**Algorithms 2 Assessed exercise report**

Task 1

Status report

Functions as specified.

Justification of operation

isFlow checks each edge to see if the flow is below the capacity, then checks each vertex to see if the incoming flow is equal to the outgoing flow except for the source and sink vertices which respectively have no incoming and no outgoing flow.

getValue relies on the fact that the total flow of a network is equal to the sum of the flows on the efferent edges of the source, so it simply calculates this.

printFlow simply loops over all the edges via the adjacency matrix and prints out the labels, capacity and flow of each edge.

Task 2

Status report

Functions as specified.

Justification of operation

The ResidualGraph constructor builds the residual graph for a network by checking each edge. A backwards edge will be added if there is a possibility of reducing the flow through the edge and a forwards edge will be added if the edge still has capacity for more flow.

findAugmentingPath looks for a path connecting the source to the sink in the residual graph. If such a path exists, it finds it via a breadth first search. This is accomplished using a queue which will take as first input the vertices adjacent to the source and check if any are the sink vertex. If so then the path is found, otherwise a vertex will be removed from the queue and its adjacent vertices added and checked. This process will continue until the sink is found. An array of predecessors for each vertex is used to reconstruct the path and to ensure there is no useless searching by eliminating any path which would go through a vertex which already has a predecessor. A stack is used to restore the path as looking through the predecessors array will give the path in reverse order. The path is then reassembled in the correct order by popping elements from the stack into the path list.

augmentPath calculates the maximum flow to increase by through checking for the limiting flow in the input path. It then adds this flow to forwards edges and removes it from backwards edges.

fordFulkerson keeps trying to augment the network by creating the residual graph, then finding an augmenting path and augmenting along it until no augmenting path can be found.

Task 3

Status report

Functions as specified.

Justification of operation

The graph was adapted to match students, projects and lecturers. All of these were modelled as vertices in the network. The source connects to all students with edges of capacity of 1. The students are connected to all their chosen projects with edges of capacity 1. If an SE student has chosen a non-SE project, then that edge is not included in the graph. The projects are connected to their supervising lecturers with edges of capacity the project’s capacity. The lecturers are connected to the sink with edges of capacity the lecturer’s capacity. A project will be assigned to a student if both vertices are on a path from source to sink with flow > 1. Since the edges from source to student have capacity 1 any path that augments the network will assign a new student to a project. Therefore, if we apply the ford Fulkerson algorithm to this constructed network it will give us the maximum matching of students to projects. Neither the projects nor the lecturers will not go over their capacity limit due to the capacity on their connecting edges.

The readNetworkFromFile method parses all the input into this above format while also adding in the illegal (SE student -> non-SE project) edges. It remembers which students and projects are SE in an array, then after reading the whole file it effectively removes them from the network by setting their capacity to 0.

Task 4

Status report

Functions as specified.

Justification of operation

To accommodate for the lecturer lower quotas the algorithm was changed so that it augments the path for a lecturer until that lecturer has their lower quota filled and repeats this for all lecturers. After doing this it augments the graph normally. This will not cause a problem since the augmenting path will never lower the flow from an edge afferent to the sink so once the lecturers’ lower quotas are reached, they will never fall below again. Indeed, if an augmenting path were found which lowered the flow from the sink to an afferent edge, then this path would not augment the flow at all since the flow along the path is constant. Such a path would instead keep the flow constant which is not an augmenting path.

This was accomplished by modifying the residual graph constructor so that if a lecturer with unmet quotas still existed it would remove all other lecturer – sink edges and therefore force the network to be augmented along that edge.

Information is passed between the parsing class (FordFulk) and the residual graph constructor via a HashMap of lecturer label to lecturer lower bound.

The respectsLowerBounds method was added to check the validity of the flow in terms of the lecturer lower quotas being met which checks the flow along the lecturer – sink edges against the lecturer lower quotas.