Summary on methods, algorithms, and data structures: For assignment 7

ASSIGNMENT NUMBER : 07
NAME : Dinesh Raj Pampati
ULID : dpampat
NAME OF SECRET DIRECTORY : namo

1. Methods Used

Ford-Fulkerson Program

1. **main(String[] args)**:
    - **Purpose**: Serves as the entry point of the program. It reads the flow network data from a file and initiates the maximum flow computation.
    - **Process**: Opens the specified file and reads the network data to construct a graph represented as an adjacency list. After constructing the graph, it calls the **fordFulkerson** method to find the maximum flow from a given source to a sink.
2. **fordFulkerson(ArrayList<ArrayList<Edges>> Graph, int source, int destination)**:
    - **Purpose**: Implements the Ford-Fulkerson algorithm to calculate the maximum flow in a flow network.
    - **Process**: Initializes flow in all edges to zero. It repeatedly searches for augmenting paths using the **bfs** method. For each found path, it updates the flow along this path. The process repeats until no augmenting path can be found. The sum of the flow values then represents the maximum flow.
3. **bfs(ArrayList<ArrayList<Edges>> Graph, int source, int destination, int[] parent)**:
    - **Purpose**: Implements the Breadth-First Search algorithm to find an augmenting path from source to sink in the residual graph.
    - **Process**: Starts from the source and explores all the vertices level by level. It keeps track of visited nodes and their parents to reconstruct the path once the sink is reached. If the sink is reached, returns the path; otherwise, returns null indicating no path is found.

Preflow-Push Program

1. **main(String[] args)**:
    - **Purpose**: Acts as the entry point for the Preflow-Push program. It reads flow network data from a file and starts the flow computation process.
    - **Process**: Reads the graph structure from a file to create an adjacency list representation of the graph. It then instantiates the **PreflowPush** class and calls its **getMaxFlow** method, followed by **printFlowNetworkAndMaxFlow** to display results.
2. **getMaxFlow(int source, int destination)**:
    - **Purpose**: Calculates the maximum flow in the graph using the Preflow-Push algorithm.
    - **Process**: Begins by initializing the preflow at the source node. It then repeatedly finds nodes with excess flow and either pushes this flow to adjacent nodes or relabels the node if no push is possible. The process continues until no nodes have excess flow. The maximum flow is determined by the excess flow at the sink.

3. **push(Edge edge)**, **relabel(int node)**, **overFlowNode()**, **initializePreflow(int source)**:
    - **Purpose**: Auxiliary methods supporting the Preflow-Push algorithm.
    - **Process**:
        - **push(Edge edge)**: Pushes flow from a node to an adjacent node along an edge if the capacity allows.
        - **relabel(int node)**: Increases the height of a node to allow further pushing of flow.
        - **overFlowNode()**: Identifies a node in the graph with excess flow that can be pushed or relabeled.
        - **initializePreflow(int source)**: Sets up the initial flow conditions for the Preflow-Push algorithm.

2. Data Structures Used in the Ford-Fulkerson and Preflow-Push Programs

Ford-Fulkerson Program

1. **ArrayList<ArrayList<Edges>> (Graph)**:
    - An adjacency list representation of the graph.
    - Each list within the main list corresponds to a vertex and contains a list of **Edges** objects representing the edges emanating from that vertex.
2. **Edges Class**:
    - Custom class to represent an edge in the graph.
    - Stores properties like source, destination, capacity, and flow of each edge.
3. **int[] (parent Array in BFS)**:
    - Used in the BFS method to keep track of the parent nodes for each vertex. This helps in reconstructing the path found by BFS.
4. **Queue<Integer> (in BFS Method)**:
    - A queue used in the BFS method for traversing the graph level by level.
5. **List<Integer> (for Storing Paths)**:
    - Used for storing the augmenting path from source to sink found by BFS.
6. **File and Scanner**:
    - Used for reading input data from a file.

Preflow-Push Program

1. **ArrayList<ArrayList<Edge>> (Graph)**:

- An adjacency list representation of the graph, similar to the Ford-Fulkerson program.

2. **Edge Class**:
    - Similar to the Ford-Fulkerson program, used to represent edges in the graph with additional property **reverseEdge** for the reverse edge in the residual graph.
3. **int[] (height Array)**:
    - An array to store the height of each node in the graph.
4. **int[] (excessFlow Array)**:
    - Used to store the excess flow at each node.
5. **File and Scanner**:
    - Utilized for reading the graph data from an input file.

3. Algorithms

Ford-Fulkerson Algorithm

The Ford-Fulkerson algorithm is a method to compute the maximum flow in a flow network. It is an iterative algorithm that increases the flow in the network until no more augmentations are possible. Here's a step-by-step explanation:

1. **Initialization**: The flow in all edges of the network is initially set to zero.
2. **Finding Augmenting Paths**:
    - The algorithm repeatedly searches for paths from the source (start) node to the sink (end) node such that it is possible to increase the flow along the edges of this path.
    - This search is typically performed using a Breadth-First Search (BFS) as it finds the shortest path in terms of the number of edges. This path is called an "augmenting path".
3. **Augmenting the Flow**:
    - Once an augmenting path is found, the minimum residual capacity (the smallest capacity available on an edge along the path) is identified.
    - The flow along the edges of the augmenting path is then increased by this minimum value.
4. **Updating the Residual Network**:
    - After each augmentation, the residual capacities of the edges along the path are updated. If the residual capacity of an edge drops to zero, it cannot be used for further augmentation.
5. **Termination**:

- The algorithm terminates when no more augmenting paths can be found from the source to the sink. At this point, the flow value in the network is maximized.

6. **Max Flow Calculation**:
    - The maximum flow of the network is the total flow leaving the source or equivalently, the total flow entering the sink.

Preflow-Push Algorithm

The Preflow-Push algorithm, also known as the "push-relabel" algorithm, is another method to calculate the maximum flow in a network. It differs from Ford-Fulkerson in its approach:

1. **Initialization**:
    - Each node, except the source, is initialized with a height of 0. The source node is given a height equal to the number of nodes in the graph.
    - The algorithm initializes a preflow by sending as much flow as possible out of the source along each of its outgoing edges.
2. **Push Operation**:
    - If a node has more incoming flow (excess flow) than it can send out, the algorithm tries to "push" this excess flow along one of its outgoing edges.
    - Pushing is only allowed if the height of the current node is greater than the height of the node at the other end of the edge and if the edge has available capacity.
3. **Relabel Operation**:
    - If a node cannot push its excess flow anywhere (no outgoing edge can accept more flow), the algorithm increases (relabels) the height of the node.
    - This relabeling allows more edges to become eligible for pushing in subsequent steps.
4. **Iterative Process**:
    - The algorithm iteratively performs push and relabel operations until no node (except for the sink) has excess flow.
5. **Termination**:
    - The algorithm terminates when there are no more nodes with excess flow (except for the sink). At this point, the preflow becomes a valid flow.
6. **Max Flow Calculation**:
    - The maximum flow is the total flow into the sink or equivalently, the total excess flow at the sink node at the end of the algorithm.

Comparison of Both Algorithms

- **Ford-Fulkerson** is conceptually simpler but its performance depends heavily on how augmenting paths are found. It can be inefficient for large networks with high capacity edges.
- **Preflow-Push** is more complex in implementation but generally faster and more efficient, especially for dense networks. It does not explicitly search for augmenting paths but instead focuses on local operations (push and relabel) at each node.

Ford-Fulkerson Program

4. Time Complexity

1. **Overall Algorithm**: The time complexity of the Ford-Fulkerson algorithm is O(Ef), where **E** is the number of edges and **f** is the maximum flow in the network. This complexity arises because, in the worst case, every increase in flow could be by an amount of 1, and finding each augmenting path takes O(E) time using BFS.
2. **BFS Method**: The BFS method used to find augmenting paths has a time complexity of O(V + E), where **V** is the number of vertices and **E** is the number of edges. However, since the algorithm may find an augmenting path in each iteration, the total time complexity becomes O(Ef).

5.Space Complexity

1. **Graph Storage**: The space complexity for storing the graph using an adjacency list is O(V + E), where **V** is the number of vertices and **E** is the number of edges.
2. **Auxiliary Space**: Additional space is used for the BFS queue and the parent array, which at most can be O(V).

Preflow-Push Program

4.Time Complexity

1. **Overall Algorithm**: The Preflow-Push algorithm has a time complexity of O(V^2E), where **V** is the number of vertices and **E** is the number of edges. This complexity results from the fact that each edge can be considered at most O(V) times across different height relabels, and each relabel operation can occur at most O(V^2) times.
2. **Push and Relabel Operations**: Each operation individually takes O(1) time, but the number of times they are called contributes to the overall complexity.

5.Space Complexity

1. **Graph Storage**: Similar to the Ford-Fulkerson program, the space complexity for storing the graph is O(V + E).
2. **Height and Excess Flow Arrays**: These arrays add an additional space complexity of O(V) each.