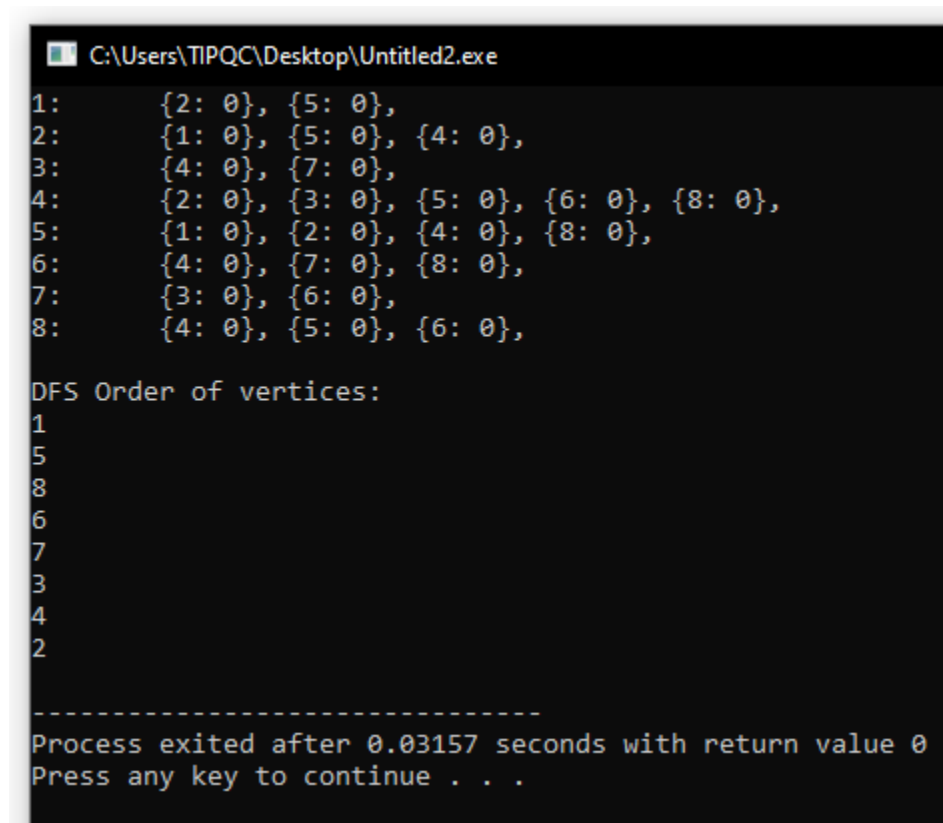


Activity No. 10.2	
Implementing DFS and Graph Applications	
<b>Course Code:</b> CPE010	<b>Program:</b> Computer Engineering
<b>Course Title:</b> Data Structures and Algorithms	<b>Date Performed:</b> 11/ 27 / 2024
<b>Section:</b> CPE21S4	<b>Date Submitted:</b> 11/ 27 / 2024
<b>Name(s):</b> BONA, Andrei Nycole So MASANGKAY, Frederick ROALLOS, Jean Gabriel Vincent SANTOS, Andrei ZOLINA, Anna Marie	<b>Instructor:</b> Prof. Maria Rizette Sayo

#### A. Output(s) and Observation(s)



```

C:\Users\TIPQC\Desktop\Untitled2.exe
1:      {2: 0}, {5: 0},
2:      {1: 0}, {5: 0}, {4: 0},
3:      {4: 0}, {7: 0},
4:      {2: 0}, {3: 0}, {5: 0}, {6: 0}, {8: 0},
5:      {1: 0}, {2: 0}, {4: 0}, {8: 0},
6:      {4: 0}, {7: 0}, {8: 0},
7:      {3: 0}, {6: 0},
8:      {4: 0}, {5: 0}, {6: 0},

DFS Order of vertices:
1
5
8
6
7
3
4
2

-----
Process exited after 0.03157 seconds with return value 0
Press any key to continue . . .

```

Figure 1: Output of given C++ code

#### B. Answers to Supplementary Activity

1. The Depth-First Search algorithm would be the best process to be done by the said person. Since the person wants to backtrack and reroute from different vertices, the process the person wants to implement in its exploration is the Depth-First Search.
2. DFS of a graph can have a unique traversal path in certain conditions. When a graph is directed, the vertex at the beginning of a graph is fixed, and each neighbor is visited in a particular order (e.g., the alphabetical order). In such a situation, the deterministic nature of the DFS builds a clear path for the algorithm to explore, thus the traversal path becomes unique. Thus, no confusions arise, and the DFS order is naturally unique.

3.

```
def dfs(graph, vertex, visited):
    if vertex not in visited:
        print(f"Visiting: {vertex}")
        visited.add(vertex)
        for neighbor in graph[vertex]:
            dfs(graph, neighbor, visited)
    else:
        print(f"Revisiting during backtracking: {vertex}")

# Example graph with cycles
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': ['A'] # Cycle back to A
}

visited = set()
dfs(graph, 'A', visited)
```

```
Visiting: A
Visiting: B
Visiting: D
Visiting: E
Visiting: F
Revisiting during backtracking: A
Visiting: C
Revisiting during backtracking: F

Process finished with exit code 0
```

In DFS, each vertex is visited once, although it might be revisited during backtracking. Revisits happen in graphs with cycles or multiple edges leading to the same vertex. For example, in a graph with a cycle involving vertex 'A', DFS revisits 'A' while backtracking but doesn't explore it again. This ensures no vertex is explored more than once.

4. DFS is useful in many areas. It helps find paths in mazes or graphs, sort tasks in order, detect cycles, and find connected components in graphs. It's also used for solving puzzles like Sudoku and exploring game decision trees in AI. Its systematic approach to exploring and backtracking makes it versatile.

5.

```
class TreeNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

def preorder_traversal(node):
    if node:
        print(node.value, end=" ")
        preorder_traversal(node.left)
        preorder_traversal(node.right)
```

```
# Example tree
root = TreeNode('A')
root.left = TreeNode('B')
root.right = TreeNode('C')
root.left.left = TreeNode('D')
root.left.right = TreeNode('E')

# Preorder Traversal
print("Preorder Traversal of Tree:")
preorder_traversal(root)
```

```
Preorder Traversal of Tree:
```

```
A B D E C
```

```
Process finished with exit code 0
```

In trees, DFS is similar to preorder traversal, where you visit the root first, then the left subtree, and finally the right subtree. For example, in a tree with nodes  $A \rightarrow B \rightarrow D, E$  and  $A \rightarrow C$ , the preorder traversal order is  $A \rightarrow B \rightarrow D \rightarrow E \rightarrow C$ . Both DFS and preorder explore one path fully before backtracking, making them equivalent in behavior.

### C. Conclusion & Lessons Learned

This activity delved into the Depth-First Search (DFS) algorithm, demonstrating its implementation and applications in graph traversal. By understanding its recursive nature and backtracking, we can effectively solve problems involving pathfinding, cycle, and connected component analysis. The provided code in this activity examples and exercises our understanding about DFS and its practical use cases, making it a valuable tool for our new learnings.

### D. Assessment Rubric

### E. External References