

INFORMATICS INSTITUTE OF

TECHNOLOGY

In collaboration with

UNIVERSITY OF WESTMINSTER

Algorithms

5SENG002C

Coursework

Module Leader’s Name – Mr. Sudharshan Welihinda

Dinuka Piyadigama

UoW ID – 17421047

IIT ID – 2018373

# Algorithmic Approach taken

## Algorithmic Strategy

* Ford-Fulkerson algorithm was used to calculate the max flow of the flow network.
* Breadth First Search (BFS) was used to find whether a path exists from source to sink. The reason for choosing BFS was because BFS always picks up the path with the minimum number of edges. The worst-case time-complexity can be reduced as well.

## Chosen Data Structure & its Traversal Towards Solution

* A LinkedList (queue) has been used for the queue that is created in the Breadth First Search (BFS) method. In BFS *poll* method of the LinkedList was used to return the first element of the queue and remove it from the queue.
* I have used a 2-dimensional array ([][] graph) to represent the flow network’s graph as a matrix. The 1st index of the array gives the starting node, 2nd index gives the ending node of a link. The value at the 2nd index gives the capacity from the starting node to the ending node. If there is a capacity, a link exists between the two nodes.
* An array ([] parent) was used to store the residual path in BFS.
* When taking inputs from the user a HashMap was used instead of an ArrayList to get inputs because then, the order of entering inputs won’t matter. This was implemented for the ease of coding.

## Pseudocode in plain English

BEGIN

INPUT graph, source, sink of flow network

Initialize the Residual graph from the initial graph and the parent array

max\_flow = 0, max\_integer\_value = 2147483647

WHILE there’s a path from source to sink:

path\_flow = max\_integer\_value

path\_flow = MIN(path\_flow, capacity\_of\_residual\_link)

max\_flow = max\_flow + path\_flow

END WHILE

DISPLAY max\_flow

END

# Methodology for empirically analysing the performance of the algorithm

|  |  |  |
| --- | --- | --- |
| Input data size | | Time spent to produce the outcome  (in nanoseconds) |
| Nodes | Links |
| 6 | 10 | 6248500 |
| 12 | 20 | 10739800 |
| 24 | 40 | 15172100 |
| 48 | 80 | 20936200 |

# Conclusions algorithmic performance