COP 4516 Spring 2024 Week 9 Team Contest #1 Solution Sketches

Alphabet Soup

In this question we have two sets of letters, red and green and we are trying to match red letters with green letters, but not all of the red letters match with all of the green letters. Keeping all of this in mind, this is exactly the bipartite matching problem. (You can think of the red letters as students looking for internships and the green letters as the internship openings.) In this problem, instead of explicitly giving you which letters can be matched with which other ones, there's a simple rule: the difference in the letter values must be at least three. So, we create a network flow graph, source goes to each of the red letters with capacity 1, the red letters link to each green letter that isn't within 2 of its value with capacity 1 and then the green letters link to the sink, each with capacity 1. The max flow through this graph represents the maximum number of coffee mugs we can create.

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.

Minesweeper

No recursion is necessary here since you aren't implementing a recursive clear. Instead, you just go through the grid, looking at each '.' square. For each of these, just do a loop to all neighboring squares via a DX, DY arrays for the eight possible directions, counting the number of stars around you. Then just store this result. You just have to watch out for array out of bounds and taking care with char vs. int issues since the board might be a 2D array of characters and you might want to store the character version of each number.

The Next Permutation

Here is an example of the algorithm running on the following input:

2381356998842111

Scan from the right to the left until you find indexes i-1 and i such that s[i-1] < s[i]. in this case this happens at the substring "69"

238135 | 6 | 998842111

It's clear that there is no permutation of the last substring that has higher value. This means that the 6 has to change and it has to change to a value greater than 6, but as small as possible. Since we want the 238135 to be fixed, we exchange the 6 with the last character that is greater than 6 in the substring after it. So, in this case, we exchange the 6 with the second 8, which is right before the 4, which is too small. This gives us:

238135 | 8 | 99864211

Now, of the digits in the last substring, we want the smallest permutation, which can be achieved by reversing this list:

238135 | 8 | 11246899

It should be fairly easy to see that these steps always work - we want to identify the last descending substring as what can't be incremented, which means that the character before must be incremented. Then, we want this character to go up by as little as possible without changing the substring before it. Finally, we want the rest of the items to be in increasing order so that they form the first possible permutations of those values.

Scientist

We want to match scientists to virus outbreaks, and want the maximum matching. Thus, we can set up a network flow graph to solve the problem. We create a vertex for each scientist and each virus outbreak. We add an edge from the source to each scientist with capacity 1. We add an edge from each virus outbreak to the sink with capacity 1. Then, we add an edge between each scientist and virus outbreak, if that scientist can extinguish that virus outbreak. In order for a scientist to be able to extinguish a virus outbreak, the scientist must be able to reach it, AND that virus must be on its list of viruses that scientist can extinguish. We can check the former via BFS or DFS (the most we have to run is 26 from the scientists, which will run pretty fast), and the latter is just a lookup, probably most easily implemented via a Boolean array or bitmask.