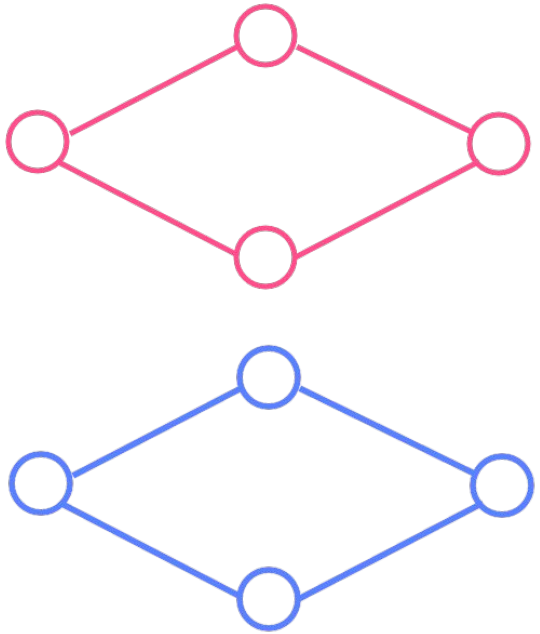
Contest & LCC Takeup

By Max, Peter, Kenneth, and ChrisT

Graph Theory P1: Zipline Bonanza

A stacked graph, similar to the ziplines, will consist of two symmetrical, connected graphs that are stacked on top of each other. The graphs will then be connected vertically to represent the poles which connect corresponding top and bottom layers.

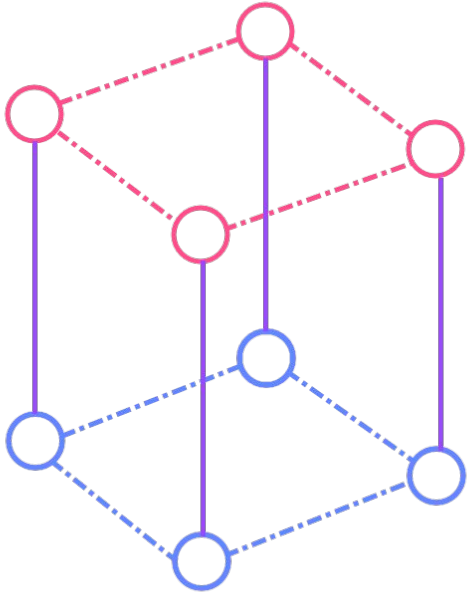


First, consider the number of nodes. One layer of the graph contains half the total nodes, so we know N must be even.

Graph Theory P1: Zipline Bonanza

Now that we know the constraints for N, let's figure out the constraints for M.

We see that there must be N edges to connect the two layers of the graph vertically. Additionally, the top and bottom graph must be symmetrical. Therefore we have (M-N)/2 edges for each layer of the graph.



Graph Theory P1: Zipline Bonanza

The final important condition is that each layer must be a connected graph, but there can at most be one edge between any two nodes.

Let X be N/2, or the number of nodes in each layer. We can then how that the minimum amount of edges to create a connected graph is X-1.

Now let us figure out the maximum number of edges we can fit. Node 1 can have X connections to the other nodes. Node 2 can then have X-1 connections. Thus, the maximum number of edges is X + (X-1) ... + 2 + 1, which can be rewritten as X*(X-1)/2

We end up with two conditions to satisfy for a stacked graph:

N must be even and X - 1 <= (M-N)/2 <= X*(X-1)/2

Graph Theory P2: Towns

This is a modified MST, where the distance between 2 points is min(Δx, Δy).

The naive solution would be to create $O(N^2)$ edges and then compute the MST, for a running time of $O(N^2 \log N)$.

To get the full solution, notice that it is never optimal to connect 2 points if there is 1 between them (between meaning inside the rectangle formed by the 2 points).

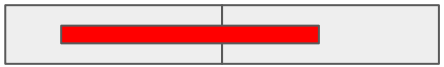
Therefore, only O(N) edges are required.

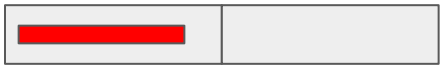
Time complexity: O(NlogN)

Graph Theory P3: Peter's Snazzy Sock Shopping Spree

This problem involves querying the maximum subarray sum on a path.

First consider how to combine 2 segments to get the maximum subarray sum of the final segment. There are 2 cases:



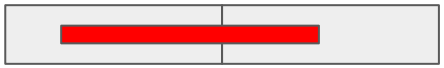


In the 1st case, the new maximum sum subarray is part of both segments. In the 2nd case, it is part of only 1.

Graph Theory P3: Peter's Snazzy Sock Shopping Spree

The 2nd case is easy to deal with (simply store the maximum subarray sum of the previous 2 segments).

To check for the 1st case, notice that we can split the new maximum subarray sum into a left part and a right part. The left part is the maximum sum suffix, and the right part is the maximum sum prefix (the prefix/suffix with the largest sum).



Therefore, to merge 2 segments, maintain the maximum subarray sum, and both the maximum sum prefix & suffix.

Graph Theory P3: Peter's Snazzy Sock Shopping Spree

For the solution, use binary lifting to find the lowest common ancestor. In addition to storing the 2^ith parent in the sparse table, store the previously mentioned quantities of the path from the node to the 2^ith parent (be careful for off-by-one errors and to not accidentally make overlapping paths). Use the same binary lifting technique to combine the individual segments together to get the entire path.

Time complexity: O(NlogN)

Challenge: Solve with updates >:).

LCC ‘20 Contest 5 J1 - Tracy the Chat Filter

Loop over each word and check if it is made completely of uppercase letters. This can be implemented in a variety of ways.

One way would be to convert each character using ASCII values. You could then see if a character’s value is less than 91 making it an uppercase character (this works because the input only contains letters).

In Python, you could use str.isUpper() to check if a word is fully capitalized.

Keep a counter of the total number of words and the number of fully capital words, then output accordingly.

LCC ‘20 Contest 5 J2 - Puzzle Streaking

Implementation problem.

Loop over each of the results (AC or WA) and keep count of Justin’s streak. Based on the conditions change Justin’s point total. Adding points for AC or when he reaches a streak. Removing points for WA.

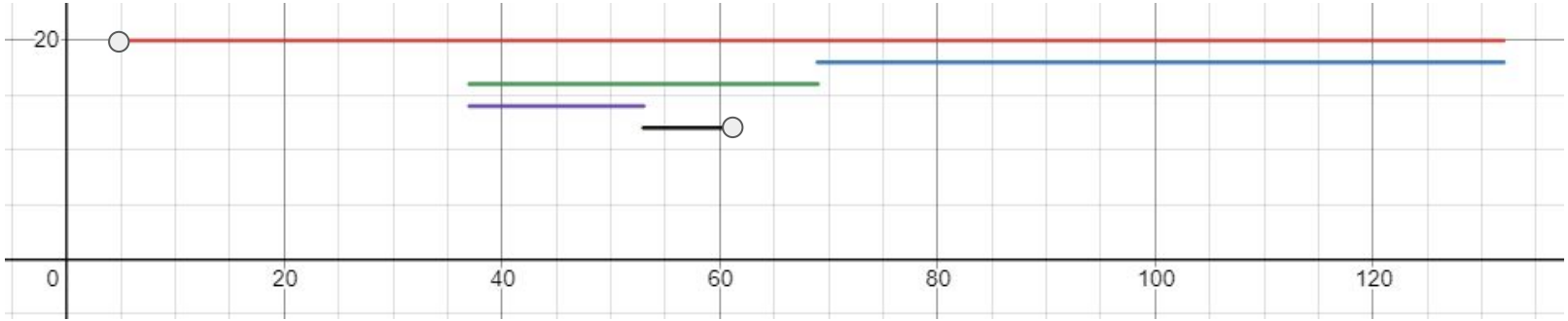
If his points ever reaches negative, end the loop and output 0. Otherwise, print his final point total.

LCC ‘20 Contest 5 J3 - Ping Difference

Let us start by analyzing the problem statement.

The distance the data travels at step i follows the sequence $K^P, K^{P-1}, \ldots, K^1, K^0$.

Where K is given in the input and P is an arbitrary value. Essentially, the data moves in magnitudes of a power. To help us visualize this, let’s imagine a scenario where K = 2 P = 7



The data travels back and forth with exponentially smaller distances.

LCC ‘20 Contest 5 J3 - Ping Difference

Expanding on that, we can show that

$$x^n >= x^{n-1} + x^{n-2} \ldots + x^1 + 1$$

for all bases $x > 1$

Thus, it can be shown that whenever the data jumps in a direction, it must go towards the destination. If goes away from the destination, the consequent jumps will never be enough to go back in the other direction.

Once we find the minimum value P that allows for the data to reach the destination, we can just simulate the jumps, moving closer to the destination each time.

LCC ‘20 Contest 5 J3 - Ping Difference

Proof $\mathbf{x^n >= x^{n-1} + x^{n-2} \ldots + x^1 + 1}$ for all bases $x > 1$

$S = x^{n-1} + x^{n-2} \ldots + x^1 + 1$

$xS = x^n + x^{n-1} \ldots + x^2 + x$

$xS = S - 1 + x^n$

$xS - S + 1 = x^n$

$S(x-1) + 1 = x^n$

As $x > 1$ we can see that $S + 1 <= x^n$

LCC ‘20 Contest 5 J4 - Alan’s Rectangle

Notice that $a_{ij} = 1$. Therefore, the problem reduces to the sum of areas of all possible rectangles with dimensions ≤ K.

For a rectangle with dimensions x and y, its area will be A = xy. There will also be (N-x)(M-y) occurrences of this rectangle. Therefore, the answer is given by:

$$A = \sum_{i=1}^{\min(n,k)} \left(\sum_{j=1}^{\min(m,k)} i\,j\,(n-i)\,(m-j) \right)$$

Because the dimensions of the rectangle are bounded by min(n, k) and min(m, k).

LCC '20 Contest 5 J4 - Alan's Rectangle

Notice that by the linearity of summation, we can split the composition of summations into a product of 2 summations:

$$\left(\sum_{i=1}^{\min(n,k)} i\,(n-i)\right)\sum_{j=1}^{\min(m,k)} j\,(m-j)$$

This solution is O(min(n, k)min(m, k)), which yields 50% of the points. To obtain full points, it is required to calculate both sums in O(1) time or similar. This is left as an exercise to the reader.

LCC '20 Contest 5 J5 - O-Mo-Te-Na-Si

We want to find max(abs(sum of a_i) + abs(sum of b_i)) for some subset of size K

Subtask 1: Loop through all possible combinations of mixtures.

Time complexity: $O(N * 2^N)$

Subtask 2:

If all the values were positive, we would be able to take the K largest values of $(a_i + b_i)$.

To consider negative values, think about trying to minimize one of the traits. For example, if we want to minimize the b's, find the sum of the K largest values of $(a_i - b_i)$.

There are only 4 cases to consider, $(a_i + b_i)$, $(a_i - b_i)$, $(-a_i + b_i)$, $(-a_i - b_i)$. Finding the K largest values takes O(N log N).

LCC ‘20 Contest 5 S1 - Herbs and Ghouls

The most difficult part of this problem is likely processing the input. After extracting your ranges from the input, you can iterate through a boolean array and set the herbs that have been eaten. Note that there is no overlap in the ranges, as given by the problem statement, so iterating through the ranges will take O(N) time.

Afterwards, you can loop through the boolean array again. Keeping track of where a patch of uneaten herbs starts/ends and outputting the ranges.

LCC ‘20 Contest 5 S2 - Facials!

In order for something to be considered a face, it must have 2 eyes and an exactly matching mouth to go with it. Every pair of eyes has 0 pixels in between them, so to find all pair of eyes for each row can be done in O(W) using a nested for loop or 2-pointers.

After finding a pair of eyes, use trigonometry to calculate the exact location and size of the mouth. If there is a group of pixels that exactly matches this mouth, then this is considered a face. If there are erroneous adjacent pixels, then it is not a face.

Do this for every pair of eyes to get the solution.

LCC ‘20 Contest 5 S3 - Larry’s Paradox

Since Larry can only drink up to 6 energy drinks, we can loop over the number of energy drinks he drinks, and determine if it is possible for him to complete all his activities in N (24+e) hour days.

For a fixed day length, we could consider every order in which Larry could conduct his activities, and for each order determine how many days it would take him to do them in that order. If any order takes at most N days, then Larry’s Paradox can be satisfied with this day length, otherwise it cannot.

Unfortunately, this solution is O((L+S)!) which would take about 1e18 operations in the worst case, clearly way too slow.

LCC '20 Contest 5 S3 - Larry's Paradox

To optimize this solution, we will do a dp over subsets. Let dp[mask] = {a,b}, where a is the minimum number of days that Larry must fully spend to complete the activities in mask, and b is the minimum number of hours Larry must have spent so far in the current day.

To transition between states, we consider the last activity that Larry did, making sure to start a new day if necessary.

The final time complexity is $O((L+S)2^{(L+S)})$, which passes comfortably in the time limit.

One may note that when N=2, this problem reduces to a variant of knapsack, leading us to believe that we cannot do much better than this.

LCC '20 Contest 5 S4 - Ninjaclasher's Mobs

We are given N points on the plane, each one of M colours. We want to determine if there is some convex polygon that contains only points of one colour.

If we take a convex polygon around points of colour C, it is clearly optimal to take the convex hull of said points, otherwise we could move our polygon inward and still cover all the points. Thus, the problem reduces to determining if any point of a different colour is contained in the convex hull formed by the points with colour C.

We can determine if a point is in a convex polygon quickly by binary searching over the sides of the polygon, and using cross product to determine which direction the point is relative to the current side.

LCC '20 Contest 5 S5 - Segment Tree Beats

Query: count how many $a_i = x$.
Updates: add x mod (1e5+3) / multiply by x mod (1e5+3). Done on ranges.

The name of the problem is a red herring.

Subtask 1: Brute force. O(N)

Subtask 2: Sqrt decomposition (sqrt(N) blocks of size sqrt(N)). Each block will store the frequencies of each value. Blindly applying the updates on the blocks will not yield a desired time complexity as we need to update a block in sublinear time.

LCC '20 Contest 5 S5 - Segment Tree Beats

Along with the frequencies, also store 2 variables p and q for each block. Instead of updating the frequencies of elements in a block, we will update the values of p and q by representing the value of an element as pa_i+q.

Updates for multiplication:

Blocks: $x*(pa_i+q) = x*pa_i+x*q$. The value of p is now $x*p$ and the value of q is now $x*q$.

Single elements: We want to avoid changing the values of p and q of the block the element is in. This can be done by considering the equation $x*(pa_i+q)=pb_i+q \pmod{1e5+3}$, where a_i is the original value and b_i is the updated value.

There is an edge case for multiplication by zero. To account for this case, think about lazy propagation for segment trees and how it can be applied to sqrt decomp. Updates for addition and queries are left for the reader to implement.

"Similar" problem that is simpler: https://dmoj.ca/problem/ecoo20p4