COP 4516 Spring 2026 Week 12 Individual Contest #5
Network Flow - Solution Sketches

Who Stole My Burrito?

Just simulate the given process. For each person in line, subtract the number of chicken and steak cubes. All you have to remember is that this difference can't go below 0. So, if there are 8 chicken cubes left and you want 10, you only get 8 of them. After that everyone else after you gets 0. It's completely permissible for the data to have cases where both the chicken and steak run out well before the last person gets to the front of the line.

Editor Navigation

There may be an urge to try a greedy strategy, but ultimately what you realize is that there are several options for moves and the lengths of lines can create some interesting cases that thwart most greedy approaches. A better approach is to simply realize that we can use a breadth first search to map out all the possible cursor positions we can reach with 1 move, 2 moves, 3 moves, etc. from our initial cursor position and we can continue searching until we get to our destination. The hardest part of this problem is properly encoding all of the possible moves. Each of the four moves needs special code and can't easily be taken care of with a DX/DY array. Storing a location is fairly simple: each location is an ordered pair of line number and column number. There are few enough of these that a BFS runs in time.

Shuttle Routes

We need to find the shortest distance from several different locations to one particular endpoint (campus). While Floyd-Warshall's solves this problem, the input allows for too many vertices for Floyd-Warshall's to run in time. The key observation is if we run Dijkstra's using the supposed endpoint as the source instead, since all of the edges are reversible anyway, this single run of Dijkstra's will provide the answer to all possible queries and easily runs in time. In addition, a single run of Bellman-Ford from the destination to all other locations will also run in time. Even if the edges were directed, this solution idea still works: just reverse the direction of all edges.

Welcome Party

One possible solution (and my intended solution) is to notice that we can create a bipartite graph by having nodes for each first name starting letter and each last name starting letter as our two sets. Connect a source to all the first name starting letters with capacity 1 and edges from all last name starting letters to a sink with capacity 1. Then connect edges with capacity one for each name going from the starting letter of the first name to the starting letter in the last name. A maximum flow in this graph is proof that each of the people in the set of the max flow edges between the two groups must have their own separate "team". (For example if E→C, J→P and S→M, this is proof that EC, JP and SM must all be on different teams as no pair of them share either a first or last name starting letter in common.) It follows that the answer to the query is the maximum flow in this graph. The original problem authors also intended an alternate solution to work: by limiting last names to 18 possible starting letters, one can try all possible 2^{18} subsets of teams based on the last names. For each of these subsets, it's easy to determine which first name letter teams are needed. (Basically, once we know the last name teams, find all names whose last names can't be included in any group, then for this leftover group, find how many unique first name starting letters there are. This solves the query for one specific subset of last name letter groups. Iterate through all possible last name letter groups and take the least. The run time of this algorithm is 2^{18} (subsets) x 300 (names) ~ 150,000,000 which is pushing it, but just barely within the realm of acceptable runtime.