AtCoder Grand Contest 044 Editorial

dario2994

May 23, 2020

A: Pay to Win

We consider the problem *backward*. Hence, we start with the number $x = N$ and we have to reach $x = 0$. We can perform the following moves:

2. If $2 \mid x$, replace x with $\frac{x}{2}$; paying A.
3. If $3 \mid x$, replace x with $\frac{x}{3}$; paying B.
5. If $5 \mid x$, replace x with $\frac{x}{3}$; paying C.

$\pm$. Replace x with $x-1$ or $x+1$; paying D.

What is the structure of an optimal strategy? It might be that we apply only move 4., in that case the cost is ND. Otherwise, the strategy begins with

$$N \xmapsto{\pm.} \cdots \xmapsto{\pm.} ky \xmapsto{k.} y\,,$$

where $k \in \{2,3,5\}$. We claim that $y \in \{\lfloor \frac{N}{k} \rfloor, \lceil \frac{N}{k} \rceil\}$. In fact, if $y < \lfloor \frac{N}{k} \rfloor$ (and the same reasoning works if $y > \lceil \frac{N}{k} \rceil$), the following sequence of moves is strictly cheaper than the above one:

$$N \xmapsto{\pm.} \cdots \xmapsto{\pm.} k\left\lfloor \frac{N}{k} \right\rfloor \xmapsto{k.} \left\lfloor \frac{N}{k} \right\rfloor \xmapsto{\pm.} \cdots \xmapsto{\pm.} y\,.$$

Hence, we have shown that an optimal strategy either goes directly to 0 using only the move $\pm$. or goes to one of $\{\lfloor \frac{N}{k} \rfloor, \lceil \frac{N}{k} \rceil\}$ for $k \in \{2,3,5\}$. In particular, if $f(n)$ is the minimum cost of reaching 0 starting from n, we have shown that $f(n)$ can be easily computed if we know $f(\lfloor \frac{n}{k} \rfloor)$ and $f(\lceil \frac{n}{k} \rceil)$ for $k \in \{2,3,5\}$.

The above observation strongly suggests a dynamic programming approach. But it remains to understand how many states we are going to reach starting from N. It turns out that the reachable numbers are those of the two forms[1]

$$\left\lfloor \frac{N}{2^a 3^b 5^c} \right\rfloor \quad \text{and} \quad \left\lceil \frac{N}{2^a 3^b 5^c} \right\rceil .$$

Since $0 \le a \le 60$, $0 \le b \le 40$ and $0 \le c \le 30$ (otherwise the denominator is larger than the numerator), the reachable numbers are less[2] than $2 \cdot 61 \cdot 41 \cdot 31 = 155.062$ (to get accepted, a map should be used to keep the values $f(n)$ for the reachable numbers n).

Implementation: https://atcoder.jp/contests/AGC044/submissions/13531498.

[1]Can be shown applying the two identities

$$\left\lfloor \frac{\lfloor \frac{n}{u} \rfloor}{v} \right\rfloor = \left\lfloor \frac{n}{uv} \right\rfloor \quad \text{and} \quad \left\lceil \frac{\lceil \frac{n}{u} \rceil}{v} \right\rceil = \left\lceil \frac{n}{uv} \right\rceil .$$

[2]With a more careful estimate, one can see that the reachable numbers are ≤ 20.000.

B: Joker

For each $1 \le i \le N^2$, we compute the minimum number of viewers that will hate viewer i forever (the answer is the sum of these values). This number coincides with the minimum cost of a path from the seat of viewer i to the sides of the square, considering that going through an empty seat has cost 0 and going through an occupied seat has cost 1. Let $H_k(i)$ be the minimum cost (as defined above) of a path from the seat of viewer i to the outside after the first k viewers have left the cinema.

Key observation. The values $H_k(i)$ are decreasing (for k going from 0 to N^2) and at the beginning we have

$$H_0(1) + H_0(2) + \cdots + H_0(N^2) \approx \frac{N^3}{6}\,.$$

Our strategy is to keep all values $H_k(i)$ updated at all times. Initializing $H_0(1), \dots, H_0(N^2)$ in $O(N^2)$ is straightforward. Let us show how to update $H_{k-1}(1), \dots, H_{k-1}(N^2)$ to get $H_k(1), \dots, H_k(N^2)$. When the viewer P_k goes away, we perform a breadth-first search (or a depth-first search) starting from the seats of P_k and updating the values. During the k-th breadth-first search, we will visit only the seats i such that $H_k(i) < H_{k-1}(i)$, hence the total number of seats visited in the N^2 steps (for $1 \le k \le N^2$) is $O(N^3)$ (see the *key observation*).

The time complexity of this solution is $O(N^3)$ with a small constant which is sufficient to get accepted (some optimization might be required in slow languages such as `python`).

Implementation: `https://atcoder.jp/contests/AGC044/submissions/13531558`.

C: Strange Dance

Let us denote with i the person starting in position i. For each $n = 0, 1, \dots, N$ we are going to compute the final permutation $P^n_0, \dots, P^n_{3^n-1}$ (after all the songs have been played) if there are 3^n people and the sequence $W^n_0, \dots, W^n_{|T|-1}$, where W^n_k denotes the person in the last position (that is $3^n - 1$) after k songs have been played.

It is straightforward to compute P^0 and W^0. We show how to compute P^{n+1} and W^{n+1} from P^n and W^n.

Key observation. For any $0 \le i < 3^n$ and any $0 \le k < |T|$, it holds

$$P^{n+1}_i \equiv P^{n+1}_{i+3^n} \equiv P^{n+1}_{i+2\cdot 3^n} \equiv P^n_i \pmod{3^n}$$

and

$$W^{n+1}_k \equiv W^n_k \pmod{3^n}.$$

Hence, we have to determine only the most significant digit in base 3 of P^{n+1}_i, as the other digits are the same of P^n_i (the same holds for W^{n+1}_k). Let $D_0, \dots, D_{3^{n+1}-1}$ be the most significant digit of the positions of the people. It is straightforward to initialize these values before any song is played. How do these values change when a song is played?

- If the $(k+1)$-th song is a Salsa, all values 2 become 1 and all values 1 become 2.
- If the $(k+1)$-th song is a Rumba, all the values stay the same apart from the entries of D with indexes W^n_k, $W^n_k + 3^n$, $W^n_k + 2\cdot 3^n$. The corresponding 3 values of D change according to the transformation:

$$0 \mapsto 1,\ 1 \mapsto 2,\ 2 \mapsto 0\,.$$

If we keep D updated at all times, it is easy to see that we are able to compute W^{n+1}_k for all values of k and P^{n+1}_i for all i (thanks to the *key observation*).

Sadly, we cannot naïvely keep D updated, because when a Salsa is played a lot of values change. This issue can be solved handling Salsas lazily: if the $(k+1)$-th song is a Salsa, we update only the only the entries of D with indexes W^n_k, $W^n_k + 3^n$, $W^n_k + 2\cdot 3^n$ (which can be done efficiently keeping track of the last time we have updated a certain entry).

The time complexity of this solution is $O(3^N + N \cdot |T|)$.

Implementation: https://atcoder.jp/contests/AGC044/submissions/13531969.

D: Guess the Password

We describe two possible solutions.

Let $A = 62$ be the size of the alphabet. Both solutions begin asking A queries, one for each character in the alphabet. The query corresponding to character c is a string containing only c with length L. It is easy to see that after these A queries, we know the frequencies of all the characters (and, of course, we know also the length of the hidden password $|S|$).

Solution 1: Bisecting the alphabet. Given a subset E of the alphabet, let $f(A)$ be the maximal subsequence of the hidden password S that contains only characters in E. We already know $f(\{c\})$ for any single character c in the alphabet.

Key observation. Given two strings S, T with $|T| \leq |S|$, the edit-distance between the two is $|S| - |T|$ if and only if T is a subsequence of S.

Thanks to the *key observation*, if we know $f(A)$ and $f(B)$ for two disjoint subsets of the alphabet, we can merge them and obtain $f(A \cup B)$ with $|f(A)|+|f(B)|-1$ queries. Hence we can find $S = f(\{a, b, \dots, z, A, \dots, Z, 0, \dots, 9\})$ with no more than $L\lceil \log_2(A) \rceil - A + 1$ queries. This solution, considering the initial A queries, asks at most $L\lceil \log_2(A) \rceil + 1 = 769$ queries.

Implementation: `https://atcoder.jp/contests/AGC044/submissions/13532026`.

Solution 2: Bisecting the string. Given $0 \leq l \leq r < |S|$ and a character c, let $q(c, l, r)$ be the number of occurrences of c in the substring $S[l \dots r]$.

Key observation. Given two characters a and b, let us define the string T as

$$T := \overbrace{aa\cdots a}^{n}\overbrace{bb\cdots b}^{|S|-n}\,.$$

The edit-distance between the hidden password S and T is $|S| - q(a, 0, n-1) - q(b, n, |S|-1)$.

Fix an interval $[l, r]$ and let $m = \lfloor \frac{l+r}{2} \rfloor$ be the midpoint. Assume that we know $q(c, 0, l-1)$, $q(c, l, r)$, $q(c, r, |S|-1)$ for all characters c and we want to find $q(c, l, m)$ and $q(c, m+1, r)$. Notice that if we have $q(c, l, m)$, the value of $q(c, m+1, r)$ can be obtained as $q(c, l, r) - q(c, l, m)$; hence we focus on computing $q(c, l, m)$.

Thanks to the *key observation*, with one query we can find $q(c, l, m) + q(z, m+1, r)$, where z is a *fixed character*. Hence, with $\min(A, r-l+1) - 1$ queries (we perform the queries only for the characters c present in $S[l \dots r]$ and we do not perform the query for the character z), we obtain $q(c, l, m) + q(z, m+1, r)$ for all characters c appearing at least once in the $S[l \dots r]$. From this information, it easy to deduce the value of $q(z, m+1, r)$ and therefore to obtain the sought values $q(c, l, m)$.

Since we learnt how to compute the frequencies in the two halves of a substring (given the frequencies in the whole substring), and at the beginning we know the frequencies of all the characters in S, we can repeat this argument bisecting the string obtaining a solution that performs $61+2\cdot 61+4\cdot 31+8\cdot 15+16\cdot 7+32\cdot 3+64\cdot 1 = 699$ queries. This solution, considering the initial A queries, asks at most 761 queries.

Implementation: `https://atcoder.jp/contests/AGC044/submissions/13463830`.

E: Random Pawn

First of all, without loss of generality (up to rotating and duplicating a position) we can assume that $A_0 = A_{N-1} = \max A$. This assumption is useful as it implies that whenever we arrive at $p = 0$ or $p = N-1$ it is convenient to end the game (hence we have simplified the game from a circle to a segment!).

Let E_p be the expected gain of an optimal strategy if the pawn is initially at position p (if we are able to compute the array E, then the answer to the problem is its average).

The values E_p are characterized by $E_0 = A_0$, $E_{N-1} = A_{N-1}$ and

$$E_p = \max\left(A_p, \frac{E_{p-1}+E_{p+1}}{2} - B_p\right) \quad \text{for any } 1 \le p < N-1.$$

Given another array $C_0, C_1, \dots, C_{N-1}$ (that we are going to fix later), let us rewrite this identity as

$$E_p - C_p = \max\left(A_p - C_p, \frac{E_{p-1} - C_{p-1} + E_{p+1} - C_{p+1}}{2} + \frac{C_{p-1}+C_{p+1}}{2} - C_p - B_p\right).$$

If we choose C in such a way that $\frac{C_{p-1}+C_{p+1}}{2} - C_p - B_p = 0$ for any $1 \le p < N-1$ (it is easy to construct such a C with integer entries), then the above identity simplifies to

$$F_p = \max\left(A_p - C_p, \frac{F_{p-1}+F_{p+1}}{2}\right),$$

where $F_p := E_p - C_p$. This condition is exactly equivalent to saying that (p, F_p) belongs to the boundary of the upper convex-hull of the points $(p, A_p - C_p)$.

Since the upper convex-hull can be computed in linear time, the time complexity is $O(N)$.

How to find the solution. Let us briefly describe how one can obtain the described solution. Since the game is *memory-less*, the optimal strategy is characterized by the subset of positions where we choose to end the game. Let $l < r$ be two consecutive positions where we want to end the game. We have $E_l = A_l$, $E_r = A_r$ and, for all positions $l < i < r$, it holds

$$E_i = -B_i + \frac{E_{i-1}+E_{i+1}}{2} \iff E_i - E_{i-1} + 2B_i = E_{i+1} - E_i.$$

In order to make this formula homogeneous, it is natural to consider the array C such that $C_i - C_{i-1} + 2B_i = C_{i+1} - C_i$. Doing so, we quickly deduce (with the same definition of F given above)

$$F_i = \frac{(r-i)F_l + (i-l)F_r}{r-l}$$

that shows clearly that the solution is given by a suitable convex-hull.

The continuous version. What follows requires a bit of not-so-elementary math; we give only a sketch.

The problem has a natural continuous version: the pawn moves according to a Brownian motion X_t (on $\mathbb{R}^d$ or on a closed manifold, as in this problem) and if we decide to end the game at time T the payoff is

$$A(X_T) - \int_0^T B(X_t)\,dt.$$

Let $E(x)$ be the expected payoff of an optimal strategy if we start with $X_0 = x$. One can prove that the function $E(x)$ is characterized by the two properties:

- Either $E(x) = A(x)$ and $\Delta E(x) \le 2B(x)$,
- Or $E(x) > A(x)$ and $\Delta E(x) = 2B(x)$.

If C is the function such that $\Delta C = 2B$ and $F := E - C$, the two properties become

- Either $F(x) = A(x) - C(x)$ and $\Delta F(x) \le 0$,
- Or $F(x) > A(x) - C(x)$ and $\Delta F(x) = 0$.

This is the classical formulation of the *obstacle problem* (the function $A - C$ being the obstacle) and its solution is the smallest super-harmonic function above $A - C$. In 1-d being super-harmonic is equivalent to being concave and thus this problem admits an efficient solution. Note that the problem, even the discrete version, would have been much harder if instead of moving on a segment the pawn was moving on a square.

Implementation: https://atcoder.jp/contests/AGC044/submissions/13532052.

F: Name-Preserving Clubs

Let us first consider the problem under the assumption that *two clubs cannot have the exact same members.* In the end, we will explain how to handle equal clubs.

Let us define a (k, n)-board as a matrix with k rows and n columns such that each cell is either 0 or 1. A (k, n)-board is good if the rows are distinct and any permutation (different from the identity) of the columns generates a (k, n)-board that cannot be obtained from the original one permuting the rows.

It is easy to see that good (k, n)-boards are in bijection with name-preserving configurations of clubs (rows represent clubs, columns represent people). Thus, we want to count the number of good (k, n)-boards with k minimum.

Given a good (k, n)-board T (notice that a good board has distinct columns):

- let T^t be its transpose, i.e., the (n, k)-board with $T^t[j][i] = T[i][j]$;
- let T^c be its complement, i.e., the $(k, 2^k - n)$-board that has all the columns not present in T (in an arbitrary order).

Key observation. If T is a good board, then also T^t and T^c are good boards. Moreover, the number of good (k, n)-boards (up to permutation of rows/columns) is the same as the number of good (n, k)-boards and the number of good $(k, 2^k - n)$-boards.

Let $g(n)$ be the minimum k such that there is a good (k, n)-board. Applying the *key observation* we get immediately $2^{g(n)} - n \geq g(g(n))$ (because we obtain the existence of a good $(2^{g(n)} - n, g(n))$-board). It turns out that this inequality is sharp.

Let $G(n)$ be the minimum function such that $G(1) = 0$ and for any $n \geq 2$ it holds $2^{G(n)} - n \geq G(G(n))$ (notice that this function coincides with $\lceil \log_2(n) \rceil$ for almost all n, but not for all n).

Lemma 1. *There exists a valid (k, n)-board if and only if $G(n) \leq k \leq 2^n - G(n)$ (in particular $g(n) = G(n)$).*

Proof. Let us prove the statement by induction on n.

If $G(n) \leq k < n$, applying the *key observation*, it is sufficient to construct a valid (n, k)-board and such a board exists by inductive hypothesis.

Moreover, again thanks to the *key observation*, it is sufficient to prove the existence of a valid (k, n)-board for $k \leq 2^{n-1}$ (indeed, if $k > 2^{n-1}$ we can consider the equivalent problem of finding a valid $(2^n - k, n)$-board). In the range $n \leq k \leq 2^{n-1}$, we construct explicitly a good (k, n)-board. Let T be the good $(n-1, n)$-board induced by the name-preserving configuration of clubs $\{\{1\}, \{1, 2\}, \{2, 3\}, \dots, \{n-2, n-1\}\}$. We fill the remaining $k - n + 1$ rows with at least 3 *ones* per row (this way, the first $n - 1$ rows do not get confused with the others). It is always possible to do so if $k \leq 2^{n-1}$. □

Now, let us focus on *counting* the number of good (k, n)-boards. Let $c(k, n)$ be the number of good (k, n)-boards (the problem asks for $c(G(n), n))$). The *key observation* provides us with the two identities $c(k, n) = c(n, k)$ and $c(k, n) = c(k, 2^k - n)$. Applying these identities, we reduce to the case $k \leq n \leq 2^{k-1}$.

Lemma 2. *If $6 \leq k \leq n \leq 2^{k-1}$, then $c(k, n) > 1000$.*

Proof. Consider the two name-preserving configurations of clubs (if $A \subseteq \{1, \dots, k\}$, we denote $A^{\mathsf{c}} = \{1, \dots, k\} \setminus A$)

$$\{\{1, 2\}, \{2, 3\}, \dots, \{k-4, k-3\}, \{k-3, k-2\}, \{k-2, k-1\}^{\mathsf{c}}\}\,;$$
$$\{\{1, 2\}^{\mathsf{c}}, \{2, 3\}^{\mathsf{c}}, \dots, \{k-4, k-3\}^{\mathsf{c}}, \{k-3, k-2\}^{\mathsf{c}}, \{k-2, k-1\}\}\,.$$

They have $k - 2$ clubs each, and all of their clubs have either 2 or $k - 2$ members. Hence, if we add, to any of the two configurations, $n - k + 2$ clubs with a number of members different from 2 or $k - 2$, we still have a name-preserving family (and all the families generated in this way are different). Hence we deduce

$$c(k, n) \geq 2\binom{2^k - \binom{k}{2} - \binom{k}{k-2}}{n - k + 2}\,.$$

It is easy to see that, under the constraint $6 \leq k \leq n \leq 2^{k-1}$, this quantity is minimized at $k = n = 6$ and therefore

$$c(k, n) \geq 2\binom{2^6 - 15 - 15}{2} = 34 \cdot 33 > 1000\,.$$

□

Given Lemma 2, it remains only to compute $c(k,n)$ for $k \leq n \leq 2^{k-1}$ and $k \leq 5$. We can iterate over all (k,n)-boards checking (in the naïve way) whether the board is good (stopping as soon as $1000+1$ good boards are found). Note that there are 22 possible (k,n) pairs and only 11 of them are such that there are ≤ 1000 good boards.

Equal clubs. Let us explain how to handle the fact that the clubs may be non-distinct. If there is a name-preserving configuration of k clubs with two equal clubs, then[3] $2^{k-1} \geq N$ and therefore $k \geq G(N)$ (so the minimum number of clubs computed above remains correct). If $k = G(N)$, then $2^{G(N)-1} \geq N$ and this inequality holds only for the numbers N of the form $2^M - G(M)+1, 2^M - G(M)+2, \dots, 2^M$ for some natural number M. Considering what we have said above, it is straightforward to see that the number of good $(G(N), N)$-boards is larger than 1000 for all those numbers apart from 4 and 7 and 8. If $N = 4$, there is 1 minimal name-preserving configuration with non-distinct clubs; if $N = 7$ there are 2 minimal name-preserving configurations with non-distinct clubs; if $N = 8$ there are 0 minimal name-preserving configurations with non-distinct clubs (these facts can be obtained either with pen and paper or generating all possible configurations with a computer).

Implementation: `https://atcoder.jp/contests/AGC044/submissions/13532081`.

[3]Indeed, if the number of unique clubs is h (two clubs having the same members are counted only once) then it holds $2^h \geq n$, because two people cannot belong to the exact same clubs (since the configuration is name-preserving).