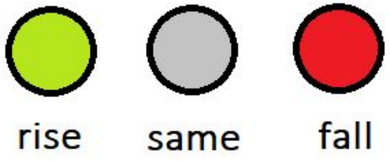
LCC Takeup

By Max, Peter, Kenneth, and ChrisT

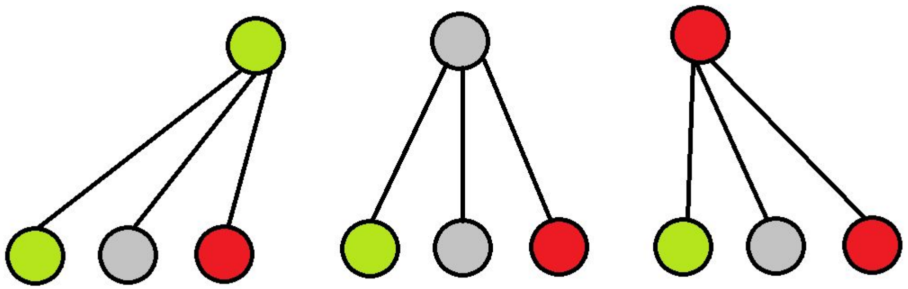
LCC ‘20 Contest 2 J1 - Predicting Stonks

Observe that in order for Ronald to guaranteed a purchase, he must have made one prediction for every combination of how the price will change.

If he is predicting for 1 day, there are 3 possible combinations.



If he is predicting for 2 days, there are 9 possible combinations.



Each day, the number of combination is three times the previous day. The solution is to raise 3 to the k-th power.

LCC ‘20 Contest 2 J2 - Mocking Case

For this problem store the case of the last letter in a variable. The following letter (excluding punctuation characters) should have the opposite case, and remember to update the variable. This can be accomplished using string methods.

Time complexity: $O(|S|)$

LCC ‘20 Contest 2 J3 - Alan’s Snowmen

To solve this problem, store the height of tallest snowman so far, h. The minimum number of snowballs needed to add to the current snowman with height a_i is $h - a_i$.

Because 32-bit integers are not sufficient for this problem, it is required to use your language’s 64-bit integer. This would be **long** in Java or **long long** in C/C++.

Time complexity: O(N)

LCC ‘20 Contest 2 J4 - IPv5

Store each IP address as an array of integers.

It can help to make a function that compares 2 IP addresses x and y. For each block from 0 to N-1, check whether x_i is less than, equal, or greater than y_i. If it is less than, then $x < y$. If it is greater than, then $x > y$. If they are equal, check the following block.

Then simply count the number of IP addresses that are in any of the ranges. It is possible to determine if an IP address is in any of the ranges by looping through all the ranges.

Time complexity: $O(NM)$

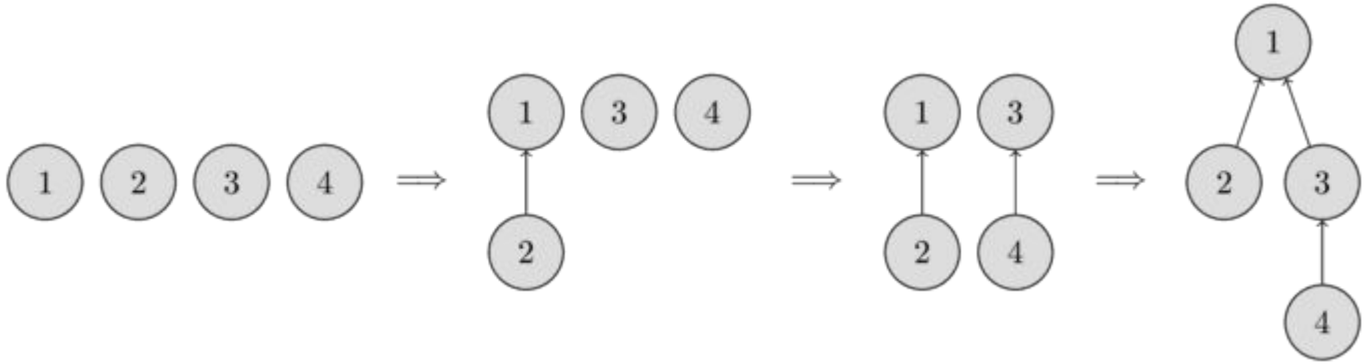
LCC ‘20 Contest 2 J5 - AQT’s Tree

First Subtask: The number of nodes in the tree is relatively small so we can just simulate the deletion of the edges. We will use an adjacency list/matrix to store our graph. To delete an edge just use a built in O(N) (O(1) if using matrix) deletion function, i.e. erase in C++. After every deletion we can just run an O(N) graph search algorithm, bfs/dfs to get the updated min/max of every component. Final complexity is $O(N^3)$ with adj matrix, $O(N^2)$ with adj list.

Full Solution: We need a quicker way of deleting edges and querying (sublinear time for each). Deleting edges is quite hard so we solve the problem in reverse. Instead of starting with one fully connected component, we start with each individual node as its own component. If we go through the list of deletions in reverse, each edge deletion will now become an edge addition.

LCC '20 Contest 2 J5 - AQT's Tree Cont.

How do we efficiently connect two components? Since we don't really need to know the exact structure of the graph (we only need the max, min, and nodes in each component), we can use a **disjoint set (union find)** data structure. We can merge two components quite easily as the max of the resulting component is just the max of the maxes of the two components. By storing a global range variable, we can find the maximum range across all of the components after every edge addition. Final time complexity: O(N log N) or less.



LCC ‘20 Contest 2 S1 - Momentum

An object will “keep going” if it has a strictly higher momentum than the object it collides with. That means for every object we must count the number of elements that are strictly less than it.

This can be accomplished by sorting the N elements a_i, and maintaining the number of elements less than a_i.

Time complexity: O(NlogN)

LCC ‘20 Contest 2 S2 - Squares? Nah.

A point is filled in correctly if it is on the edge of the circle. We can use our equation for the circle, $\mathrm{sqrt}(x^2+y^2) = r$ to answer each query. Make sure to round your answer as the circle is plotted on an integer coordinate plane.

This solution works due to the constraints of the problem R, ($1 \le R \le 1000$), otherwise there may be issues with precision.

LCC ‘20 Contest 2 S3 - Alan’s Got Cake 😳

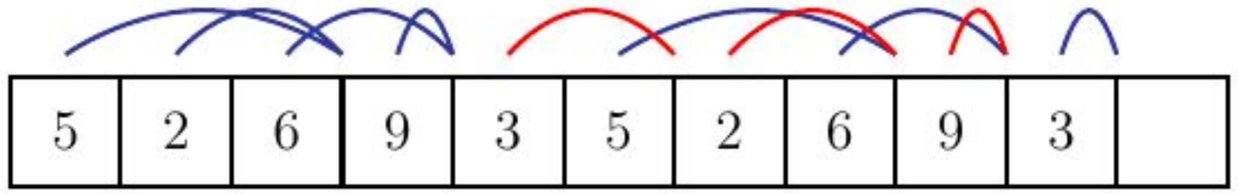
First Subtask: Binary search for the answer. We greedily try to find a partitioning that satisfies the current condition in the bsearch. For example, let’s say we are currently trying to see if an array can be partitioned into subarrays where the sum in each subarray >= x. For a linear array, we can maintain a running sum. When the sum >= x, a new subarray can be started and the sum will be reset. This array will work if the number of partitions made using this greedy algo is >= K.

Since the array is circular, every rotation of the array will be checked in the worst case. It might help to double the array. Since there are N possible rotations and checking if a rotation is possible is O(N). This gives a final complexity of $O(N^2 \log A_{max})$

LCC ‘20 Contest 2 S3 - Alan’s Got Cake 😳 Cont.

Full Solution: We use an array next[] to store the index after the first position in which the subarray sum exceeds x. We realize that if we continuously jump from an index, and end up at the position that is <= to the original index, this is the optimal partitioning starting at that index. The maximal number of partitions made is the number of jumps taken. Instead of searching naively for how many jumps can be made starting at an index, we can precompute every 2^i jump from an index using a **sparse table.** Precomputing takes O(N log N). Searching from an index takes O(log N).

Final Complexity: $O(N \log N \log A_{max})$



LCC '20 Contest 2 S4 - Gamer Alan

Let $f(m,k)$ be the number of ways for Alan to rank up for the first time after playing exactly m games given it takes him k games to rank up. Our answer is

$$n + \sum_{m<n} f(m,k) p^{\text{\# of wins}} q^{\text{\# of losses}} (m-n)$$

Some simple algebraic manipulation gives us # of wins $= \frac{m+k}{2}$ and # of losses $= \frac{m-k}{2}$. Thus, to find the answer we just need to be able to evaluate $f(m,k)$ quickly.

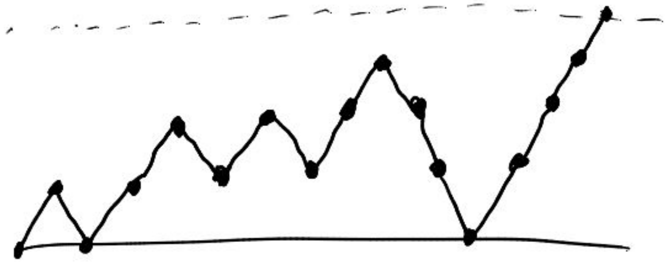
Let's interpret Alan's ELO over time as an integer lattice path. For $f(m,k)$ we are interested in the number of paths that intersect $y = k$ for the first time at $x = m$. This is equivalent to the number of paths of length $m-1$ that end at $y = k-1$ and do not intersect $y = k$ (since we just add a win to the end).

The total number of paths of length $m-1$ that end at $y = k-1$ is $\binom{m-1}{\text{\# of losses}}$, so $f(m,k) = \binom{m-1}{\text{\# of losses}} - g(m,k)$ where $g(m,k)$ is the number of paths of length $m-1$ that end at $y = k-1$ but intersect $y = k$ at some point before $m-1$.

Consider the first time some given path intersects $y = k$. Let's invert this prefix of the path (win $\to$ loss, loss $\to$ win). Note the number of losses we now have is (# of losses $+ k$).

Now consider any path with that number of losses, we can find the first time it intersects $y = -k$, and invert that prefix of the path. Note that the new path we get is a path that ends at $y = k-1$ but intersects $y = k$ at some point.

Thus, there is a bijection between these "bad" paths and the paths with number of losses # of losses$+$ k. Therefore $f(n,k) = \binom{m-1}{\text{\# of losses}} - \binom{m-1}{\text{\# of losses}+k}$ and we are done.



LCC ‘20 Contest 2 S5 - AlanLiChess2004

$$f(x,y) = xy^2 + xy^{-2} + x^{-1}y^2 + x^{-1}y^{-2} + x^2y + x^2y^{-1} + x^{-2}y + x^{-2}y^{-1}$$

want: $[x^a y^b] f(x,y)^n$

$$g(x,y) = x^2y^2 f(x,y) = x^3y^4 + x^3 + xy^4 + x + x^4y^3 + x^4y + y^3 + y$$
$$= x(x^2+1)(y^4+1) + y(y^2+1)(x^4+1)$$

want: $[x^{2n+a} y^{2n+b}] g(x,y)^n$

$$g(x,y)^n = \sum_{i=0}^{n} \binom{n}{i} \left(x(x^2+1)(y^4+1)\right)^i \left(y(y^2+1)(x^4+1)\right)^{n-i}$$
$$= \sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i} \left(\sum_{j=0}^{i} \binom{i}{j} x^{2j}\right) \left(\sum_{j=0}^{i} \binom{i}{j} y^{4j}\right) \left(\sum_{j=0}^{n-i} \binom{n-i}{j} y^{2j}\right) \left(\sum_{j=0}^{n-i} \binom{n-i}{j} x^{4j}\right)$$

$$[x^{2n+a} y^{2n+b}] g(x,y)^n = \sum_{\substack{i+2j_0+4j_3=2n+a \\ n-i+4j_1+2j_2=2n+b \\ 0\le i\le n \quad 0\le j_0, j_1\le i \\ 0\le j_2, j_3\le n-i}} \binom{n}{i}\binom{i}{j_0}\binom{i}{j_1}\binom{n-i}{j_2}\binom{n-i}{j_3}$$

LCC '20 Contest 2 S5 - AlanLiChess2004

$$= \sum_{i=0}^{n} \binom{n}{i} \left(\sum_{\substack{j_0 + 2j_3 = \frac{2n+a-i}{2} \\ 2j_1 + j_2 = \frac{2n+b-n+i}{2} \\ 0 \le j_0, j_1 \le i \\ 0 \le j_2, j_3 \le n-i}} \binom{i}{j_0} \binom{i}{j_1} \binom{n-i}{j_2} \binom{n-i}{j_3} \right)$$

$$= \sum_{i=0}^{n} \binom{n}{i} \left(\overset{O(n)\text{ terms}}{\underset{\substack{j_0 + 2j_3 = \frac{2n+a-i}{2} \\ 0 \le j_0 \le i \\ 0 \le j_3 \le n-i}}{\sum}} \binom{i}{j_0} \binom{n-i}{j_3} \right) \left(\overset{O(n)\text{ terms}}{\underset{\substack{2j_1 + j_2 = \frac{2n+b-n+i}{2} \\ 0 \le j_1 \le i \\ 0 \le j_2 \le n-i}}{\sum}} \binom{i}{j_1} \binom{n-i}{j_2} \right)$$

$$= O(n^2)$$