UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

BSc (Hons) Computer Science

Algorithms: Theory, Design and Implementation

5SENG003C.2

Coursework: Test Case Report
Maximum Flow Computation

Student Name - A.A.S. Ranasinghe
UoW Number - w2051972
IIT Number - 20231160

1. Introduction

This report is for the implementation of a maximum flow solver for the coursework in Algorithmic Analysis of Network Flow. The Task was computing the maximum flow between a source node and a sink node in a directed graph. For testing purposes, use 39 benchmark files. The implementation includes a user-friendly menu to select benchmarked files, compute the maximum flow, and output augmenting paths and flow increases.

2. Choice of Data Structure and Algorithm

Data Structure: The network flow is established using an adjacency list, which employs ArrayList and HashMap. Each node consists of a HashMap mapping destination nodes to Edge objects, which include the destination, capacity, and flow. Edge lookups and updates operate in time, making it ideal for numerous residual graph updates in the Ford-Fulkerson algorithm. Reverse edges are introduced to manage flow changes, and Java standard collections effectively leverage language APIs.

Algorithm: I implemented the Ford-Fulkerson algorithm with BFS to find augmenting paths in my implementation. The use of BFS guarantees that augmenting paths are the shortest in terms of edges. This implementation offers straightforward incremental progress for output, as required by Task 4, and is also stable for integer capacities. BFS avoids the exponential worst-case behavior of arbitrary path selection in the Ford-Fulkerson approach, making it effective for the benchmark inputs.

3. Example Run on Benchmark (bridge_1.txt)

The smallest benchmark, bridge_1.txt, has 6 nodes (0 as source, 5 as sink) and edges inferred from the output:

```
Options:
Enter a number (1-39) to select a file
Enter 'q' to quit
Your choice: 1

Processing file: bridge_1.txt
Maximum Flow: 5
Augmenting path: [0, 1, 5], Flow added: 1
Augmenting path: [0, 4, 5], Flow added: 1
Augmenting path: [0, 1, 2, 4, 5], Flow added: 1
Augmenting path: [0, 1, 3, 4, 5], Flow added: 1
Augmenting path: [0, 1, 2, 3, 4, 5], Flow added: 1
Maximum flow: 5
```

The total flow is 5, achieved over four augmenting paths. The first path, [0, 1, 5], sends 1 unit (bottleneck: 1→5 capacity 1). The second path, [0, 1, 4, 5], sends 1 unit (bottleneck: 1→4 or 4→5 capacity 1). The third path, [0, 1, 3, 4, 5], sends 1 unit (bottleneck: 1→3, 3→4, or 4→5 residual capacity 1). The fourth path, [0, 1, 2, 3, 4, 5], sends 1 unit (bottleneck: 1→2, 2→3, 3→4, or 4→5 residual capacity 1). No further augmenting paths exist, confirming the maximum flow of 5.

4. Performance Analysis

Theoretical Analysis: The Ford-Fulkerson algorithm with BFS has a time complexity of $O(V \cdot E^2)$, where:

- V is the number of nodes.
- E is the number of edges.
- BFS: O(E) to explore all edges.
- Path processing: O(V) to update flows along the path. Therefore, the total time is $O(V \cdot E)$ iterations × O(E) per iteration = $O(V \cdot E^2)$. The space complexity is O(V + E) for the adjacency list and O(V) for BFS (queue and parent array), totaling O(V + E).

Methodology: The $O(V \cdot E^2)$ complexity arises because BFS ensures the shortest augmenting paths, reducing the number of iterations compared to arbitrary path selection. The benchmark networks are likely sparse, making the algorithm efficient. The implementation is optimized with O(1) edge lookups via HashMap, and the menu system adds negligible overhead.

Big-O Classification: The order-of-growth is $O(V \cdot E^2)$, as derived above. This is appropriate for the coursework inputs, ensuring efficient computation and clear incremental outputs for analysis, meeting the requirements for both performance and documentation.