15295 Spring 2020 #14 -- Problem Discussion
April 22, 2020

This is where we collectively describe algorithms for these problems. To see the problem statements follow this link. To see the scoreboard, go to this page and select this contest.

A. Coloring Contention

We wish to alternate the colors as much as possible along any path. Note that if there are two paths and one is shorter than the other, then the longer path will have at least as many switches as the shorter path, when colored optimally. When the paths are disjoint, there are more switches in the longer path. When it is not, remove the overlapping parts to reduce to the previous case. When we add it back, at the vertices where they meet, the longer path might make one less switch depending on the parity of the lengths, but this will still give us our desired result. Hence we only need to consider the shortest path between 1 and N, which is easily found with BFS. Then the number of switches is just the length of the shortest path - 1.

B. Maze Connect

C. Finite Fractions

A fraction a/b having a finite number of decimal places in base B is equivalent to there existing a n >= 0 such that a/b * B^n is an integer. If a/b is in reduced form we see this implies that b | B^n. So B must contain a copy of every prime factor in b, for all the fractions. We see this is sufficient, since we can set n = highest exponent of any prime in b.
From here we just need to simplify all the a/b, and factor their denominators. We can run a prime sieve up to \sqrt(b) and store prime divisors of each number so we can factor b efficiently.
-Zack Lee

D. Error Correction
Note that we can represent the problem as a graph. For each string, create a node in the graph. Then, connect the nodes if they are off by exactly one swap (like in the problem statement)
Now, note that finding the size of the maximum independent set in the graph is equivalent to creating the largest set of non-swappable strings.

However, finding the max independent set is NP hard, so n <= 500 is way too slow.
Now notice that the graph is actually bipartite, since we can never reach the same in an odd number of swaps. As a result, we can solve the size of the max independent set in polytime using max flow.
-- Ryryme

Let me elaborate on this last point a little bit. Here's how to compute the maximum independent set of a bipartite graph G = (A,B,E) using max flow. Set up a source vertex s with edges of unit

capacity from s to all of A. Set up a sink vertex t with edges of unit capacity from all of B to t. For an edge e = (a,b) of E, add an edge from vertex a of A to b of B with infinite capacity. Let n = |A| + |B|.

Claim: Let C be any cut of finite capacity in this flow graph. Let |C| be the capacity of C. Then there exists an independent set in G of size n-|C|, and we can find it from C.

Proof: Note that C must only separate edges involving s and t (the rest have infinite capacity). Consider the vertices A' of A on the s side of cut C. And consider the vertices B' of B on the t side of the cut. Note that n-|C| = |A'| + |B'|. Also note that there cannot be an edge of G between any vertex of A' and any vertex of B'. If there were, |C| would be infinite. Therefore A' Union B' is an independent set, and it's of size n-|C|. Similarly, this construction is reversible, so any independent set of size m gives a cut of size n-m. So a minimum cut will give a maximum independent set. QED.

---Danny

E. Perfect Flush

We loop through the array from left to right. At each index i, we hope to compute the lexicographically smallest subsequence of the prefix ending at index i that contains the value A[i] and can be extended into a valid subsequence using the remaining elements. At a new index i, if A[i] is not already in our subsequence, then we append it. But before we do so, we should remove the longest suffix of our subsequence that contains elements that are larger than A[i] and exist later on in the array.

You can probably prove that this gives the correct subsequence for prefix i using induction. The answer for prefix i will consist of the answer for some prefix j < i, such that we know A[j] must come before A[i] and all elements between j and i should come after A[i].

-Andy

F. Pivoting Points

I didn't know this at the time, but Andy pointed out that the pivoting process of this problem is from a famous IMO problem from 2011. Here's a nice video to watch about it.

A key invariant of the process is that between the pivots the number of points on the left side of the line always remains the same. So when you're pivoting on some point p, to the left of the current line there are, say, k points to its left. When you leave and come back to p there are still k to its left.

So think about what happens if we just spin the line around the point p (always staying on point p) in clockwise fashion. As you go the number to the left of the line changes. It goes in a sequence like 3 2 3 2 1 2 3 2 3 4. Just changing by 1 each time, and it's cyclic.

Consider what happens to that point p in the original process. Say k=2. For an angle in the sequence when the number is 2, we know that the line must be pivoting on p. And we know that there are four 2s in the above sequence. So we know that the sequence must leave and enter the point p 4 times during a 360 degree traversal.

So to solve the problem we compute for each point the above defined sequence. Compute the histogram of the lines-to-left counts for each point p. Let the "score" of a point be the maximum entry in that histogram. The final answer to the question is the score of the point of maximum score.

I believe it would be possible (but much harder) to simulate the originally defined process to run in O(n^2 log n). I wonder if this is how others who solved this problem did it.

---Danny Sleator