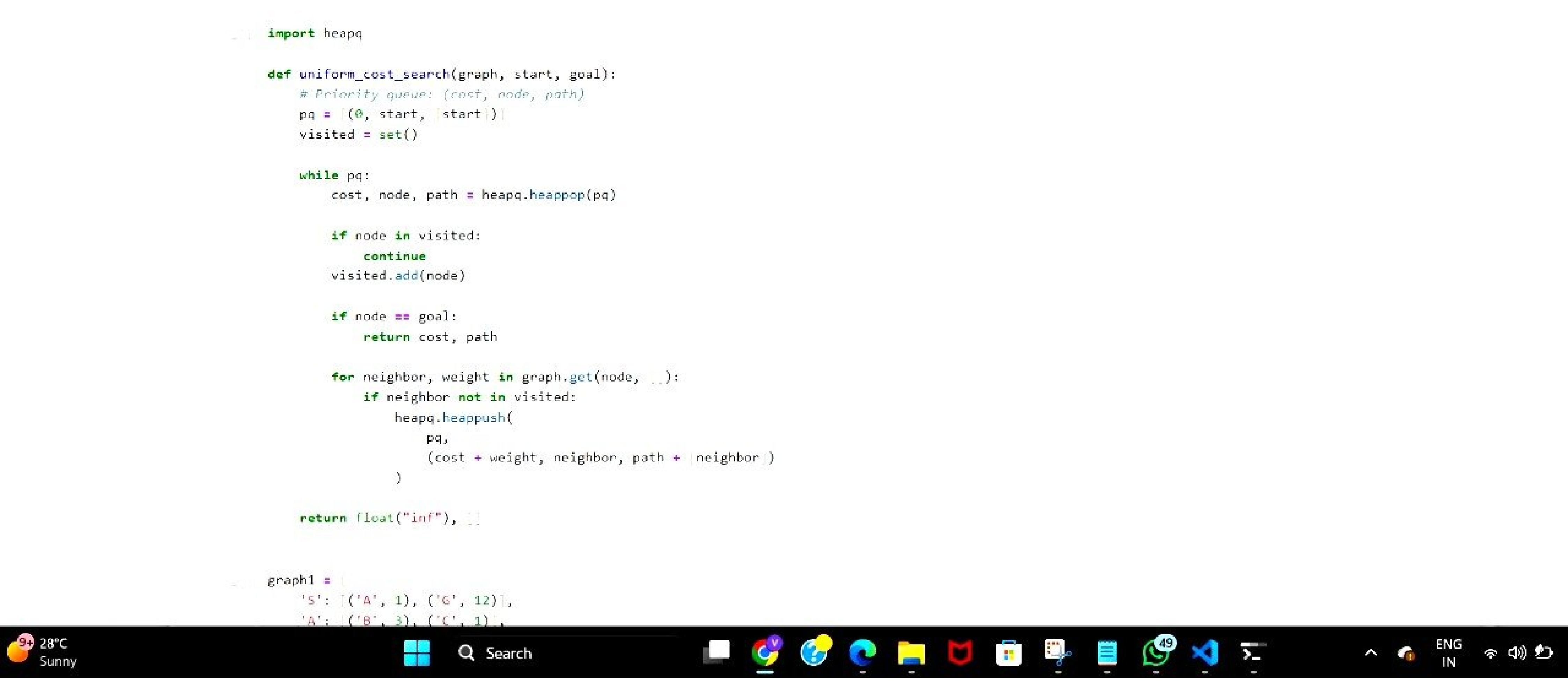
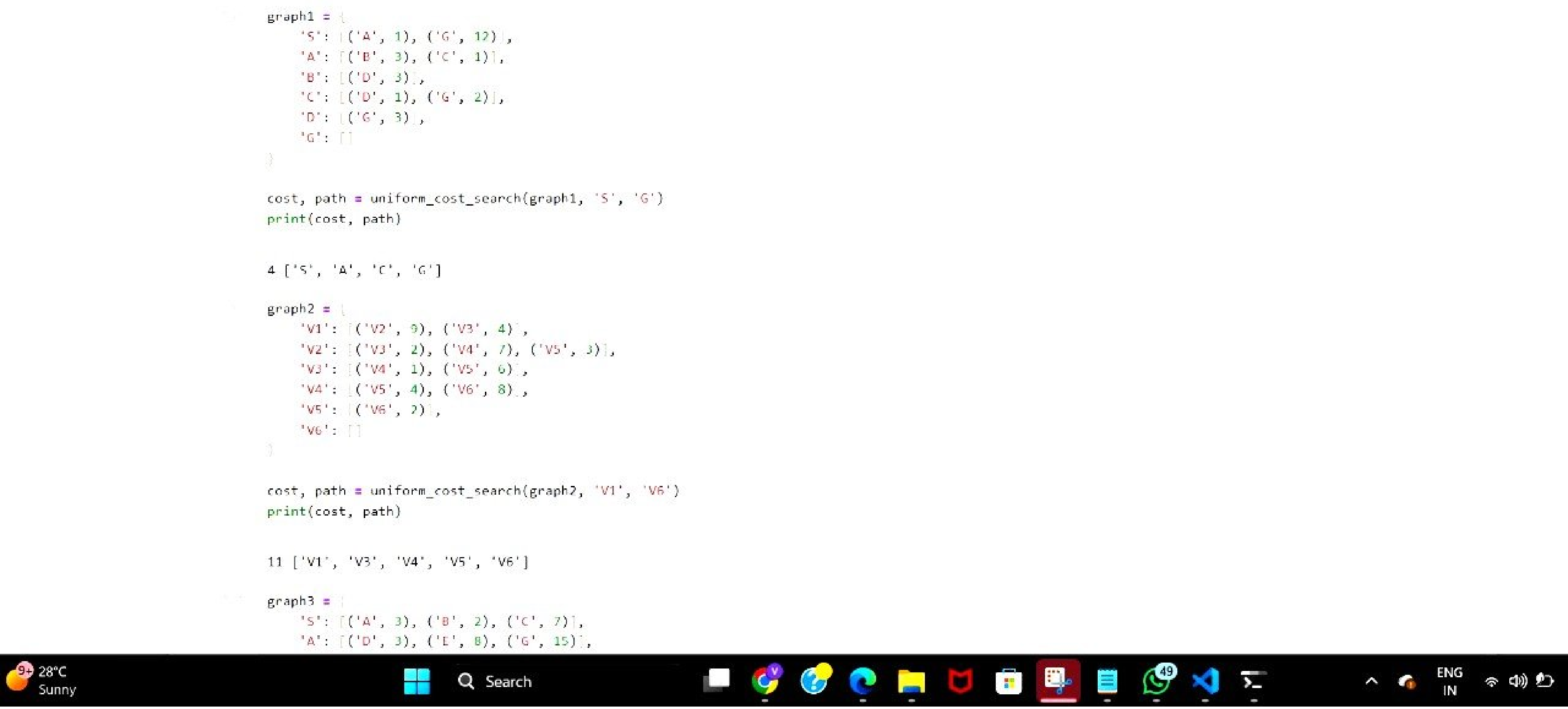


```python
import heapq

def uniform_cost_search(graph, start, goal):
    # Priority queue: (cost, node, path)
    pq = [(0, start, [start])]
    visited = set()

    while pq:
        cost, node, path = heapq.heappop(pq)

        if node in visited:
            continue
        visited.add(node)

        if node == goal:
            return cost, path

        for neighbor, weight in graph.get(node, []):
            if neighbor not in visited:
                heapq.heappush(
                    pq,
                    (cost + weight, neighbor, path + [neighbor])
                )

    return float("inf"), []


graph1 = {
    'S': [('A', 1), ('G', 12)],
    'A': [('B', 3), ('C', 1)],
```



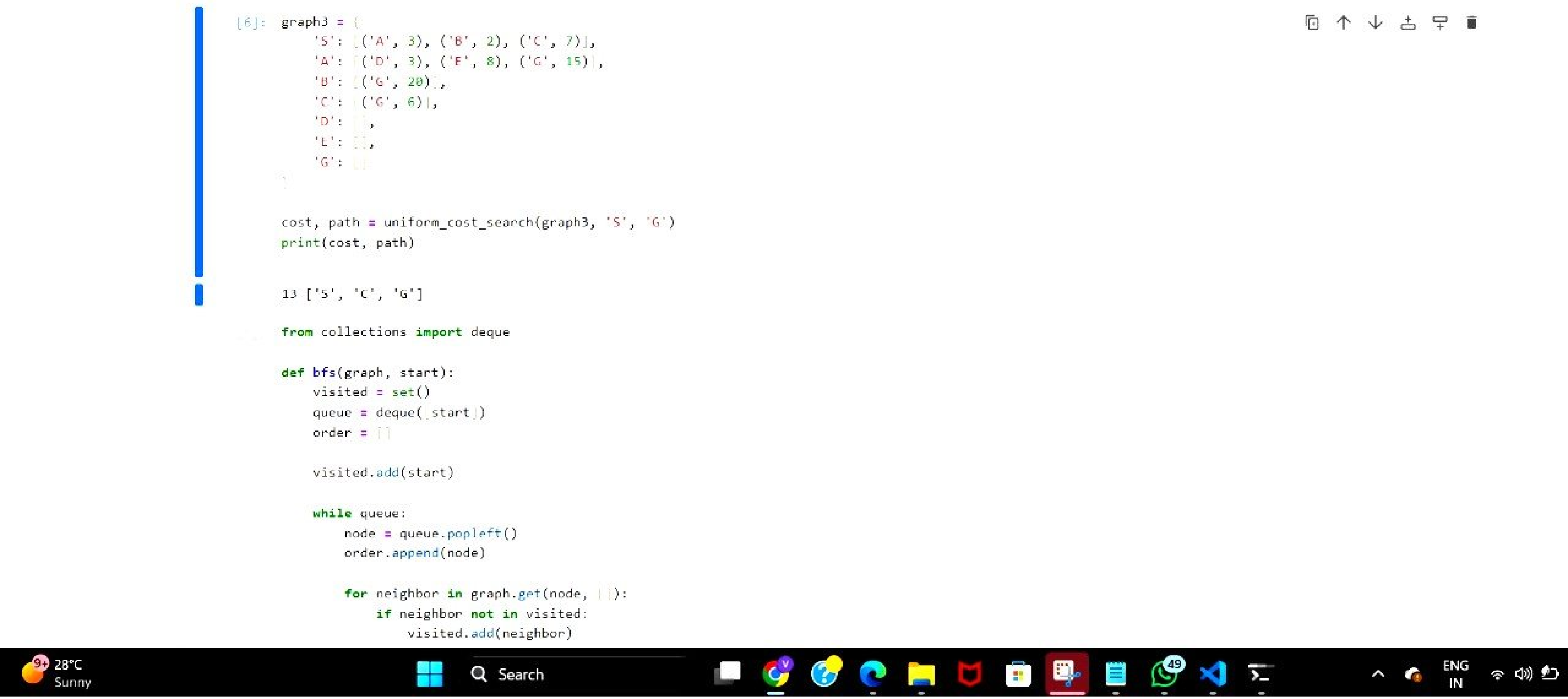
```
graph1 = {
    'S': [('A', 1), ('G', 12)],
    'A': [('B', 3), ('C', 1)],
    'B': [('D', 3)],
    'C': [('D', 1), ('G', 2)],
    'D': [('G', 3)],
    'G': []
}

cost, path = uniform_cost_search(graph1, 'S', 'G')
print(cost, path)
```

```
4 ['S', 'A', 'C', 'G']
```

```
graph2 = {
    'V1': [('V2', 9), ('V3', 4)],
    'V2': [('V3', 2), ('V4', 7), ('V5', 3)],
    'V3': [('V4', 1), ('V5', 6)],
    'V4': [('V5', 4), ('V6', 8)],
    'V5': [('V6', 2)],
    'V6': []
}

cost, path = uniform_cost_search(graph2, 'V1', 'V6')
print(cost, path)
```

```
11 ['V1', 'V3', 'V4', 'V5', 'V6']
```

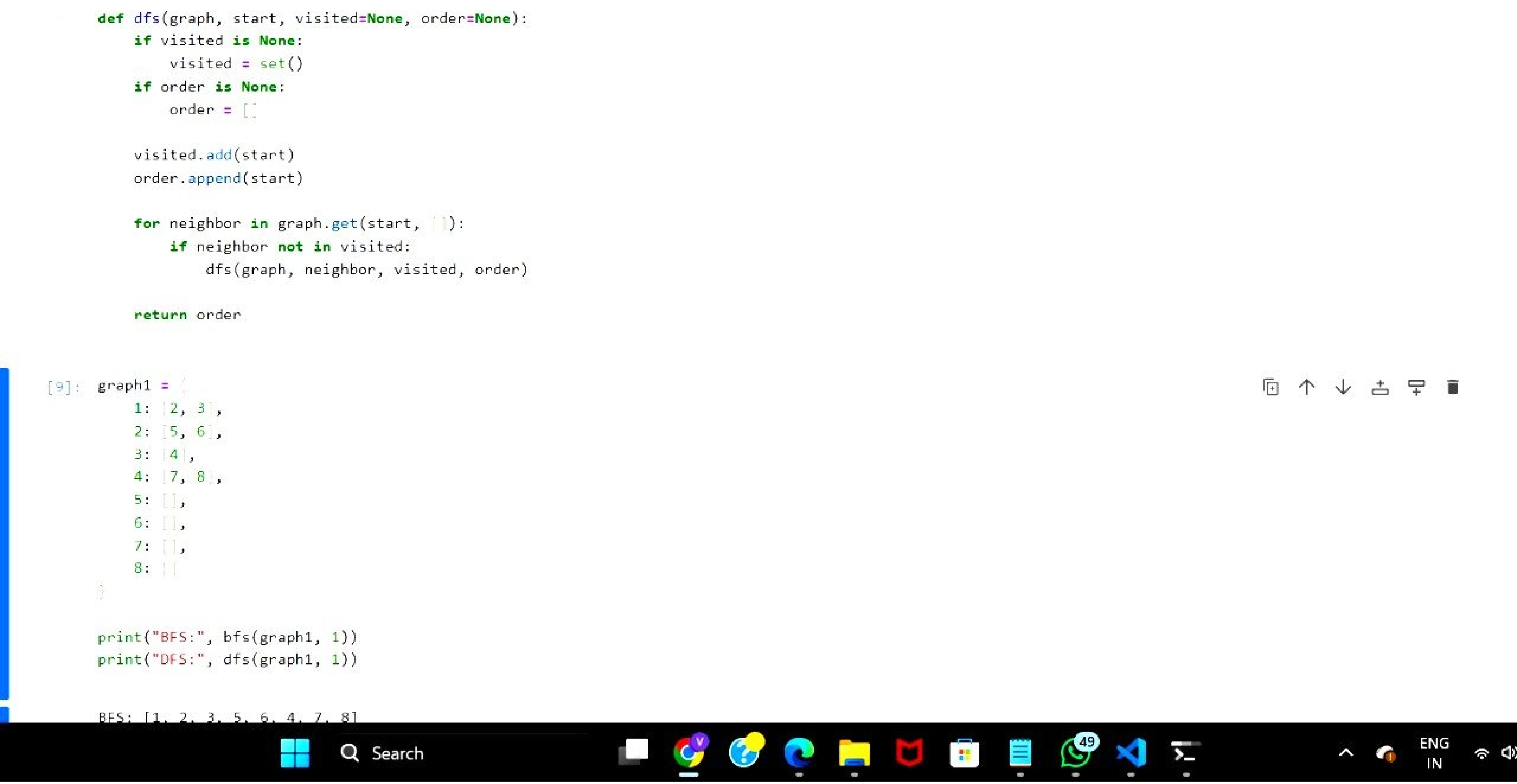
```
graph3 = {
    'S': [('A', 3), ('B', 2), ('C', 7)],
    'A': [('D', 3), ('E', 8), ('G', 15)],
```



```
[6]: graph3 = {
         'S': [('A', 3), ('B', 2), ('C', 7)],
         'A': [('D', 3), ('E', 8), ('G', 15)],
         'B': [('G', 20)],
         'C': [('G', 6)],
         'D': [],
         'E': [],
         'G': []
     }

     cost, path = uniform_cost_search(graph3, 'S', 'G')
     print(cost, path)
```

```
13 ['S', 'C', 'G']
```

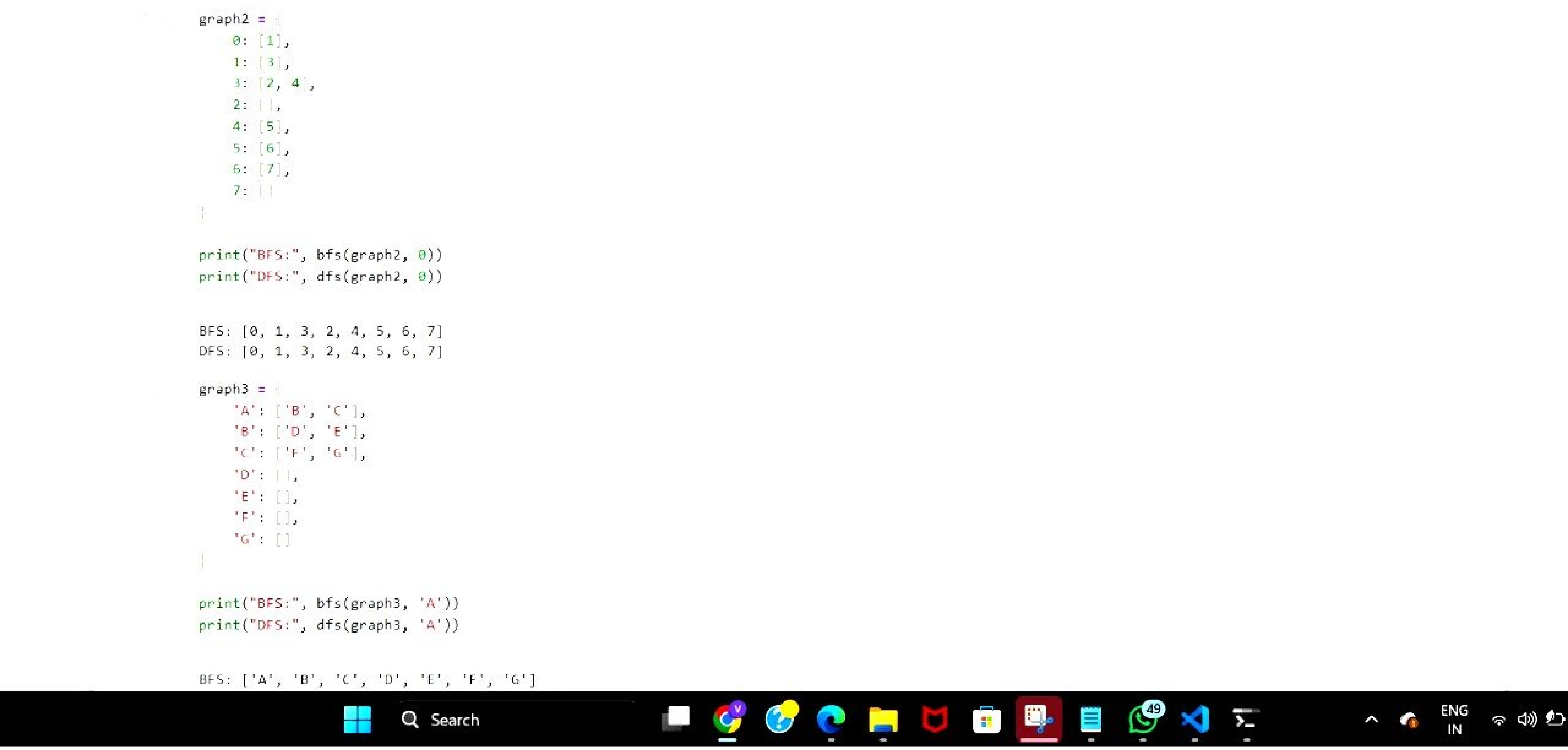
```
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    order = []

    visited.add(start)

    while queue:
        node = queue.popleft()
        order.append(node)

        for neighbor in graph.get(node, []):
            if neighbor not in visited:
                visited.add(neighbor)
```



```python
def dfs(graph, start, visited=None, order=None):
    if visited is None:
        visited = set()
    if order is None:
        order = []

    visited.add(start)
    order.append(start)

    for neighbor in graph.get(start, []):
        if neighbor not in visited:
            dfs(graph, neighbor, visited, order)

    return order
```

[9]:
```python
graph1 = {
    1: [2, 3],
    2: [5, 6],
    3: [4],
    4: [7, 8],
    5: [],
    6: [],
    7: [],
    8: []
}

print("BFS:", bfs(graph1, 1))
print("DFS:", dfs(graph1, 1))
```

```
BFS: [1, 2, 3, 5, 6, 4, 7, 8]
```

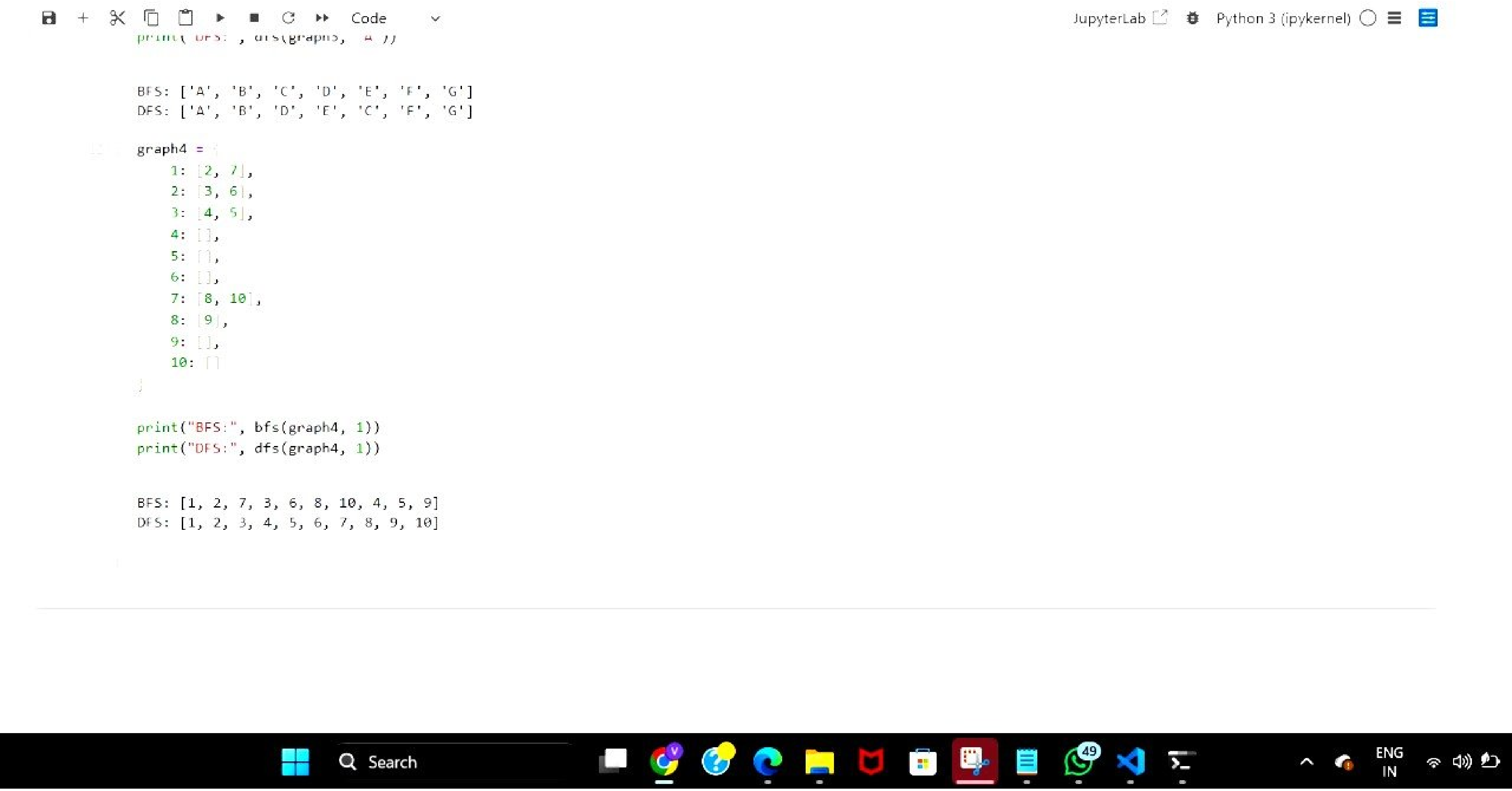


```
graph2 = {
    0: [1],
    1: [3],
    3: [2, 4],
    2: [],
    4: [5],
    5: [6],
    6: [7],
    7: []
}

print("BFS:", bfs(graph2, 0))
print("DFS:", dfs(graph2, 0))
```

```
BFS: [0, 1, 3, 2, 4, 5, 6, 7]
DFS: [0, 1, 3, 2, 4, 5, 6, 7]
```

```
graph3 = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F', 'G'],
    'D': [],
    'E': [],
    'F': [],
    'G': []
}

print("BFS:", bfs(graph3, 'A'))
print("DFS:", dfs(graph3, 'A'))
```

```
BFS: ['A', 'B', 'C', 'D', 'E', 'F', 'G']
```



```
print("DFS:", dfs(graph3, 'A'))


BFS: ['A', 'B', 'C', 'D', 'E', 'F', 'G']
DFS: ['A', 'B', 'D', 'E', 'C', 'F', 'G']

graph4 = {
    1: [2, 7],
    2: [3, 6],
    3: [4, 5],
    4: [],
    5: [],
    6: [],
    7: [8, 10],
    8: [9],
    9: [],
    10: []
}

print("BFS:", bfs(graph4, 1))
print("DFS:", dfs(graph4, 1))


BFS: [1, 2, 7, 3, 6, 8, 10, 4, 5, 9]
DFS: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```