Problem A. Atcoder Problem

to do

Problem B. Best Problem

First, we change the notations to make the solution easier to understand. For two adjacent bits, if they are `00`, we write down an `R` between them. If they are `11`, we write down an `L`. If they are `01` or `10`, we write down an `X`. WLOG, we add two `1`s to the left of the string, and two `0`s to the right of the string.

Consider the way how `0101` changes:

1. `001010 -> 010100`, which means `RXXXX -> XXXXR`
2. `101011 -> 110101`, which means `XXXXL -> LXXXX`
3. `001011 -> 010101`, which means `RXXXL -> XXXXX`
4. `101010 -> 110100`, which means `XXXXX -> LXXXR`

The new problem is, there is a string, we can move an `R` to 4 cells right, and an `L` to 4 cells left, and the path from the start to the destination must be `X`. If there is an `RL` pair, where `R` is 4 cells left to `L`, they can cancel each other out. For consecutive `X`s, we can produce `LR` pairs from that. By parity, we need to ensure there are odd `X`s between `LR`, and even `X`s between `LL` and `RR` pairs.

We will do operation 3 first. We can cancel out all matched `RL` pairs first greedily. Since the remainder modulo 4 of all positions of `L` and `R` will not change, there will not be useful operation 3 anymore.

Then for two consecutive `R`s, we will move the left `R` to right by the first operations greedily. If we use operations 3 and 4, these newly produced pairs will finally be canceled out. The number of steps will not change. We can do similar things for `L`.

After these greedy steps, the sequence will be turned into something like `L...RR..RR.......L..LL.......RR.........LL.....R.` There are many `LR` groups pointing to each other. For each `L` or `R` group, there will be zero or two `X`s between them.

Then we will consider moving these entire `L` or `R` groups to 4 cells left or 4 cells right until they meet together, and then produce new `LR` pairs by operation 4 at the end.

Finally, we need to do dp on it, to record where each `L` and `R` group will go, and what is the maximum number of steps. The weight is a quadratic function, so it's possible to solve it by convex hull trick. What's more, since we are trying to find the maximum, for some reason, the maximum can be reached on the leftmost or the rightmost position. It means for each pointing pair, only one of `LR` groups will move. Then we can get rid of the convex hull trick and solve it in simply $O(n)$.

Problem C. Cryptography Problem

Let $M = \lfloor \frac{p}{200} \rfloor, x' = (a_1 x - c_1 + M) \bmod p$. So we know $x' \in [0, 2M]$. We can rewrite all other equations in terms of x', and we try to solve x' instead of x. I define x to this x' in the following part.

The basic idea of the solution is you need to find some random linear combinations of equations that make the coefficient small and don't amplify the error term.

If we multiply the i-th equations by y_i and add them together, we can get a new equation $(\sum a_i y_i) \times x + \text{error term} \equiv (\sum c_i y_i) \pmod p$. And the error term will not be greater than $(\sum |y_i|) \times M$. Let $A = (\sum a_i y_i) \bmod p$, $E = (\sum |y_i|) \times M$, $C = (\sum c_i y_i) \bmod p$. If the coefficient A is small enough and the error term E doesn't exceed $p/2$, the solution of this equation can be represented by C intervals $[\frac{C+ip-M}{A}, \frac{C+ip+M}{A}]$.

We can intersect these intervals with the old solution set iteratively, and keep a set of intervals that x can be. If we can find the linear combinations randomly, and intersect them with proper random solutions.

The set will become smaller exponentially. "Proper solutions"here mean if the interval is a bit smaller, you can try to intersect them with larger A. Otherwise, the set may not become smaller very fast.

The next question is how to find these random linear combinations. This technique is always used in hacking rolling hash, which is called multi-tree-attack. Detail here. The brief idea is that you have a set of numbers, and you can sort them, and generate a new set from the difference of near numbers. For example, in this problem, we can generate a new set $\{a_i - a_j | j < i \le i + 5\}$ from a sorted sequence a, and keep the smallest numbers among them. and do it 5 times. It's good enough to pass this problem.

Problem D. Digit Sum Problem

The main idea of this problem is something like sqrt decomposition or meet in the middle.

Let $M = \sqrt{n}$, and $M_2 = 2^p, M_3 = 3^q$ be the nearest power to M. If we listed all multiples of M_2 and M_3, we can divide the interval $1 \sim n$ to $O(\sqrt{n})$ pieces. The question is how to compute the sum for each interval in $O(1)$ with some precalculations.

For each piece, the prefix of numbers in binary and ternary is fixed. So we don't need to consider the prefix. And two endpoints of this interval should be a multiple of M_2 or M_3. So there are at most $O(M_2 + M_3)$ different states. If one end is the multiple of M_2 and the other end is the multiple of M_3, the state is determined by its length. If both ends are multiple of M_2, the state is determined by the remainder modulo M_3 of the first number. It's similar to M_3.

The next question is how to compute answers for these states. We can compute the answer for $(2M_2, M_3)$ or $(M_2, 3M_3)$ from $O(M_2, M_3)$ in $O(M_2 + M_3)$. For example, for all different states of $(2M_2, M_3)$, we can divide it into $O(1)$ intervals by the multiples of M_2 and M_3, and sum them up. So we can start from $(1, 1)$, and increase the smaller power until they both reach $O(\sqrt{n})$.

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

Problem E. Elliptic Curve Problem

First we notice that the condition $[l \le (x \bmod p) \le r] = \lfloor \frac{x-l}{p} \rfloor - \lfloor \frac{x-r-1}{p} \rfloor$. And $1^2, 2^2, \ldots \left(\frac{p-1}{2}\right)^2$ are all different quadratic residues of p.

We can rewrite the problem to $\sum_{i=1}^{(p-1)/2} [l \le (i^2 \bmod p) \le r] = \sum \left(\lfloor \frac{i^2-l}{p} \rfloor - \lfloor \frac{i^2-r-1}{p} \rfloor \right)$.

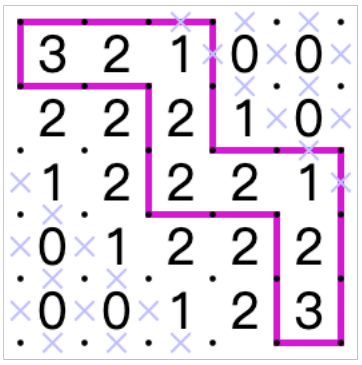
Then we need to compute $\sum \lfloor \frac{i^2+c}{p} \rfloor$ effectively. The technique is quite similar to computing $\sum \lfloor \frac{n}{i} \rfloor$.

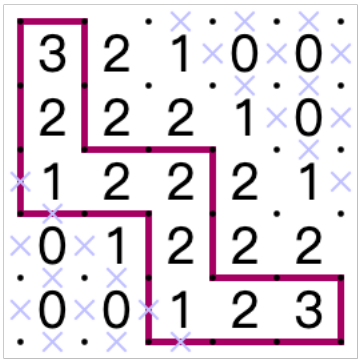
The basic idea is we need to compute the number of integral points between this parabola and the positive x-axis. The region is convex, and the range of coordinates of points are in $[0, p + O(1)]$. So the number of edges of the convex hull is $O(p^{2/3})$. We can find the convex hull and then compute the answer.

We don't include the details of this technique here. To who is interested, there is a short survey(in Chinese) about this technique.

Problem F. Full Clue Problem

The answer:





You may find it by writing a brute force on small n or just playing by hand.

Problem G. Graph Problem

Let A be the adjacent matrix of the graph. The reachability matrix can be something like that $I + A + A^2 + \cdots = (I - A)^{-1}$. We don't consider the convergence here. In our solution, we can choose a random diagonal matrix D, and assign a random weight to each edge (denote this new weighted adjacent matrix by A'). Then we can compute the inverse matrix $M = (D - A')^{-1}$. s can reach t iff $M_{i,j} \neq 0$.

If we delete several vertices of the graph, we will apply a low-rank update on the adjacent matrix, and we need to know the value of some specific elements of this new inverse matrix, which can be done by Woodbury matrix identity:

For an $n \times k$ matrix U and $k \times n$ matrix V, $B = A + UV$, then $B^{-1} = A^{-1} - A^{-1}U(I_k + VA^{-1}U)^{-1}VA^{-1}$.

details to do

For each query, we need to compute the inverse matrix of this intermediate matrix in $O(k_1^3)$, and answer each query in $O(k_1^2)$. The time complexity is $O\left(n^3 + qk_1^3 + \left(\sum k_2\right) k_1^2\right)$.

Problem H. Hard Problem

There are several ways to solve this problem.

The first approach is by the dual of linear programming.

There are several ways to choose intervals (including gapped intervals), denote them be $I_1, I_2, \ldots, I_m$. Let x_i be the number of operations on interval I_i. The constraints is that for every $i (1 \le i \le n)$, $\sum_{i \in I_j} x_j = a_i$. And we want to minimize $\sum x_j$. The solution to this linear programming is integral, you can prove it by yourself.

The dual of this problem is that $\sum_{j \in I_i} y_j \le 1$, and we want to maximize $\sum_{i=1}^{n} y_j a_j$. y_j can be negative here. The solution to this linear programming is also integral, you can prove it by yourself.

The combinatorial meaning is that you need to assign a weight to each position and the sum of weights of all intervals should be not greater than 1.

First we can notice that $y_i \in \{-1, 0, 1\}$. If $y_i < -1$, we change it to -1. Since the sum of every interval is not greater than 1, so the sum of an interval containing y_i is not greater than $1 + (-1) + 1 = 1$.

Let $dp_{i,j,k,l}$ be the maximum sum of $a_i y_i$ if we filled the first i elements, the maximum suffix sum of consecutive interval/odd gapped interval/even gapped interval to i. And we can enumerate the value y_{i+1} and update these three sums.

The time complexity is $O(n)$.

The second approach is by greedy. Details here.

Problem I. Interval Problem

WLOG, we assume $r_1 < r_2 < \cdots < r_n$, and the graph is connected. If the graph is not connected, we can simply compute the answer for each connected component.

First, we consider how to compute the distance from the interval j to the interval $i (j < i)$. We can start from the interval j and jump to the interval that intersects with j and with the rightmost endpoint greedily until it reaches i.

We can notice a tree structure here. For an interval i, let its parent be the rightmost interval that intersects with it.

Then we consider how to compute the distance from i to all intervals j such that $r_j < r_i$. It's not hard to see, for all intervals intersecting with i, the distance is 1, the distance of their children is 2, and so on. So we can compute the answer by something like tree dp and cumulative sum in $O(n)$.

We can compute the distance from i to all intervals j such that $l_j > l_i$ in the same manner. For each interval such that $l_j < l_i$ and $r_j > r_i$, the answer always equals to 1.

So we can solve this problem in $O(n \log n)$.

Problem J. Junk Problem

If there are no bidirectional edges, the problem can be solved simply by LGV lemma. Let $M_{i,j}$ be the number of paths from $(1, a_i)$ to (n, b_j), we can compute the determinant of this matrix.

For the general graph, we can deal with it by Talaska's formula. You'd better read this paper for a better understanding.

The brief idea is we can add two directed edges with weight x for bidirectional edges. The number of paths between two vertices should be replaced by the sum of the weight of all paths. There can be infinite paths. For example, we can walk on a bidirectional edge indefinitely. the sum equals $1 + x^2 + x^4 + \cdots = \frac{1}{1-x^2}$. So the sum of all paths is actually a formal power series. We can see later that we will compute the value when x equals some specific numbers, so we can still compute the sum by simple dp.

From Talaska's formula, the determinant of this new matrix is equal to the collections of nonintersecting and non-self-intersecting paths and cycles, divided by a sum over collections of nonintersecting cycles. To be specific, $\det(M) \times (1 - x^2)^k$ equals

$$\sum_{\text{all disjoint simple paths}} x^{\text{\#bidirectional edge passed}} (1 - x^2)^{\text{\#bidirectional edge unvisited}}$$

It's a polynomial with a degree not greater than $2k$. Let $x = 1$, we can find the answer. But we can not directly substitute x by 1, since the denominator can be 0 during the computation. We can substitute x by $2, 3, \ldots, 2k + 2$ and get the polynomial by interpolation.

The time complexity is $O(n^3 k)$.

Problem K. Knapsack Problem

This problem is solvable by solving the linear programming, then trying to enumerate how to round the solution, or just copying an ILP solver. But they are not intended.

The intended solution is digital dp. A similar problem here.

Problem L. Linear Congruential Generator Problem

First, we can know the result of RNG mod$i$ after i-th execution from the permutation.

Since it's a linear congruential generator, we can get some equations $((a_i \times seed + b_i) \bmod p) \bmod i = c_i$.

We try to solve this from two equations. For example, we choose the first equation M_1 (we can see later, let $M_1 = n/2$ is good enough), let $x = (a_{M_1} \times seed + b_{M_1}) \bmod p$. From $x \bmod M_1 = c_{M_1}$, we can get $x = M_1 t + c_{M_1}$. Here $t \le p/M_1$, which is roughly in $O(p/n)$.

Then we choose another equations $M = M_1 + M_2$, we have $((a_{M_2} \times x + b_{M_2}) \mod p) \mod M = c_M$. We can substitute x, and get $((a_{M_2}(M_1 t + c_{M_1}) + b_{M_2}) \bmod p) \bmod M = c_M$, denoted by $((k_1 t + k_2) \bmod p) \bmod M = c_M$, which is a linear congruent equations of t.

We can carefully choose a_{M_2} to make $k_1 = a_{M_2} M_1 \bmod p$ as small as possible, which is roughly in $O(p/n)$. Then $k_1 t + k_2$ is in $O(p^2/n^2)$. Let $q = \lfloor (k_1 t + k_2)/p \rfloor$, then q is in $O(p/n^2)$, and $(k_1 t + k_2) \bmod p = k_1 t + k_2 - qp$.

We can enumerate q, then solve the equations for $(k_1 t + k_2 - qp) \bmod M = c_M$. We can solve t based on the range of t where $qp \le k_1 t + k_2 < (q+1)p$ and the linear congruent equation. There are about $O(p/k_1/M) = O(1)$ solutions for each q. So we can check all possibilities by brute force. If t is not correct, it will reject in $O(1)$ time.

The time complexity is $O(p/n^2 + n)$.

Problem M. Minimum Element Problem

First, we can see two permutations are equivalent iff their Cartesian trees are the same. If no element is fixed, the answer is simply $C_n = \frac{1}{n+1}\binom{2n}{n}$. Since there is at least one permutation for each binary tree with n vertices.

If $p_x = y$, there are two cases becoming invalid.

- The depth of x is greater than y, since x should be greater than all ancestors.
- The size of the subtree of x is greater than $n + 1 - y$, since x should be less than all descendant.

And these two cases are independent. If x violates both conditions, the number of vertices will be greater than n.

So we solve these two cases separately. We compute the ways that the depth of x equals y for $y = 1 \sim n$, and the size of the subtree of x equals y for $y = 1 \sim n$.

For depth, we consider the path from x to the root. If there are p vertices on the left of x(denoted by $l_1 < l_2 < \cdots < l_p$) and q vertices on the right of x (denoted by $r_1 > r_2 > \cdots > r_q$). Let $l_0 = 0, l_{p+1} = x$, $r_0 = n + 1, r_{q+1} = y$,. The number of ways is

$$\prod_{i=1}^{p} C_{l_i - l_{i-1} - 1} \times \prod_{i=1}^{q} C_{r_{i-1} - r_i - 1} \times \binom{p+q}{p}$$

The elements from $l_{i-1} + 1$ to $l_i - 1$ form a binary tree independently, so the number of ways is $C_{l_i - l_{i-1} - 1}$. The right part is similar. There are $\binom{p+q}{p}$ ways to interleave l and r to form the path from x to the root.

For a fixed p, there is a closed formula (denoted by L_p) for $\sum_l \prod_{i=1}^{p} C_{l_i - l_{i-1} - 1}$, called k-th convolution of Catalan, which is basically something like $\frac{k+1}{n+k+1}\binom{2n+k}{n}$. Details here: https://codeforces.com/blog/entry/87585

So if the depth of x is y, the number of ways is $\sum_{p+q=y-1} L_p \times R_q \times \binom{p+q}{p}$. It's a simple convolution.

For subtree, we can consider shrinking all vertices in the subtree to a single vertex. And this vertex is a leaf.

The subproblem is to compute the number of binary trees with n vertices and k-th vertex is a leaf. Surprisingly, the answer is C_{n-1}, unrelated to k. It is not hard to see we can build a bijection by removing this leaf in a binary tree with n vertices, and adding a leaf in a specific position of a binary tree with $n-1$ vertices.

So if the size of the left subtree of x is p and the size of the right subtree is q, the number of ways is $C_p \times C_q \times C_{n-p-q-1}$. It's also a simple convolution.

Then we solve the problem in $O(n \log n)$.