A4 – Outline of Approach

The core of my approach to using Edmond-karp Max-flow algo to solve the biparte matching problem focuses on the set up of my graph. I used an adjacency list graph made using a hashmap with sources as keys and arraylists as the values. I added a source node to the graph (src) which had edges of weight 1 directed to each of the 9 pilots (if they were included as qualified pilots). Each outgoing edge from each pilot points to a vertex representing the flight(s) the pilot is qualified for, also with weight 1 as a pilot can only fly once per day. The final part of the setup was to add the edges from the flights to the sink vertex. These edges were given a weight of the number of times that flight needs to be flown. So if the input was,

A4 012345

Q1 5

B4 6789

The source would point to vertices 0-9, which would point to their respective flights (ex. 0 -> A), and the flights A, Q, B would have edge of weight 4, 1, 4 to the sink vertex.

Edmonds-Karp:

The Edmonds Karp algo takes the residual graph (which, at the start, is just the graph), the source vertex, and the sink vertex. I started by creating a hashmap called “previous” to track the augmenting paths that get stored during bfs. Next is the main loop while will repeat until there are no more augmenting paths in the residual graph. After that the algo gets the maxflow through the augmenting path by getting the capacity of each edge in the path and updating the maxflow if the capacity is smaller than the other capacities, eventually finding the bottleneck capacity. Next, the algo updates the weight of each edge to subtract the pathflow from the capacity and adds the reverse edges with the pathflow as the weight. The final step is adding the pathflow to the maxflow.

Output:

The Edmond Karp algo returns the maxflow which is then compared with the sum of each flight that needs to be taken that day (the “4” from A4). If they are not equal we know the biparte problem has no solution and can print “!”. If they are equal, we loop through an array of flights (created during graph setup) and get ary outgoing edges with weight 1 from the flight vertecies. Since the residual graph will only have reverse edges with weight 1, the destination of that edge will be the pilot flying that flight and the flight can therefore be

added to the output at the position corresponding to the pilot. The output array has been initialized with "_" so if any pilot does not fly the corresponding position will be empty.

Example Drawing: