3rd Annual

Brookfield Computer Programming Challenge

2019

Editorial

Balloons

Since there are only 99 balloons, we can calculate the position of each balloon at time T (balloon i will be at position T-i, or it won't have been released yet if T-i is negative). We now just need to check if T-i is >= L and <= H, and count the number of balloons for which that is true.

Frenemies

This was a tricky problem. First, notice that if there is a group of n people who are frenemies, this forms n*(n-1)/2 pairs in total. Each time two people talk, we need to know if they are in the same group. If they are, we can ignore the query since the answer doesn't change. If they aren't, we can first subtract out the contribution of each of their original groups, then merge the two groups, then add back in the contribution of the new group. The contribution of a group of size n is just n*(n-1)/2 as discussed above.

Merging groups is the tricky part. We can't just assign the group number of everyone in the first group to the second group every time. If the first group has n people, each of these people might have their number reassigned m times; this would lead to a runtime of roughly n*m which is almost 10^10. Instead, we can always merge the smaller group into the bigger group. When we do this, we know the size of someone's group will double whenever they are reassigned. This can only happen less than 20 times if there are only 3*10^5 people (because 2^20 ~= 10^6). Therefore, by merging the smaller group into the bigger group, our runtime is n*log(n), which is about 20*3*10^5, which clearly runs in time.

Frosty

A key observation is that there is no need to bother any of the elves, since you have your own refrigerator. For every time you need to wait in an elf's house, you could have just waited in your own house in the beginning. Now it is clear that you need to be -n degrees (or colder) before moving. The amount of time it takes is then (max(n-|D|, 0) + n), since you need max(n-|D|, 0) steps to get to the correct temperature, and then n steps to move to Jingle's house.

Linus

This was another tricky problem. For every pair of starting indices i and j, we need to know the smallest x for which either (string[i+x]!=string[j+x]) or max(i+x, j+x) == lengthOfString. If we can figure this out quickly, we can sum these for every starting i and j to calculate the expected play time.

We can calculate this match length using dynamic programming. Our state is the first and second index. If either of these indices is the last index in the string, the answer is 1. Otherwise, if they are different, the answer is also 1. Otherwise the answer is 1+go(i+1, j+1). If we store the answer for each possible input as soon as we calcuate it, we will only hit each state a constant number of times, and there are only n^2 states. This gives us a final runtime of 3000^2, which runs in time, as opposed to the naive 3000^3 which TLE's (Time Limit Exceeded).

We can also easily calculate GCD with Euclid's GCD algorithm.

Planes

Two observations required: first, it requires n*(n-1)/2 unique edges for the answer to be 1; second, the answer can always be 2 if it can't be 1. For the first observation, in order for the farthest pair of airports to only be one flight away, that means every pair needs to be directly connected. This happens if there are n*(n-1)/2 unique flights. For the second observation, we can use our flights to connect every airport to airport 1. Then every pair is at most distance 2 away: on their first flight they can fly to airport 1, then have a single layover, then fly to their destination.

To calculate the number of unique flights we are given in input, we can use a hashset, or a treeset, or sort the flights somehow and check the number of adjacent pairs that are different.

Snowflakes

Interestingly, this graph theory problem didn't really require any graph theory programming (like a BFS/DFS/et cetera), just some intuition. Let's count snowflakes by their center (i.e. the one node that doesn't have degree 1). Some node is a center iff it does not have degree 1, and if all of the neighbors that it has all do have degree one. If this is the case for a node, we can increment a counter for the number of snowflakes of this size. Then, at the end, we can check how many of our counters are equal to exactly 1, and that is our answer.

Three

There are many approaches that work for this problem. One involves making a shape that looks like one of these, depending on whether the number of vertices was even or odd:

Windows

Notice that if there are D digits that are illegal, there are 10-D digits that are legal. In other words, we need to represent n in base (10-D), and then rename the digits once we are done. Writing numbers in different bases is a common APCS trick that comes up on the AP exam. It can be done as follows:

```
while (n!=0) {
        digitToAdd=n%base
        n/=base
        //Add digitToAdd as the next most significant digit to the answer
}
```

The only difference with this problem is that digitToAdd would need to be converted to the corresponding legal digit.