Problem Tutorial: "Avg"

If k^x is not divisible by n for any $x > 0$ (or, equivalently, if n has a prime divisor that k doesn't have), no sequence of steps exists. To prove that, consider an array $A = (0, 0, \ldots, 0, 0, 1)$: each element of this array must be equal $\frac{1}{n}$ in the end, but after x steps we can only obtain rational values whose denominators are divisors of k^x.

Otherwise, a valid sequence always exists, and we can construct it inductively. If $n = k$, take $b_i = i$. Otherwise, find any $d > 1$ that is a divisor of k and $\frac{n}{k}$ (for example, $d = \gcd(k, \frac{n}{k})$). Split n elements into groups of size d. For each $\frac{k}{d}$ consecutive groups, perform a step equalizing them. Now the elements in each group are equal. Finally, form d groups of size $\frac{n}{d}$, one element from each group, and solve the problem recursively for each group.

The case when n is not divisible by k is more interesting. It's not even obvious how to check if a sequence exists: for example, when $n = 8$ and $k = 6$, it seems that there is no solution. If you have any insights about this, please share!

Problem Tutorial: "Bin"

Let's denote the given function as $f_k(n)$, then $f_k(n) = \sum\limits_{i=1}^{\min(n-1, \lfloor \frac{k+n}{2} \rfloor)} f_k(i) \cdot f_k(n-i)$.

Consider the following problem: we are given n integers $a_0, a_1, \ldots, a_{n-1}$ one by one, and after we get each a_i, we need to respond with $c_i = \sum_{j=0}^{i} a_j a_{i-j}$. If we can solve this problem in $O(T(n))$, we can solve the original problem in $O(T(n) + nk)$. Let's concentrate on this problem then.

If we are given all values of a_i in advance, we can just convolve the sequence with itself using NTT in $O(n \log n)$, however, the main difficulty lies in finding the coefficients of the convolution on the fly.

Let $f(L, R)$ be a recursive function that solves the problem for all $i \in [L; R)$, assuming that we already have coefficients up to $i-1$-th, and also b_i for $i \in [L; R)$ has been calculated as $b_i = \sum_{j=i-L+1}^{L-1} a_j a_{i-j}$.

If $R - L = 1$, read a_L and respond with $c_L = b_L + 2a_0 a_L$ if $L > 0$, and with $c_0 = a_0^2$ otherwise.

If $R - L > 1$, find $M = \lceil \frac{L+R}{2} \rceil$ and call $f(L, M)$ first. Now we want to call $f(M, R)$, but we need to recalculate b before that. If $L > 0$, convolve $a_L, a_{L+1}, \ldots, a_{M-1}$ and $a_0, a_1, \ldots, a_{M-L-1}$ using NTT, then add corresponding coefficients of this convolution to b_i for $i \geq M$. Otherwise, if $L = 0$, just convolve $a_0, a_1, \ldots, a_{M-1}$ with itself and replace b with the result.

Time complexity of this solution is $O(n \log^2 n)$.

Problem Tutorial: "Cat"

We'll consider a solution using suffix array, but solutions using other suffix structures are possible as well.

Let $s = a+b$. We need to find the number of distinct substrings of s with at least one occurrence containing characters at positions $|a|$ and $|a|+1$.

Let $n = |s|$. Let's build the suffix array $p_1, p_2, \ldots, p_n$ of s and let $l_i = LCP(s_{p_i..n}, s_{p_{i+1}..n})$. If we just needed to count distinct substrings of s, that number would be $\binom{n+1}{2} - l_1 - l_2 - \ldots - l_{n-1}$.

Let's consider suffixes in order $p_1, p_2, \ldots, p_n$. For each i, first, some prefixes of suffix p_{i-1} can be marked as they will never appear again. Then, suffix p_i brings substrings $s_{p_i..p_i+l_{i-1}}, s_{p_i..p_i+l_{i-1}+1}, \ldots, s_{p_i..n}$ into play. If $p_i \leq |a|$, for $|b|$ longest prefixes of $s_{p_i..n}$, we also know now that they have an occurrence covering positions $|a|$ and $|a|+1$.

It's enough to maintain some data structure that simulates an array with the following queries:

- set 0 or 1 to all values in some range;
- find the sum of value in some range.

A usual segment tree will do. (It's also possible to use the structure of queries and go with `std::set` or something similar.)

Problem Tutorial: "Div"

Multiply both sides by $(x-1)$. We need to count $x > 1$ such that $P_0(x) = d_0x^{b_0} + d_1x^{b_1} + \ldots + d_{k-1}x^{b_{k-1}}$ is divisible by $x^m - 1$ (check $x = 1$ separately).

Taking $P_0(x)$ modulo x^m-1 as a polynomial, we'll get $P(x) = d_0x^{b_0 \bmod m}+d_1x^{b_1 \bmod m}+\ldots+d_{k-1}x^{b_{k-1} \bmod m}$. If $P(x) \equiv 0$, the answer is infinite. Otherwise, let's count $x > 1$ such that $P(x)$ is divisible by m.

Let's rewrite $P(x)$ as $P(x) = e_0 + e_1x + \ldots + e_{m-1}x^{m-1}$. Note that if $x > \max(|e_0|, |e_1|, \ldots, |e_{m-1}|) + 1$, then $P(x) \neq 0$ and $|e_0| + |e_1x| + \ldots + |e_{m-1}|x^{m-1} < x^m - 1$, therefore, we don't need to consider such x. Now we only need to consider $O(n)$ values of x.

Let's fix x and transform $P(x)$ as follows:

- if there's some i such that $e_i \geq x$, subtract x from e_i and add 1 to $e_{(i+1) \bmod m}$;
- if there's some i such that $e_i \leq -x$, add x to e_i and subtract 1 from $e_{(i+1) \bmod m}$;
- otherwise, stop the process.

This transformation keeps the value of $P(x)$ the same. Each step decreases the sum of $|e_i|$ by at least $x - 1$, therefore we'll do $O(\frac{n}{x})$ steps.

After this process, we have $|e_i| < x$ for all i, and $P(x)$ is divisible by $x^m - 1$ if one of the following applies: all $e_i = 0$, all $e_i = x - 1$, or all $e_i = -x + 1$.

We can perform all steps efficiently using built-in associative containers with $O(\log n)$ overhead and rollback changes done for some x before proceeding to the next x. Since there are $\sum_{x=2}^{n} O(\frac{n}{x}) = O(n \log n)$ steps to be done, the overall time complexity is $O(n \log^2 n)$.

Problem Tutorial: "Exp"

This problem might be well-known in some countries, but how do other countries learn about such problems if nobody poses them?

Consider a polynomial $P(y) = p_0 + p_1y + \ldots + p_ky^k$. If we find the coefficients q_i of $Q(y) = P^n(y)$, the answer is $\sum_{i=0}^{nk} q_i \cdot \min(i, x)$. Since the sum of p_i is 1, we can also rewrite this as $\sum_{i=0}^{x-1} q_i \cdot i + (1 - \sum_{i=0}^{x-1} q_i) \cdot x$. Hence, we just need to find the first x coefficients of $P^n(y)$.

The title of this problem, Exp, stands for expected, experience, and exponentiation.

Consider the derivative of $P^{n+1}(y)$ and find two different expressions for it:

- $(P^{n+1}(y))' = (P(y)P(y)\ldots P(y))' = (n+1)P^n(y)P'(y) = A$;
- $(P^{n+1}(y))' = (P^n(y)P(y))' = (P^n(y))'P(y) + P^n(y)P'(y) = B$.

Since $A = B$, we have $nP^n(y)P'(y) = (P^n(y))'P(y)$. Consider the coefficient of y^i in both parts of this equation:

- in the left part, it's $n(q_ip_1 + 2q_{i-1}p_2 + \ldots + kq_{i-k+1}p_k)$;
- in the right part, it's $(i+1)q_{i+1}p_0 + iq_ip_1 + \ldots + (i-k+1)q_{i-k+1}p_k$.

It turns out that we can derive q_{i+1} from the equality of these two expressions if we know $q_0, q_1, \ldots, q_i$. Each coefficient can be calculated in $O(k)$, hence time complexity is $O(xk)$.

Problem Tutorial: "Flip"

Consider strings of length $2n$ with n letters `A` and n letters `B`, corresponding to team assignments. What is the probability that a string s corresponds to the final team assignment? Let's define l_A be the position of the last occurrence of `A`, and l_B similarly. Then the probability $p(s) = 2^{-min(l_A, l_B)}$.

We need to find the total probability of strings such that $s_{a_1} = s_{a_2} = \ldots = s_{a_k} = \mathtt{A}$.

Let's classify strings on the value of $m = min(l_A, l_B)$ (all such strings have the same probability).

If $m = a_k$, then $s_m = \mathtt{A}$ and the number of such strings is $\binom{m-k}{n-k}$.

If $a_i < m < a_{i+1}$ or $m < a_1$ (then let $i = 0$) or $m > a_k$, then $s_m = \mathtt{B}$ (in the $m > a_k$ case, this is not the only option) and the number of such strings is $\binom{m-i}{n-1}$. If we find prefix sums of values $\binom{j}{n-1} \cdot 2^{-j}$, we can answer such queries in $O(1)$.

If $m > a_k$, then $s_m = \mathtt{A}$ is also possible, and the number of such strings is $\binom{m-k-1}{n-k-1}$. If we find prefix sums of values $\binom{j}{n-k-1} \cdot 2^{-j}$ for each k appearing in the input, we can answer such queries in $O(1)$. There are only $O(\sqrt{n})$ different values of k.

Overall time complexity is $O(n\sqrt{n})$.

Problem Tutorial: "Grp"

The lower bound on the number of groups if k is odd is $\sum_{i=\frac{k+1}{2}}^{k} \binom{n}{i}$: all subsets of size at least $\frac{k+1}{2}$ have to belong to different groups. Similarly, if k is even, the lower bound is $\frac{1}{2}\binom{n}{k/2} + \sum_{i=\frac{k}{2}+1}^{k} \binom{n}{i}$.

Note that $\binom{n}{i} \geq \binom{n}{k-i}$ when $i \geq \frac{k}{2}$. Hence, if we can match all subsets of size i with subsets of size $k - i$ into non-intersecting pairs without common elements, we can achieve the lower bound.

Such a matching always exists when $i \neq k - i$, since the graph is bipartite and "regular" (not exactly, but all vertices in each part have equal degrees). When k is even, the graph is not bipartite, but it turns out that forming $\left\lfloor \frac{1}{2}\binom{n}{k/2} \right\rfloor$ pairs of subsets of size $\frac{k}{2}$ is always possible for $n \leq 17$. Even though the graphs are huge, we can build them and try to find maximum matchings: using Kuhn's algorithm for bipartite graphs, and using Edmonds' blossom algorithm (or maybe Kuhn's algorithm with hacks...) for non-bipartite graphs. Even though time complexity looks big, a greedy initialization already builds a huge part of the matching, and augmenting chains are very short on average too. You can try all possible test cases to make sure your solution is fast enough.

If you know a constructive way to build the matchings, or if you have a proof that an optimal matching of subsets of size $\frac{k}{2}$ for even k always exists, please share!

Problem Tutorial: "Hit"

Let's start with placing points so that each segment contains at least one point greedily: while there's at least one uncovered segment, find the one with the smallest r_i and place a point at r_i. Suppose that the largest number of points inside one of the given segments is t in this placement.

It turns out that the answer is either t or $t - 1$. Indeed, consider points $x_1, x_2, \ldots, x_t$ inside the segment $[l_i, r_i]$ with the largest number of points. Consider the rightmost point with coordinate less than l_i in the optimal placement. Then the next point to the right of it has to be at coordinate at most x_2, the second next point has to be at coordinate at most x_3, ..., the $t - 1$-th next point has to be at coordinate at most x_t. Hence, segment $[l_i, r_i]$ will contain at least $t - 1$ points.

It remains to check if the answer is $t - 1$. Let's compress the ends of segments and go through points from right to left. For each point x, let's answer the following question: if this point is included into the set, is it possible to include some points with coordinates more than x so that each segment with $r_i \geq x$ contains at least one point, and each segment contains at most $t - 1$ points? We'll call points satisfying this condition *legal.*

For each point x, let w_x be the rightmost legal point such that there are no segments strictly inside

$[x, w_x]$. To check the condition for a particular point x, let's start with finding w_x. After that, find $y = w(w(\ldots w(x)\ldots))$ ($t-1$ times). If there exists a segment containing both points x and y, this segment would contain t points if point x was placed, thus, point x is illegal. Otherwise, point x is legal.

Finally, if the point to the left of all segments is legal, we can form a sequence of legal points that is a valid solution for $t-1$, otherwise, the answer is t and we output the initial greedy placement.

Problem Tutorial: “Ineq”

Rewrite the inequality as $a_i x + b_i y \le c_i - 1$. We can see that each triple denotes a half-plane including its border on the (x, y) plane.

The most restricting set of half-planes including the given points can be found by building the convex hull of points (x_i, y_i): our half-planes correspond to its edges then.

We need to check if there exists another integer point $(x', y') \notin S$ inside the convex hull. It is not very easy to find all integer points inside a convex polygon, but instead of finding them, let's just count them using Pick's theorem: if their number is n, the answer is positive, otherwise it's more than n and the answer is negative.

Problem Tutorial: “Joy”

If our position in the queue is fixed, we can use DP. Consider a binary tree with n leaves, and let $f(i, j)$ be the probability that person (leaf) j wins in the subtree of vertex i. Time complexity of such DP is $O(n^2)$. However, if we try all n positions independently, we'll arrive at an $O(n^3)$ solution, which is too slow.

Note that if we fix our position, we know that to become the champion, we have to beat the winners of some subtrees. If we know DP values for these subtrees, we can calculate the probability of becoming the champion in $O(n)$, totalling in $O(n^2)$ for n starting positions.

Finally, note that there are not so many different subtrees we might want to beat. For example, if $n = 16$, we might want to beat all segments of 1 and 2 people, only the following segments of 4 people: $a_{1..4}, a_{5..8}, a_{9..12}, a_{4..7}, a_{8..11}, a_{12..15}$, and the following segments of 8 people: $a_{1..8}, a_{8..15}$. The number of interesting segments is only twice the number of segments in the original DP for one tree, and time complexity of calculating DP for interesting segments is still $O(n^2)$.

Problem Tutorial: “Kilk”

WLOG assume $x \le y$. It's not hard to compute the smallest possible length of the longest substring consisting of equal letters: in fact, it is $k = l(x, y) = \left\lfloor \frac{x+y}{x+1} \right\rfloor$.

Let's fix some value of k and find the required number of strings for all pairs (x, y) such that $l(x, y) = k$. Let $a_k(x, y)$ be the number of strings that have x letters ‘`a`’ and y letters ‘`b`’, don't have substrings of equal letters of length more than k, and end with ‘`a`’. Let $b_k(x, y)$ be defined similarly, except that here we count strings ending with ‘`b`’. Then, the answer for a pair (x, y) is $a_{f(x,y)}(x, y) + b_{f(x,y)}(x, y)$.

Trying all possible lengths of the substring of equal letters at the end of the string, we get the following formulas:

- $a_k(x, y) = \sum\limits_{i=1}^{\min(k,x)} b_k(x-i, y)$;
- $b_k(x, y) = \sum\limits_{i=1}^{\min(k,y)} a_k(x, y-i)$.

Computed in a straightforward way, we can compute the answers for all pairs (x, y) with $x, y \le n$ in $O(n^4)$.

We can use the following two optimizations to speed it up to $O(n^2 \log n)$:

- use prefix sums to find $a_k(x, y)$ faster, or just notice that $a_k(x, y) = a_k(x - 1, y) + b_k(x - 1, y) - b_k(x - 1 - k, y)$ (almost follows from the definition);
- note that we only need $x \le \frac{2n}{k}$, therefore, the number of interesting DP states is about $\sum\limits_{k=1}^{n} \frac{2n^2}{k} = O(n^2 \log n)$.

Time limit is a bit (unnecessarily) strict, so one needs to be careful with implementation.