Algorithmic II Assessed Exercise

2581690b - Joseph Brown

Semester 1 - November 2023

1 Part A, Task 1

public boolean isFlow()

Flow in a directed graph is a function f: E $\rightarrow R$ such that:

- Capacity constraint: for every edge, 0 $\leq$ flow $\leq$ capacity
- Flow conservation constraint: for every vertex other than the source and sink, total incoming flow = total outgoing flow

where E denotes an edge in the graph.

Th Boolean isFlow is initially set to true, this is updated or preserved through the function based on the previous constraints for a valid flow.

The array edges stores all edges adjacent to the source. These form a basis to process all the edges and subsequently all the vertices in the graph.

The for loop iterates through all existing edges in the network and verifies the capacity constraint.

First quickly check that there is no incoming flow to the source vertex and no outgoing flow from the sink vertex.

Finally the remaining vertices and edges are checked, when the vertex is an source vertex the flow is added to outFlow and when the vertex is a target vertex the flow is added to inFlow. If these two integers are equal then the flow conversation constraint is met, hence the flow is valid.

public int getValue()

The flow conversation constraint means that the total flow leaving the source is conserved throughout the network, hence the total flow is deduced by summing the values of the flow along the edges leaving the source node.

The variable value stores an integer which denotes the accumulated value of the flow, and is initialised as 0.

The array edges is used to store the edges of those which are adjacent to the source vertex.

By iterating through these edges; the value variable is updated, sums their individual flow to return the total flow, which corresponds to that of the entire network.

public void printFlow()

The HashSet stores all the edges for a given flow in the network, and is initialised to be empty.

An outer loop iterates through all vertices in the network and the inner loop iterates through the linked lists which are all the adjacent vertices for the given vertex. The 'currentEdge' variable is an edge which stores the edge adjoining the adjacent vertices found by performing the previous iteration.

The edge is checked not to be null and or present in HashSet.

2 Part A, Task 2

public ResidualGraph (Network net)

By inheriting numVertices and net.adjMatrix, the residual graph has the same vertices as the network

Forward and backward edges are formed based on the following criteria:
(u,v)$\in E'$, where E' is an edge in the residual graph

- $(u, v) \in E$ and $f(u, v) < c(u, v)$, so (u,v) is a forward edge and $c'(u, v) = c(u, v) - f(u, v)$

  OR

- $(u, v) \in E$ and $f(u, v) > 0$, so (u,v) is a backwards edge and $c'(u, v) = f(u, v)$

public LinkedList<Edge> findAugmentingPath ()

The variables are initialised as such; currentVertex stores the source vertex, augmentingPath stores the augmentingPath to be returned if found, predecessors keeps a pointer to the previous vertices used for backtracking, finally foundTarget is used to flag when the target vertex is found.

A BFS loop is used to search for an augmenting path which terminates when the sink is reached.

The augmenting path is then built and returned.

public void augmentPath (List<Edge> path)

This function first finds the minimum capacity among the edges in the path then iterates through the path updating the flow.

public void fordFulkerson ())

A residual graph is formed based on the network, where an augmenting path is found if it exists.

Finally the the augmenting path in the residual graph is augmenting in the network.

3 Part B, Task 3

The text file is read and used to adapt the graph, where all are represented as vertices. The basic layout consists of a source connected to the students with an edge of capacity 1, these are then linked to projects. However if the student is doing software engineering but has chosen a project that isn't software engineering an edge isn't added. The project capacity are given in the text file and are used as the edge capacity linking the projects to lecturers, and finally a similar logic is applied to connect lecturers to the sink where the edge capacity in this case is the lecturers capacity also given in the text file. A student is only assigned a project when flow is greater than 1, hence the Ford Fulkerson algorithm gives the maximum matching of students to projects.

4 Part C, Task 4

I wasn't able to complete Part C, however lower bounds could be implemented for each lecturer then change the constructor for the residual graph to accommodate for this change. In a similar approach to Part B.