SPb SU Contest: LVII SPb SU Championship

August 27, 2023

B (I flipped the Calendar...)

- We are interested in the number of days that are Mondays or the first days of the month.
- If both events happen, the day is still counted exactly once.
- You can use the unbuilt date/time functions. We were kind enough to make the Year 2038 problem impossible.
- Or you can start counting from the 1st of January; you only need to figure out on which day of the week the 1st of January lands on.

E (Fischer's Chess's Guessing Game)

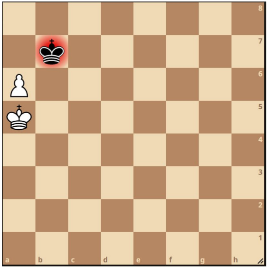
- Use a greedy algorithm. Let the current *state* be the set of positions that still could be the corrent answer.
- Suppose that we guessed a position g.
- Distribute the positions from the current state into 9 bin The position h goes to the bin with a number that would be the Jill's response if the correct answer was h.
- Here the greedy comes: enumerate over all possible g-s and pick g that minimizes the size of the most filled bin.
- An exhaustive search proves that 5 turns isn't enough.

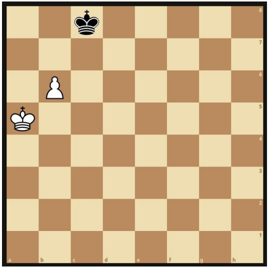
F (Forward-Capturing Pawns)

- A classic game theory problem (the threefold repetition rule doesn't change the outcome and only makes all games finite).
- Retrograde analysis. Remember that stalemates are draws (not losses).
- Implement all chess rules (both old and new) *very* carefully.
- If your implementation does not fit into the time limit, consider the following facts.
- The positions can be easily encoded as a single number.
- A symmetry with respect to a vertical axis doesn't change the outcome.
- You can precompute the answers.

F (Forward-Capturing Pawns), continuation

- You can also try to create a full list of *rules* that determine the outcome of the game.
- It is *very* difficult. For example, it is *not* true that white win in each situation where the pawn is defended and there are no immediate stalemate troubles.





I (Password)

- The simplest problem of the contest. Solved by almost all teams.
- We are allowed to make a password fully consisting from non-letters. Therefore, the maximum number of non-letters is n, where n is the length of the password.
- If we ignore the center requirement, we need at least $\lfloor n/3 \rfloor$ non-letters.
- To deal with the center requirement, fill the center with non-letters; the remaining string splits into two parts of equal length. By itself, each part is equivalent to the case with no center requirement.

K (Poor Students)

- A standard min-cost maxflow problem, but the constraints are too big.
- Indeed, build a flow network with two parts: the left one with the students and the right one with exams. Create an edge from i-th student to j-th exam with capacity 1 and cost $c_{i,j}$. Create $\mathrm{cap} = 1, \mathrm{cost} = 0$ edges from the source to all students. Create a $\mathrm{cap} = a_j, \mathrm{cost} = 0$ edges from the j-th exam to the sink.
- The standard min-cost maxflow algorithm repeatedly finds the shortest path in the residue network. How does it look like? It looks like $s \to \ell_0 \to r_1 \to \ell_1 \to r_2 \to \ldots \to \ell_u \to r_u \to t$. Here, ℓ_i are the students and r_i are the exams.
- Basically, we take a student ℓ_0 and make them pass the exam r_1. To compensate, we take a student ℓ_1, who passed r_1 but not r_2 and transfer them from r_1 to r_2, take ℓ_2 and transfer them from r_2 to r_3, e.t.c.

K (Poor Students), continuation

- Then, for each pair of the courses j and k, we only need to know the best student we can transfer from j to k. The quality of the student i is defined by $c_{i,k} - c_{i,j}$: the less, the better.
- Now, instead of searching for the shortest path in the whole graph, we can construct a smaller graph on the courses and search for a short path within the graph (of course, we still need to take the student ℓ_0 into account; to do so, keep a best new student for each course).
- We have n iterations, each taking $O(k^3)$ time (Ford-Bellman in the course graph).
- To construct the graph quickly, keep $O(k^2)$ sets (described below).
- For each course, order the potential new students by their cost: k sets.
- For each pair of courses, order the potential transfers by their cost (defined above): k^2 sets.
- Each time, at most k students actually transfer. Therefore, we need to do $O(k^2)$ set updates. Total time for set updates: $O(nk^2 \log n)$.
- $O(nk^3 + nk^2 \log n)$ time in total.

M (Hardcore String Counting)

- Denote: g_n — the number of good strings of length n, h_n — the number of strings of length n that don't contain w, $A := 26$ — the alphabet size, $m := |w|$.
- All good strings are obtained in the following way: take a string that doesn't contain w and append w. But not all such strings are good.
- The problem: the last m letters *are not* the first appearance of w.
- Iterate over the first appearance.
- A *border* p of a string s — a string that is simultaneously a prefix and a suffix of s.
- The result: $g_n = h_{n-m} - \sum\limits_{\text{all borders } p \text{ of } w} g_{n-m+|p|}$.

M (Hardcore String Counting), continuation

- Also, $h_n - Ah_{n-1} = g_n$. Therefore,
  $Ag_{n-1} = Ah_{n-m-1} - \sum_{\text{all borders } p \text{ of } w} Ag_{n-m-1+|p|}$
- Subtract. All h-s disappear, because $h_{n-m} - Ah_{n-m-1} = g_{n-m}$.
- The result is a linear recurrence for g-s with $m + O(1)$ terms.
- Compute the n-th term by a standard $O(m \log m \log n)$ algorithm.
- Each time, we take the answer modulo the same polynomial, so we don't need to invert a series more than once. Then, we have $O(\log n)$ iterations, each taking $O(m \log m)$ time, but $O(m \log m)$ comes from a normal polynomial mutliplication and not from a costier inversion.
- Also, there is an even faster way to divide by this particular polynomial. It exploits the fact that border lengths split into $O(\log m)$ arithmetic progressions. However, it wasn't necessary to solve the problem.