Jason Caplan
Design and Analysis of Algorithms
Professor Leff
HW15

1. We can model both students and chaburos as nodes, treating the assignment of each student to a chabura as a flow of value 1 from student node to chabura node. Furthermore, we can capture the capacity of each chabura p using the flow-capacity from each chabura to the sink node, which also ensures that no more than p students will flow to this chabura (by the equilibrium constraint). Additionally, edges of capacity 1 point from each student to his top 2 choices of chabura, and to ensure that only 1 is filled with flow, we point exactly 1 edge of capacity 1 from the source node to each student node, so that only 1 unit of flow can possibly exit the student node (by the equilibrium constraint).
2. As explained above, we have a source node that points 1 edge of capacity 1 to each of the student nodes (A-F), and each student has 2 capacity-1 edges to his top 2 chaburos (e.g., A points to X and Y, D points to Y and Z). Finally, each chabura node has an edge to the sink node with capacity $p = 2$. (The nodes are circles; the edge weights are rectangles).

3. If $maxflow == n$, there exists an arrangement wherein each student is assigned to one of his top 2 chaburos. This is because if this holds, then all students have been assigned, and all assignments can be made in accordance with the student-chabura edges that hold flow in the $maxflow$.
4. The worst case is finding n paths, since in the worst case, n is the maximum flow (by construction).

5. Instead of 2 capacity-1 edges leaving each student node, we'd make k such edges, from each student to his top k chabura choices.
6.
    a. We use the same general idea as above, but we populate the network flow graph such that there are edges from each student to every single chabura on his preference list. Also, instead of using vanilla BFS or DFS to find augmenting paths, we prioritize preferred choices, by essentially going through the input array of student preferences column-wise first (i.e., we first add paths that contain each student's top choice, then the paths that contain each student's second choice, and so on). We keep a counter c that is initialized to zero and is incremented each time we successfully find an augmenting path. We break once the flow reaches $maxflow$, i.e. when $c == n$, and return the number of the column we are up to in our scan of the input array (1-indexed), because this number represents the lowest ranking we had to allow in order to find a suitable arrangement (e.g., in the example input given, if we just added the path going through B and X when we reached the exit condition, we are in the second column of the input array, so the answer would be $k = 2$).
    b. All in all, the runtime will be the same as having known k in advance and having run the algorithm described in the original problem, because we've only added $O(1)$ checks and operations to the vanilla algorithm. The only additional performance issue is that the graph construction requires that we add an edge for every possible student-chabura pair, for a total pre-processing time and space requirement of $O(nm)$.