Maximum Flow Calculation Using Edmonds-Karp Algorithm Implemented in Java

1. Introduction

This implementation shows a solution to the Maximum Flow Problem using the Edmonds-Karp Algorithm in Java. The aim is to calculate the maximum amount of flow that can be pushed from a source node to a sink node through a directed graph where each edge has a defined capacity. The Edmonds-Karp approach is a specific implementation of the Ford-Fulkerson method, using Breadth-First Search (BFS) to find the shortest augmenting paths.

2. Data Structures and Algorithm

Graph and Edge Classes

Edge Class:

- Represents a directed edge with the fields which is from, to, capacity, flow, and a reverse edge.
- Provides getResidualCapacity() to calculate remaining flow capacity.
- Provides addFlow() to update the flow and maintain residual capacities.

Graph Class:

- Maintains an adjacency list structure (ArrayList<Edge>[]) for efficient storage.
- When adding an edge both a forward edge and a reverse edge are created to support the residual graph concept.

Edmonds-Karp Algorithm

- A loop repeatedly uses Breadth-First Search (BFS) to find an augmenting path from the source to the sink.
- The bottleneck capacity which is smallest residual capacity along the path is identified.
- Flow is augmented along the path, and residual capacities are updated accordingly.
- The process continues until no more augmenting paths are available.
- Using BFS guarantees that each augmenting path has the minimum number of edges.

3. Example Run

The program was tested with the input file bridge_1.txt. The output is printed to a output.txt separate file.

Output given:

```
Augmenting path : [0 -> 1 -> 5]
Bottleneck capacity: 1

Augmenting path : [0 -> 4 -> 5]
Bottleneck capacity: 1

Augmenting path : [0 -> 1 -> 2 -> 4 -> 5]
Bottleneck capacity: 1

Augmenting path : [0 -> 1 -> 3 -> 4 -> 5]
Bottleneck capacity: 1

Augmenting path : [0 -> 1 -> 2 -> 3 -> 4 -> 5]
Bottleneck capacity: 1


======= Final Flow Values =======
0 -> 1 : 4 / 4
0 -> 4 : 1 / 1
1 -> 2 : 2 / 2
1 -> 3 : 1 / 1
1 -> 5 : 1 / 1
2 -> 3 : 1 / 1
2 -> 4 : 1 / 1
3 -> 4 : 2 / 2
4 -> 5 : 4 / 4
=========================================
File tested: bridge_1.txt
```



```
Maximum Flow: 5
Execution Time: 16 milliseconds
=========================================
```

4. Performance Analysis

Theoretical Analysis

The time complexity of the Edmonds-Karp Algorithm is $O(V \times E^2)$ where V is the number of vertices (nodes) and the E is the number of edges.

Each BFS takes $O(E)$ time, and each edge can only become critical $O(V)$ times, resulting in $O(VE)$ augmenting paths, and thus $O(VE^2)$ overall runtime.

Practical Performance

Small graphs like bridge_1.txt and bridge_2.txt solve almost instantly (5-10 milliseconds).

For larger graphs execution time increases.

Measuring execution time provides useful verification of practical performance.