LCC Takeup

By Max, Peter, Kenneth, and ChrisT

LCC ‘20 Contest 1 J1 - Remembering Ratings

Frequency Array - an array storing how many times each element occurs

Take the input and maintain a frequency array.

Select the most frequent element with the lowest rating from the array.

Frequencies	[1,	1,	1,	1,	2]
Ratings	1	2	3	4	5

LCC ‘20 Contest 1 J2 - Estimating Marks

Maintain two variables, one that stores the marks she has earned and the other that stores the total number of marks on the test.

Make sure to use a floating point number.

We calculate Nayaab’s mark through marksEarned/totalMarks.

LCC ‘20 Contest 1 J3 - Contact Lists

We will utilize a data structure called a hashtable/map, which will store key-value pairs. Since this data structure is used very often, many programming languages have built in implementations, map in C++, HashMap in Java.

By using two of these, one for mapping each name to the respective phone number and the other mapping the phone number of the name, for each query we can look in the appropriate map for the answer.

LCC ‘20 Contest 1 J4 - Tracy and Cows

For the first subtask we can just loop through all of the positions of brown cows for every black and white cow. Note that because of the condition that Tracy can magically teleport between either two rows or two columns, we only need to consider the difference between the x or y values, depending on which way we teleport.

For the full solution, reduce the problem to finding the 2 closest values from 2 lists. We can solve this by using binary search. Alternatively, there is a two pointers method.

LCC ‘20 Contest 1 J5 - orz Alan

Notice that for subtask 1, $0 \leq N, R, x_i, y_i \leq 100$. This means that we can store a 2D boolean array of all lattice points. For every circle, we set all the lattice points inside the circle to true. This can be done using the distance formula.

For the last 2 subtasks, we use a line sweep approach. For every x_i that is contained by a circle, we store a list of intervals that represents the starting and ending y-coordinate representing the range of y-coordinates that contains a valid lattice point. Finally we loop through all x-coordinates and merge all the intervals to get the final answer.

Memory optimization may be required to pass the last subtask. Because maps tend to use lots of memory, coordinate compression is a possible optimization that can decrease the memory usage.

LCC ‘20 Contest 1 S1 - Cheetahs

Observe that a has reported their score correctly if all the scores they have reported are the same, and they reported their score at least once.

This means that all a student did not report their score correctly if either they didn’t report any score, or they reported multiple different scores.

LCC ‘20 Contest 1 S2 - Spellcheck

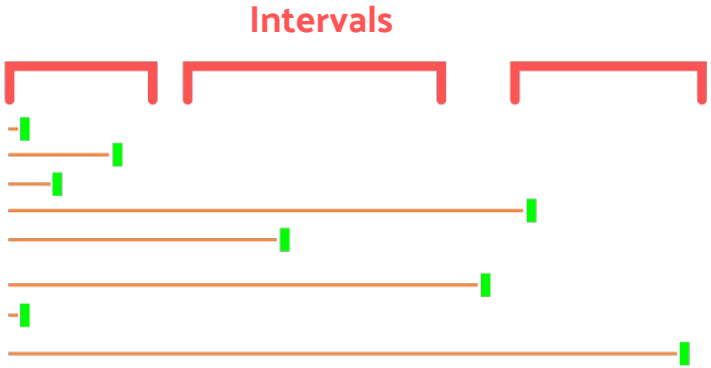
For the first subtask, a brute force search of each word will pass.

For the second subtask, we make the observation that words can be compared lexicographically, so binary search can be used for lookup in a sorted array of words. Applying that logic, the dictionary provided can be binary searched for each word in the sentence. If the binary search does not return a result add it to the output. Another solution is to store the dictionary in a hash table to get constant time lookup.

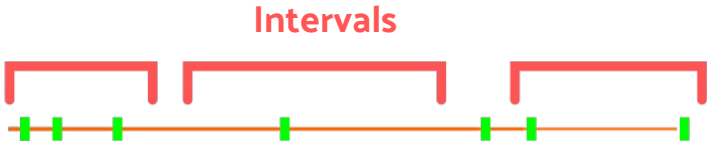
LCC '20 Contest 1 S3 - Subnetting (Hard)

For the first subtask, it suffices to check each query with every interval. The IPs can be compared with an implementation of the rules given in the problem statement. Alternatively, the IPs can be converted to integers.

For the full solution, we observe that by sorting our queries we prevent this recomputation. Alternatively, merge and sort the intervals instead.



Queries



LCC ‘20 Contest 1 S4 - Clowns and Shoes

We will approach the problem greedily.

Looping over the clowns in decreasing order of shoe size, we will maintain the set of shoes that have larger size and have not been taken yet (for example using two-pointers), and make greedy decisions.

Let’s describe a shoe as ”taken” if this shoe is matched with a clown.

Let’s describe a shoe as ”actually taken” if the favourite color of the clown who has this shoe is the same as the shoe’s color (aka this pair adds 1 to our answer).

When we arrive at some clown i, we must decide which shoe to match this clown with. There are a few cases:

Case 1: there are no shoes in the set of ”not actually taken” shoes with clown i’s favourite colour.

If this is the case, we cannot increase our answer, and will thus take the shoe with size closest to this clown’s size for the time being.

LCC ‘20 Contest 1 S4 - Clowns and Shoes

Case 2: there is a shoe in the set of ”not actually taken” shoes with clown i’s favourite color.

We consider taking this shoe. There are three subcases.

Subcase 1: this shoe is in the set of ”not taken” shoes.

We can take this shoe trivially.

Subcase 2: there is a shoe in the set of ”not taken” shoes with size $\geq$ this shoe’s size.

We can make an exchange where whichever clown originally had the shoe we want now takes the bigger shoe, and we get this shoe.

Subcase 3: none of the above

We cannot take this shoe and thus must resort to taking the smallest possible shoe as in case 1 above.

All of this can be maintained with a set of pairs and a map of vectors (implementation details left as an exercise).

Some intuition for taking the smallest possible shoe when we have no other option is that shoe is the least restrictive in terms of what we can take in the future. (A proof would take too long and likely bore everyone)

LCC '20 Contest 1 S5 - Alan's Walk

Let $dist(i, j)$ be the product of $\frac{1}{deg_k}$ for all k on the path from i to j. Note that this is very similar to the probability that this path is chosen, except the first node has a $\frac{1}{deg_i+1}$ chance of taking this path rather than $\frac{1}{deg_i}$. Thus, we have

$$ans_i = \sum_{1 \le j \le n} dist(j, i) \cdot \frac{deg_j}{deg_j + 1}$$

Noting that $dist(j, i) = dist(i, j)$, we can rewrite this as

$$ans_i = \sum_{1 \le j \le n} dist(i, j) \cdot \frac{deg_j}{deg_j + 1}$$

Having derived this formula, we can now solve the problem with a two-pass tree dp. Root the tree arbitrarily, and let $down_i$ be the aforementioned sum over all j in i's subtree. Similarly, let up_i be said sum over all j not in i's subtree. We have

$$down_i = \frac{1}{deg_i + 1} + \sum_{j \in \text{children}_i} \frac{1}{deg_i} \cdot down_j$$

Since $dist(i, i)$ is covered by the first term, and $dist(i, j)$ for all j strictly under i are considered in the sum.

LCC '20 Contest 1 S5 - Alan's Walk

Determining up_i is less trivial, but we notice that we can determine the blue values from $down_{p_i}$, and the red values from up_{p_i} (as seen in the image). We must ensure to subtract the contribution of i's subtree from $down_{p_i}$, and to only consider p_i in one of these sums. We get

$$up_i = \frac{1}{deg_i}\left(up_{p_i} + down_{p_i} - \frac{down_i}{deg_{p_i}}\right)$$

Finally, now that we have both of $down_i$ and up_i, ans_i is merely $down_i + up_i$.