Problem Tutorial: "Array"

Let $prev_i$ be the position of previous occurrence of a_i (-1 when there is no such position) and $next_i$ be the position of next occurrence of a_i (n + 1 when there is no such position). Apparently, there's no need to change such a_x that $prev_x \neq -1$. So, we can only focus on those a_x where $prev_x = -1$.

1 change a_x to some a_y $(y < x)$

The cost will be $|a_x - a_y| + \frac{next_x(next_x-1)}{2} - \frac{x(x-1)}{2}$. We should make $|a_x - a_y|$ as minimal as possible. It can be done by using a set to maintain all the value before x.

2 change a_x to some a_y $(x \leq y < next_x)$

The cost will be $|a_x - a_y| + \frac{y(y-1)}{2} - \frac{next_x(next_x-1)}{2}$. Assume $a_x \leq a_y$ ($a_x \geq a_y$ is the same), this can be done by using a segment tree to maintain the minimum value of $\frac{y(y-1)}{2} - a_y$.

Problem Tutorial: "Chiaki Chain"

Some observation of the k-th order Chiaki Chain with n vertices and m edges:

- $n + k - 1 = m$
- connected, no loops, no multiple edges;
- each vertex in the cycle has degree 2 except one vertex connected with the sub-chain;
- after removing all the cycles and sub-chains, the remainder is a chain.

Firstly, use a simple depth-first search to find all the cycles and check whether the degree constraint is satisfied.

Then remove the cycles and sub-chains using breadth-first search and we should get a chain or a single vertex (all other graphs we get is invalid).

If we get a single vertex, we should check if the length of sub-chain is greater than 1 when $k = 1$ and check if the length of at least one sub-chain is greater than 1 when $k = 2$ and check if the length of at least two sub-chains is greater than 1 when $k = 3$.

Otherwise, we should enumerate each vertex with degree 3 and check if the sub-chain is valid: the length of the sub-chain is at least 1. In addition, if the vertex is the end of the chain and two sub-chains extend from it, the length of at least one sub-chain should be greater than 1.

Problem Tutorial: "Cut the Plane"

At first, find a line to split the points into two parts – the first part has $\lfloor \frac{n}{2} \rfloor$ points and the second has $\lceil \frac{n}{2} \rceil$.

Then, find a tangent line of the convex hulls of each part and we can use a line to separate both the tangent points. Repeat until only one point left in each part.

Problem Tutorial: "Edges Counting"

Let t_n be the number of trees with n labeled vertices, and its exponential generating function be $T(x) = \sum_{n \geq 1} \frac{t_n}{n!} x^n$. Cayley's formula states that $t_n = n^{n-2}$. Similarly, let rt_n be the number of rooted trees with n labeled vertices, and its exponential generating function be $RT(x) = \sum_{n \geq 1} \frac{rt_n}{n!} x^n$. Since

$rt_n = nt_n$, we can figure out that $RT(x) = xT'(x)$, where $T'(x)$ is the derivative function of $T(x)$ with respect to x.

We call a simple graph generated from a tree with an additional edge "pseudotree", which contains exactly one simple cycle, and if one removes the cycle from the pseudotree, the graph will be changed into a forest. Let p_n be the number of pseudotrees with n labeled vertices, and its exponential generating function be $P(x) = \sum_{n\geq 1} \frac{p_n}{n!} x^n$. An observation is that $P(x) = \sum_{k\geq 3} \frac{RT^k(x)}{k!} \frac{(k-1)!}{2} = \sum_{k\geq 3} \frac{RT^k(x)}{2k}$. The combinatorial meaning is to enumerate the number of vertices on the only cycle, denoted by k, and then the remain parts are k rooted trees, so we can get the conclusion by convolution. Moreover, we can get $P'(x) = \frac{RT'(x)}{1-RT(x)} \frac{RT^2(x)}{2}$.

After that, we can count the number of good graphs. That is to say, let g_n be the number of good graphs with n labeled vertices, and its exponential generating function be $G(x) = \sum_{n\geq 1} \frac{g_n}{n!} x^n$, and we can get $G(x) = \sum_{k\geq 0} \frac{(T(x)+P(x))^k}{k!} = e^{T(x)+P(x)}$. By the way, it can be rewritten as $G'(x) = G(x)(T(x) + P(x))'$.

If we know the number of good graphs, when we count the number of edges on cycles, we could count the contribution of each component. That is to say, we may enumerate a component, which is a pseudotree, and count the number of vertices on its cycle. The generating function should be $\sum_{k\geq 3} k \frac{RT^k(x)}{2k} = \frac{RT(x)}{1-RT(x)} \frac{RT^2(x)}{2}$. Let the generating function be $C(x)$, and we know $G(x)C(x)$ yields the answer.

Due to that the modulus may not be a prime larger than n, so we can hardly calculate each generating function. However, each integer sequence (e.g. t_n, rt_n, p_n, g_n, ...) is easy to get. One can use the binomial coefficients to calculate the convolution. The time complexity is $\mathcal{O}(\max^2\{n\})$.

Problem Tutorial: "Equanimous"

It should be the hardest problem in this contest. The conclusion is that we can construct a deterministic finite automaton (DFA), whose minimal automaton contains only 715 states, and use digit DP on this DFA. To improve the query time complexity, we can prepare more information at first and calculate less information for each query. The time complexity should be $\mathcal{O}(D^2LS + TL)$, where $D = 10$ is the radix, $L = 100$ is the maximal length of a number, $S = 715$ is the number of states on DFA, and T is the number of queries.

To build the DFA, we need to understand how to calculate $f(m)$ for a given integer m firstly. Let $dp(i, j)$ be the possibility (true or false) such that the first i digits of m could form an expression whose absolute value is exactly j. The transition should be $dp(i-1, j) \to dp(i, j + d_i), dp(i-1, |j - d_i|)$, where d_i is the i-th digits of m. Actually, we don't need to memorize the information of $j \geq K$ for some integer K. We know $0 \leq f(m) \leq 9$, and the worst case for this DP occurs when m contains consecutive digits like $\underbrace{99\ldots9}_{8 \text{ times}}\underbrace{88\ldots8}_{9 \text{ times}}$, so we can just set K as $(8 \times 9 + 1)$.

After this improvement, the information of each integer (for the sake of transition) could be represented as a 01-sequence of length 73. Actually, the number of different sequences is fairly small, so we can build a DFA consisting of these sequences and corresponding transitions. Inspired by the aforementioned worst case, we can minimize the DFA using partition refinement through at most 9 iterations. The remaining part is just a typical digit DP problem.

Problem Tutorial: "Fighting Against Monsters"

It is easy to prove that no more than 100 seconds the first two monsters must die.

Let $dp(i, j, k)$ be the minimum total damages the hero suffered when it is the i-th second, the first monster has suffer a total damage j and the second monster has suffer a total damage k.

If $HP_C \leq 100$, it is easy. If $HP_C > 100$, just enumerate all i, j and k, make sure that the first two monsters are dead and then use greedy to attack the boss.

Problem Tutorial: "Mysterious Triple Sequence"

It's a bit hard to solve the problem directly. Let's reduce this problem into the **discrete logarithm** problem which is easier to solve in **sqrt** time.

Lemma 1: Let $f_0 = 0, f_1 = 1, f_{n+2} = 2f_{n+1} + f_n$ $(n \in \mathbb{N}^+)$. It shows that $(a_k, b_k, c_k) = (f_{2^k+1}, f_{2^k}, f_{2^k-1})$.

Brief Proof. There are many approaches to prove that (and further things), so I'd like to leave a type of brief introduction here.

Let $F = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}$. It can be proved by induction that $F^n = \begin{bmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{bmatrix}$ $(n \in \mathbb{N}^+)$.

There is an observation that $\begin{bmatrix} f_{n-1} & f_n \\ f_n & f_{n+1} \end{bmatrix} \begin{bmatrix} f_{m-1} & f_m \\ f_m & f_{m+1} \end{bmatrix} = \begin{bmatrix} f_{n-1}f_{m-1} + f_nf_m & f_{n-1}f_m + f_nf_{m+1} \\ f_nf_{m-1} + f_{n+1}f_m & f_nf_m + f_{n+1}f_{m+1} \end{bmatrix}$ which is helpful to prove $(a_k, b_k, c_k) = (f_{2^k+1}, f_{2^k}, f_{2^k-1})$ by induction.

Lemma 2: Sequence $\{(f_{k+1}, f_k)\}_{k=0}^{\infty}$ is a pure periodic sequence in modulo any integer p.

Brief Proof. Assuming that there exist two distinct integers x and y such that $(f_{x+1}, f_x) \equiv (f_{y+1}, f_y) \pmod{p}$, it can be proved that $(f_{x+k+1}, f_{x+k}) \equiv (f_{y+k+1}, f_{y+k}) \pmod{p}$ for any positive integer k just by using matrix multiplication of matrix F.

Furthermore, it is easy to show that $F^{-1} = \begin{bmatrix} -2 & 1 \\ 1 & 0 \end{bmatrix}$ always exists in modulo any integer p and thus it is possible to extend the above k from any positive integer $(k \in \mathbb{N}^+)$ to any integer $(k \in \mathbb{Z})$.

Lemma 3: Let $L(p)$ be the minimal positive period of sequence $\{(f_{k+1}, f_k)\}_{k=0}^{\infty}$ in modulo integer p. It shows that $L(p) \leq \frac{8}{3}p$ for any integer p.

Brief Proof. If we replace sequence $\{f_n\}$ by Fibonacci sequence, it will be a classic conclusion that $L(p) \leq 6p$. In addition, it is not difficult to conclude the formula of the period for $\{f_n\}$ by a similar approach. If you want to know more details, please check "Pisano period - Wikipedia" and "The Period of the Fibonacci Sequence Modulo j". If not, the following are conclusions.

Case 1. If $p = 2$, $L(p) = 2$.

Case 2. If p is an odd prime such that there exists an integer x satisfying $x^2 \equiv 2 \pmod{p}$, $L(p)$ is a divisor of $(p-1)$. In that case, $p \equiv 1$ or $7 \pmod{8}$.

Case 3. If p is an odd prime such that there exists no integer x satisfying $x^2 \equiv 2 \pmod{p}$, $L(p)$ is a divisor of $2(p+1)$. In that case, $p \equiv 3$ or $5 \pmod{8}$.

Case 4. If $p = q^e$ such that q is a prime and e is an integer greater than 1, $L(p)$ is a divisor of $q^{e-1}L(q)$.

Case 5. If $p = q_1^{e_1}q_2^{e_2}\cdots q_m^{e_m}$ such that $q_1, q_2, \cdots, q_m$ are distinct primes, $L(p)$ equals to the least common multiple of $L(q_1^{e_1}), L(q_2^{e_2}), \cdots, L(q_m^{e_m})$.

The worst case such that $\frac{L(p)}{p}$ is maximum occurs when $p = q^e$, q is a prime, e is an integer and $q \equiv 3$ or $5 \pmod{8}$. In that case, $\frac{L(p)}{p} \mid \frac{2(q+1)q^{e-1}}{q^e} = \frac{2q+2}{q} \leq \frac{8}{3}$.

By the way, please note that $L(181^4) = 2158425724$, which is actually the only one case such that $p \leq 2^{30}$ but $L(p) \geq 2^{31}$ (random tests might not cover this case), so be careful with the 32-bit integer.

Solution:

We can reduce the problem into two stage:

1. First, find the integer u $(0 \leq u < L(p))$ such that $(x, y, z) \equiv (f_{u+1}, f_u, f_{u-1}) \pmod{p}$. Because $\{(f_{k+1}, f_k)\}_{k=0}^{\infty} \pmod{p}$ is pure periodic, you can use "Baby-step giant-step algorithm" (a type of meet-in-the-middle algorithm) directly.
2. Then, find the integer k $(k \geq m)$ such that $2^k \equiv u \pmod{L(p)}$. Because $\{2^k\}_{k=0}^{\infty}$ is not pure periodic in modulo $L(p)$ ($L(p)$ is always even), you should enumerate the acyclic part and reduce $L(p)$ into an odd number and then use Baby-step giant-step algorithm.

However, solution in time complexity $\mathcal{O}(n\sqrt{p}\log p)$ or $\mathcal{O}(n\sqrt{p})$ would be rejected. Actually, this type of meet-in-the-middle algorithm could be optimized easily for multi-queries. Let's set a threshold T first (we will determine an optimal value for it soon).

In the first stage, we need to find the least power of the invertible matrix F which equals to a given matrix G. Assuming that $F^{xT+y} = G$ $(0 \le y < T)$, we have $F^y = G(F^{-T})^x$. After precalculating $F^0, F^1, \cdots, F^{T-1}$, you can just enumerate x and check if y exists (by binary search or hash). The time complexity for multi-queries is $\mathcal{O}(T\log T + n\frac{L(p)}{T}\log T)$. (note: there are only two element in F^y is necessary to memorize)

In the second stage, we need to find the least power of 2 which equals to a given number u. let $L(p)$ be $2^\alpha L'(p)$ such that $L'(p)$ is an odd integer. If there exists integer k such that $0 \le k < \alpha$, $2^k \equiv u \pmod{L(p)}$, that is finished. If not, we can reduce the problem into $2^{k-\alpha} \equiv \frac{u}{2^\alpha} \pmod{L'(p)}$ (the minimal positive period of 2 will be a divisor of $\varphi(L'(p))$) and then solve it in time complexity $\mathcal{O}(T\log T + n(\alpha + \frac{L'(p)}{T}\log T))$ for multi-task.

To approximate optimal performance, we can just set T as $\sqrt{np}$ and the time complexity can be revised as $\mathcal{O}(\sqrt{np}\log\sqrt{np})$ or $\mathcal{O}(\sqrt{np})$.

By the way, you can get $L(p)$ and the period of 2^k in modulo $L'(p)$ by Baby-step giant-step algorithm instead of utilizing some conclusions in number theory (if you'd known or guessed that $L(p) \le \frac{8}{3}p$, :P).

Problem Tutorial: "Inner Product"

If we enumerate an edge (u, v) in the first tree, remove it from the first tree, the first tree will split into two parts. And if we color the vertex in the second tree according the part which it belong to in the first tree, the contribution of (u, v) will be

$$w_1(u,v)\sum_{x=1}^{n}\sum_{y=1}^{n}[col_x \neq col_y]d_2(x,y)$$

Use the technique **dsu on tree** to maintain the color of each vertex in the second tree, and at the same time use **centroid decomposition** to maintain the sum of distance between two vertices with different color in the second tree.

The time complexity will be $O(n\log^2 n)$. There also exists an $O(n\log n)$ solution, we leave it for the readers.

Problem Tutorial: "Counting Polygons"

Formally, this problem wants you to count the number of distinct integer sequences $A = [a_0, a_1, \ldots, a_{m-1}]$ meeting the following conditions:

- $\sum_{i=0}^{m-1} a_i = n$, where $a_i \in \mathbb{Z}^+$;
- $2\max\{a_0, a_1, \ldots, a_{m-1}\} < n$;
- if there exists another integer sequence $B = [b_0, b_1, \ldots, b_{m-1}]$ meeting the above conditions and an integer k such that $a_i = b_{(i+k) \bmod m}$ for $i = 0, 1, \ldots, m-1$, then A and B are considered same and they should be counted only once;
- if there exists another integer sequence $B = [b_0, b_1, \ldots, b_{m-1}]$ meeting the above conditions and an integer k such that $a_i = b_{m-1-i}$ for $i = 0, 1, \ldots, m-1$, then A and B are considered same;
- if A and B are considered same, and B and C are considered same, then A and C are considered same.

If we ignore the second condition, we can count the number using Burnside's lemma. To deal with the second condition, we can use the Inclusion-Exclusion principle, since there may be at most one a_i that breaks the condition. In this case, we don't need to consider all the equivalence classes.

By the way, solution with time complexity $\mathcal{O}\left(\max\{n\} + \sum_T d(\gcd(n,m))\right)$ should be accepted, where $d(x)$ is the number of divisors of x. It can get an accepted verdict because $d(x) \leq 500$ for $x \in [1, 10^7]$.

Problem Tutorial: "Square Graph"

Recall the method to extract all the squares using suffix array: assume the half length of the square is L, find the longest common prefix and the longest common suffix of substring $s[iL, (i+1)L)$ and $s[(i-1)L, iL)$, and the start position (it is an interval) can be found using the longest common prefix and the longest common suffix.

Let's enumerate the half length L of the square according to sorted order of w_i and find all the squares. We can get a set of triples (x, y, L) which means for all i in $[x, y]$, i and $i + L$ should be connected with an edge of weight L.

Next, let's try to use two kinds of transformations which we refer to as **Split** and **Reduce** to solve this subproblem for L.

A **Split** operation transforms a single triple into an equivalent system of shorter triples. For a triple (x, y, L), we can find an integer k such that $2^k < y - x + 1$ and $2^{k+1} \geq y - x + 1$. The triple can be spilted into two triples $(x, x + 2^k - 1, L)$ and $(y - 2^k + 1, y, L)$.

A **Reduce** operation will remove useless triples in a set of triples with the same interval length. Since the length of all the intervals is the same, we can only care about the left end the interval. Make a graph by connected x and $x + L$ and find the spanning forest of the graph and only the triples in the spanning forest is usefull.

Repeat those two transformations until all the length of interval equals to 1, we solve this subproblem.

For the whole problem, we can maintain a disjoint union set for every k from 0 to $\lfloor \log_2 n \rfloor$, and share the disjoint union set to find the spanning forest.

The time complexity will be $O(n \log n + n\alpha(n))$.

Problem Tutorial: "Three Dimensions"

Let $dp(i, S, T)$ be the sum when

- first i bits are considered,
- $S = (i_1, i_2, \ldots, i_9)$ where i_1, i_2 and i_3 means the sign of $|x_a - x_b|$, $|y_a - y_b|$, and $|z_a - z_b|$ respectively; i_4, i_5 and i_6 means whether $|x_a - x_b|$, $|y_a - y_b|$, and $|z_a - z_b|$ is the maximum value; i_7, i_8 and i_9 means whether we have to borrow a bit from the $(i-1)$-th bit when doing subtraction.
- $T = (i_1, i_2, \ldots, i_6)$ where i_k means whether the k-th number from $x_a, y_a, z_a, x_b, y_b, z_b$) exceeds the limit.

The recurrence equation is quite straightforward.