SCT Blitz 3 Editorial

June 12th, 2020

Card Deck

We can efficiently calculate the answer for every prefix by maintaining a running sum of how many red, blue, and yellow cards there are (call these values r, b, and y). If we pick $3n$ cards, meaning we want n of every color, then the number of cards we need to change is $(|r-n|+|b-n|+|y-n|)/2$. Take the minimum over all prefixes as the answer.

AP vs AD

If an item gives Akali p AP and d AD, then it will increase her average damage per second by $d+p/K$. To avoid working with decimals, which can result in precision errors, we can express this instead as $dK+p$. Sort the weapons by this value, keeping track of original indices, and greedily take the largest M of them.

Baron Nashor

The main idea here is to binary search on the answer. For a given t, it's relatively straightforward to calculate how much damage the champions will have done. Simply check if this damage is enough to slay Baron Nashor and adjust your lower and upper bounds accordingly.

Binary Path

First, let's find a number n_1 such that a path exists from $a \to n_1$, and a number n_2 such that a path exists from $n_2 \to b$. If both these numbers exist, we can always find a path by using a buffer, such as $1110\cdots n_1 \to 1100\cdots \to 1000\cdots n_2$.

A straightforward value for n_1 is $a \wedge 2a$. If this is zero, then no value for n_1 exists. Finding a value for n_2 is a bit trickier. The pattern $\cdots 011\ldots110\cdots$ must be present in b—then, n_2 can be $\cdots 111\ldots111\cdots$. Be sure to handle the edge case where, even though there is no value for n_1 or n_2, a solution still exists because $a=b$.

Forge God

Since the values of a_i are small, we can brute-force every possible strength of a weapon Ornn can upgrade. For every strength, find the maximum subarray sum that contains that strength.

To find the maximum subarray sum, maintain a prefix and suffix array. For the prefix array, let $pref[i]$ be the greatest subarray sum of weapons that ends at index $i-1$ (since the upgraded weapon doesn't gain strength). The transition is fairly simple;

$pref[i] = max(0, pref[i-1] + a[i-1] + s)$, where s is the strength of the weapon Ornn upgrades—for the suffix array, do the reverse. Then, for every index i such that $s_i = s$, update the answer as $pref[i] + suff[i] + s_i$.

City Repair

You can use a single DFS to calculate the maximum imbalance for every city by maintaining the largest value seen from the current node to the root. Sort the imbalances of every city, and greedily repair the city with the most imbalance (decreasing the largest imbalance by 1) K times, or until the maximum imbalance is 0.

This greedy approach works because the imbalance of a city is caused by the maximum-valued ancestor, and the imbalance of that ancestor is 0 (since there are no ancestors with a larger value). Therefore, repairing the most-imbalanced city will never cause the imbalance of another city to increase.