Time Spent: 2 hours

Collaborators and Resources: Discussed how to make flights chronological with Nathaniel MacArthur-Warner, also used the Ford-Fulkerson algorithm described in Chapter 7.1

Problem 1

Here we're given a problem in which we have to coordinate flight schedules for students and faculty. Everyone must begin in Northfield and end in Orlando, FL via a series of flights and (instantaneous) transfers between flights. Because of this we can conceptualize the flight schedules in the format of a graph wherein vertices are locations and the edges represent students and faculty going on and between flights. Specifically, a student in this problem can be represented as a unit of flow such that when we maximize the flow we are able to maximize the number of people who make it to GHC. Furthermore it seems like the flights are not cyclic (that is, they only occur once) and so maximizing the flow through the graph would be exactly equivalent to maximizing the number of people who make it from Northfield to Orlando. In this sense the amount of time spent total flying per person varies but this does not matter (bar exceptional circumstances) so we can be assured that they will make it on time. So my algorithm constructed for this problem is as follows below ...

1. Given a schedule F of n flights available where each entry $F_i = (s_i, t_i, d_i, a_i, c_i)$
2. Transform the schedule into a flow network (a directed graph G = (V,E)) as follows
   (a) Specifically, let the source s be Northfield and the sink t be Orlando, FL
   (b) For all flights F_i, i $\in$ {1, ..., n}
      i. Add s_i and t_i to V if they are not already there

ii. Add the edge (s_i, t_i) to E and set its capacity to c_i, the number of seats available on the flight

(c) For all pairs of flights F_i, F_j where i $\neq$ j, i,j $\in$ {1, ..., n}

i. If $a_i \leq d_j$ then add the edge (s_i, t_j) to E and set its capacity to the total number of students and faculty. # Essentially, if it's possible to transfer within an airport then create an edge to allow this to occur.

3. Now we need to run the Ford-Fulkerson algorithm which maximizes the flow

4. So initially let f(e) = 0 for all edges e $\in$ G

5. While there is an s-t path in the residual graph G_f of a flow

(a) Let P be a simple s-t path in G_f

(b) Let b = the minimum residual (left over) capacity of any edge on P

(c) For each edge (u,v) $\in P$

i. If $e = (u, v)$ is a forward edge then increase $f(e)$ by b

ii. Else (e = (v,u) is a backward edge) then decrease $f(e)$ by b, thereby updating the residual graph G_f

Claim 1. My algorithm outlined above generates the correct graph and thereby maximizes the flow in $O(n^2 \cdot C \cdot n^2)$ time, where m represents the number of edges, n represents the number of vertices and C represents the total sum of all capacities out of s (that is, the total number of students and faculty involved)

Proof. My justification for the algorithm's correctness is as follows. Flow is defined in integer values in terms of each person. It is impossible for each flight (represented by an edge) to contain more people than the number of available seats allows because we set the capacity to be c_i. Furthermore, if it is not possible to transfer between flights because the arrival date for the first flight is after the departure date for the next flight, then we don't draw an edge and thus the students and faculty cannot incorrectly transfer to another

flight. However, if it is possible to transfer, so by the direct transformation of the schedule into the graph G we are able to show that the graph should be a correct demonstration.
Furthermore, the Ford-Fulkerson algorithm is known to maximize the flow from s to t that is possible across the entire graph. When it returns a value, we know that this is a valid flow for the graph we have created. If we convert the corresponding sub-flows' integer values back into number of people, we can see that the number of people on any flight does not exceed the limit and that people do not transfer between flights unless it is possible to do so (if they arrive on time). Basically, we know the algorithm is correct by the fact that it is a valid representation of the schedule.
The Ford-Fulkerson algorithm we know to run in O($m \cdot$ C) time (K-T (7.5)). We see that we add an edge for each i in 1:n, and then add another edge (potentially) between each flight in the next for-loop. So since we are considering an upper bound, we say that m $\leq n^2$ so the Ford-Fulkerson algorithm really runs in O($n^2 \cdot$ C) time.
Also, the run-time is multiplied by the number of vertices n three times. This is first because we have to add $2 \cdot$ n vertices to the graph, and secondly because we have to add one edge to the graph n times and then another edge (potentially) to the graph n times in a separate for-loop. The constant 2 evens out over large values of n, so we end up multiplying by n three times. That is why we also multiply by n^3. While the algorithm may not be the most efficient it works on the graph described. □