INFORMATICS INSTITUTE OF TECHNOLOGY

UNIVERSITY OF WESTMINSTER

Informatics Institute of Technology

Department of computing

BSc Computer Science

Module: 5SENG002C Algorithms Theory Design & Implementation

Coursework

Date of submission	7th April 2021
Student ID	2019163
Student UoW ID	W1761196
Student First Name	M. Ammar
Student Surname	Raneez
Student Group	A

Task 5

a)

The data structure implemented was technically a directed weighted Graph – that will be used to model a Flow Network.

A Graph would be the easiest and most efficient way to model vertices and a set of edges connecting those vertices. Since it is required that a Flow Network be created using input files that have vertices and edges, the chosen way to model this was through a graph, where it would hold the vertices and the edges each of them is associated with.

Flow Edges were initially created that will be used to represent the edges in the implemented Network. Since, in this Network, there are weights as well (capacities) the flow edge API had to contain along with the vertex from, vertex to, the capacity of the edge.

The Flow Network data structure was implemented, that will hold the flow edges. The structure used to hold these edges is an implementation of an Adjacency List. The reason for using an Adjacency List, instead of an Adjacency Matrix, is that for very large graphs, the space complexity would get too large, and possibly overflow – due to the matrix taking $vertices^2$ space. For instance, a billion number of vertices, would take $billion^2$ space, which might be too big for a computer to hold. The adjacency list would be a vertex-indexed List of FlowEdge Lists. Where each inner list would hold the edges, each vertex is part of. An inner list was used since the edges will be added periodically.

The algorithm implemented used to obtain the Max Flow is the "Edmonds Carp" Algorithm, that is a specialized implementation of the "Ford Fulkerson" method, using Breadth-First Search to obtain the Augmenting paths. Breadth -First search finds shortest path, thereby removing the chances of very large number of augmentations (for instance if an edge has a capacity of 1, while the others have a capacity of 1000).

The algorithm is run, while there are remaining Augmenting paths. The augmenting paths are found using a Breadth-First Search as mentioned-above. This method marks the vertices that it has visited as well as the edge it used to reach the vertex, using the edgeTo[] array. The Ford Fulkerson Algorithm then uses this array to retrack the path, to obtain the bottleneck capacity (the maximum possible flow that can be passed at that point). This obtained value is then added to the used edge's flow value, as well as the max flow value.

The algorithm terminates if there are no more augmenting paths, what this essentially means is that the Breadth-First Search algorithm got stuck, while searching for a path from source to target, this happens when there is no path left, or in other words, the available forward edges in a path have reached their full capacities or the backward edges have become empty, not allowing any more residual flow.

b)

Flow Network: Edges = 5, Vertices = 4

Augmenting Path: 1

Bottleneck Capacity: 3

Flow Value Incrementing from: 0 to: 3

Augmenting Path: 2

Bottleneck Capacity: 4

Flow Value Incrementing from: 3 to: 7

Augmenting Path: 3

Bottleneck Capacity: 1

Flow Value Incrementing from: 7 to: 8

Max Flow determined: 8

c)

Using the Doubling Hypothesis– Recording times for double the number of edges each time:

Hypothesis – The running time of the program is $T_N \sim a N^b$.

Consequence – As N increases, $T_N / T_N/2$ approaches 2^b.

Proof - $\frac{a(2N)^b}{a(N)^b} = 2^b$

Number of Edges	t_1	t_2	t_3	T_N	$T_N / T_N/2$
189 (ladder_5.txt)	0.03	0.03	0.033	0.031	
381 (ladder_6.txt)	0.10	0.11	0.11	0.11	3.5
765 (ladder_7.txt)	0.32	0.33	0.32	0.32	2.9
1533 (ladder_8.txt)	1.14	1.44	1.13	1.24	**3.9**
3069 (ladder_9.txt)	5.05	5.20	5.11	5.12	**4.1**

Based on consequence:

$2^b \sim T_N / T_N/2 \sim 4$

$2^b \sim 4$

$b \sim 2$.

Substituting the value of b into the hypothesis we get:

$T_N \sim a N^2$

For a function $f(N) \sim ag(N)$, g(N) is considered as the order of growth of the function.

$T_N \sim a N^2$

Therefore, the order of growth of the program is $\sim N^2$.

According to the doubling hypothesis and the results obtained, the **Order of Growth of the Program is $O(N^2)$**

Appendix – Detailed Example Output

Flow Network: Edges = 5, Vertices = 4

Before Adding Residual Flow: Flow Edge: 1->3, Flow: 0, Capacity: 3 || After Adding Residual Flow: Flow Edge: 1->3, Flow: 3, Capacity: 3

Before Adding Residual Flow: Flow Edge: 0->1, Flow: 0, Capacity: 6 || After Adding Residual Flow: Flow Edge: 0->1, Flow: 3, Capacity: 6

Augmenting Path: 1

Bottleneck Capacity: 3

Flow Value Incrementing from: 0 to: 3

Before Adding Residual Flow: Flow Edge: 2->3, Flow: 0, Capacity: 5 || After Adding Residual Flow: Flow Edge: 2->3, Flow: 4, Capacity: 5

Before Adding Residual Flow: Flow Edge: 0->2, Flow: 0, Capacity: 4 || After Adding Residual Flow: Flow Edge: 0->2, Flow: 4, Capacity: 4

Augmenting Path: 2

Bottleneck Capacity: 4

Flow Value Incrementing from: 3 to: 7

Before Adding Residual Flow: Flow Edge: 2->3, Flow: 4, Capacity: 5 || After Adding Residual Flow: Flow Edge: 2->3, Flow: 5, Capacity: 5

Before Adding Residual Flow: Flow Edge: 1->2, Flow: 0, Capacity: 2 || After Adding Residual Flow: Flow Edge: 1->2, Flow: 1, Capacity: 2

Before Adding Residual Flow: Flow Edge: 0->1, Flow: 3, Capacity: 6 || After Adding Residual Flow: Flow Edge: 0->1, Flow: 4, Capacity: 6

Augmenting Path: 3

Bottleneck Capacity: 1

Flow Value Incrementing from: 7 to: 8

Max Flow determined: 8