SCT Blitz 5 Editorial

June 29th, 2020

Pearls

There are a couple options here. One is to generate every permutation of the necklace and add them to a set. Another way is to count the frequencies of each color, and hardcode the answer — there are only 5 different frequency combinations.

Yet Another Binary Question

We need to find a couple different values to reach the final answer. Let $f(x)$ be the number of set bits in x. If (and only if) $f(k) > \sum_i f(a_i)$, then there is no answer. Otherwise, let $v_1 = \sum_i f(a_i \wedge k)$ be the total number of set bits every a_i has in common with k. Then, let $v_2 = \sum_i f(a_i) - v_1$, the total number of set bits every a_i does **not** have in common with k. Finally, let v_3 be the number of set bits in k such that no a_i has a set bit in the same position. Our answer is at least v_2, since we must perform an operation for each one of these bits, and at least v_3, since we need to perform an operation of the second type for each one of these bits. Therefore, our final answer is $\max(v_2, v_3)$.

Teacups

The main idea for this problem was to binary search on the answer. Calculate the number of operations it takes to fix all the teacups, if each operation costs a fixed energy e, and compare this number with T and adjust your lower and upper bounds accordingly.

Building a Wall

At first glance, this appears to be DP. Clearly, $dp[h] = dp[h-1] + dp[h-2]$. However, since $dp[0] = dp[1] = 1$, we can see that this is actually the Fibonacci sequence! We just have to find the N-th Fibonnaci number mod M. However, this still doesn't help us that much; N is far too big for an $O(N)$ solution to pass.

Instead, note that if we keep writing out the Fibonnaci sequence mod M, we'll eventually write out $1\ 1$ again (which are the first two values) and the sequence will start all over. The length of the sequence has an upper bound of M^2, so we can find the period p of the sequence and find the $N \bmod p$-th Fibonnaci number.

Social Distancing

First, let's find only the greatest number of friends Justin can invite, and worry about reconstructing the answer later. Arbitrarily root the tree at node 1, and run a DFS. Let $dp[i][0]$ be the greatest number of friends Justin can invite in the subtree rooted at i without inviting friend i, and $dp[i][1]$ be the greatest number of friends while also inviting friend i.

First, if Justin decides to invite friend i, then he cannot invite any friend who lives next to friend i. Therefore, $dp[i][1] = \sum_j dp[j][0]$, where friend j is a child of friend i. If Justin decides not to invite friend i, he can choose, independently of all other choices, to either invite or not invite friend j. Therefore, $dp[i][0] = \sum_j \max(dp[j][0], dp[j][1])$.

To reconstruct our answer, we run a second DFS with a flag f. If $f = 0$, then we cannot invite the current friend; otherwise, we can choose to either invite or not invite the current friend, based on whether $dp[i][0]$ or $dp[i][1]$ is greater. If we choose to invite the current friend, then add it to the answer, set $f = 0$, and repeat on all children. Otherwise, set $f = 1$ and repeat on all children.

Bad Driver

In order to solve this problem, you need to know that, given an adjacency matrix A, then A^N represents all paths of length N.

Initiate an adjacency matrix filled with zero, and for every edge $i \leftrightarrow j$, $A[i][j] = A[j][i] = 1$. Then, use fast exponentiation to find the answer in $O(N^3 \log K)$.