Name: Anuska Nath
Dept: IT (UG3)
Section: A3
Roll: 002311001003

Assignment 1

1. Understand the Python language, code library, package etc.

NumPy

Overview:

NumPy is a library for efficient numerical computations, providing support for large, multi-dimensional arrays and matrices.

Key Features:

- **ndarray**: A fast, flexible, and efficient array object.
- **Broadcasting**: Allows operations on arrays of different shapes.
- **Mathematical Functions**: Includes operations like addition, square root, etc.

Pandas

Overview:

Pandas is used for data manipulation and analysis, providing powerful data structures like `DataFrame`.

Key Features:

- **DataFrame**: A table-like data structure with rows and columns.
- **Data Handling**: Tools for cleaning, transforming, and merging data.

Matplotlib

Overview:

Matplotlib is a plotting library for creating static, animated, and interactive visualizations.

Key Features:

- **Plotting: Create line, bar, scatter, and more plots.**
- **Customization: Extensive options for customizing plot appearance.**

2. Write a BFS search function of a small graph without using recursion.
3. Try to print the graph.

BFS Implementation for Graph Traversal

1. Introduction

Breadth-First Search (BFS) is an algorithm for graph traversal that explores all the nodes level by level, starting from a source node. In this implementation, we use an **Adjacency List** to represent the graph and perform BFS iteratively without recursion.

2. Input Data

1. **Graph Details**:
    - Nodes are represented by single characters (e.g., `A`, `B`, `C`).
    - The graph is represented by a list of edges, where each edge connects two nodes.

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

2. **BFS Start Node**:
    - The starting node for the BFS traversal is specified while calling.

```
print("BFS Traversal starting from 'A':")

bfs(graph, 'A')
```

3. Output Results

1. **Graph Representation**:
    - The graph is printed as an adjacency list.
2. **BFS Traversal Output**:
    - The nodes are printed in the order they are visited during the BFS traversal.

OUTPUT:

BFS Traversal starting from 'A':

A B C D E F

Graph Representation:

A: B, C

B: A, D, E

C: A, F

D: B

E: B, F

F: C, E

3. Process Used to Solve the Problem

To solve the BFS traversal problem, the following approach was used:

1. **Graph Representation**:
    - The graph was represented using an **Adjacency List**. This data structure allows easy access to each node's neighbors.

2. **BFS Traversal**:
    - A **Queue** was used to keep track of the nodes to be processed. BFS is a level-wise traversal, so the queue helps ensure that nodes are processed in the correct order.
    - **Visited Nodes**: A set was maintained to track the nodes that had already been visited, ensuring that no node is visited more than once.

3. **Traversal Process**:
    - Starting from the specified node, we enqueue the node and mark it as visited.
    - For each node, we dequeue it and print it. Then, we enqueue all its unvisited neighbors and mark them as visited.
    - The process repeats until all reachable nodes have been visited.

This process guarantees that all nodes connected to the start node are explored, and all nodes are visited in the correct BFS order.

4. Conclusion

This BFS implementation effectively solves the problem of graph traversal. By using a queue for level-by-level exploration and a set to track visited nodes, the program efficiently explores all nodes in the graph starting from the specified node. The result is a clear, step-by-step output of the graph's traversal.