Problem Tutorial: "Airport Check-in"

$$answer = \text{Prob(Pusheen will choose fastest counter)} = \sum_{k=1}^{n} \text{Prob(the counter } i \text{ is the fastest}$$

$$\text{and will finish work with current passenger faster than others)}$$

$$\text{Prob(the counter } i \text{ is the fastest...)} = \frac{1}{r-l}\int_l^r dt \cdot \frac{1}{t}\int_0^t dx \cdot \text{Prob(all other counters work slower and finish later)}$$

$$\text{Prob(all other counters...)} = \text{Prob(one other fixed counter work slower and finish later)}^{n-1}$$

$$\text{Prob(one other fixed counter...)} = \frac{1}{r-l}\int_t^r dt_2 \frac{t_2 - x}{t_2}$$

$$answer = n\frac{1}{(r-l)^n}\int_l^r dt \cdot \frac{1}{t}\int_0^t dx \ \cdot \ \left(\int_t^r dt_2 \frac{t_2 - x}{t_2}\right)^{n-1}$$

Solution 1 (intended): Calculate the integral manually.

Solution 2: Do numerical integration over t and inside, calculate integral precisely for given t.

Problem Tutorial: "Beyond the Rescue"

First, we should notice that actually this task is not about tree, but about line segment: we can get rid of everything except path from s to t, because assassin will use only this path. Now we have a sequence of edges, and a cyclic route of guards, where each segment of guards' route is a subsegment of edges of path from s to t, in direct or reverse direction.

Now we will go from s to t and determine minimal time, when we can go from vertex v_i to v_{i+1}. How to solve this task in nk time? Suppose passed time has remainder t modulo p, where p is length of guards' cyclic path. We can check for all guards' checkpoints a_j and a_{j+1} if guard will pass through edge from v_i to v_{i+1} in between these checkpoints. We want to choose earliest pair of guards' consecutive checkpoints so assassin will be able to pass through this edge between two consecutive guards' visits of this edge. It can be done by careful looking at some formulas.

We can notice that guards will always go between v_i and v_{i+1} in alternate directions. We can store all active guards' segments in segment tree (there will be two types of segments — direct and reverse), and after that everything we need is to find first position in segment tree where assassin will have enough time while guards are off this edge. It can be done by descent in segment tree, after writing out some formulas.

Problem Tutorial: "Casino Cheating"

First of all, since you know interactor's behaviour, probably you want to write your own interactor to test task locally.

First observation: if you act greedily, by taking each turn $\frac{2}{3}$ of opponent's maximal piece, you will get around $\frac{1}{2}$ of chocolate in the end.

Second observation: if you in first turn take $\frac{1}{3}$ of opponent's only piece, and in last turn you take $\frac{2}{3}$ of remaining from first turn piece, you will get $\frac{4}{9}$ chocolate in last turn. Everything you have to do is to get at least $\frac{1}{10}$ chocolate in previous turns. It can be done by using previous greedy solution. Combination of these two heuristics will give you around 0.6 chocolate in the game.

Problem Tutorial: "DotA Quals"

The mathematical expectation of the number of rounds can be expressed as sum of probabilities of participation in each round:

$$res = \sum_{i=1}^{n} \text{Prob(Idned will participate in round } i)$$

This probability can be easily calculated using binomial coefficients. For example,

$$\text{Prob(Idned will participate in round } i) = \frac{\binom{\text{number of players worse than Idned}}{\text{number of players indirectly overtaken by each winner of round } i}}{\binom{\text{number of all players except us}}{\text{number of players indirectly overtaken by each winner of round } i}}$$

To achieve small error we can pre-calculate $\log(m!)$ for each m from 0 to 2^n. After that, we can easily solve the problem without huge loss of precision.

Problem Tutorial: "Enumeration of Tournaments"

Let $f(n)$ be the answer to the problem.

$$f(2m) = (2m-1)!! \cdot f(m)$$
$$f(2m+1) = (2m+1)!! \cdot f(m+1)$$

This way, problem reduces to «how to calculate $(2m-1)!!$ mod 2^{64}».

There are multiple ways to calculate that, let's we'll discuss one of them.

$$1 \cdot 3 \cdot 5 \cdot 7 \cdot ... \cdot (2m-1)$$

Let's split everything into blocks of 2^{16} numbers, one block:

$$block = [(k \cdot 2^{17} + 1)(k \cdot 2^{17} + 3) \dots (k \cdot 2^{17} + 2^{17} - 1)]$$

Let $q = 2k + 1$

$$block = [(q \cdot 2^{16} - (2^{16} - 1))(q \cdot 2^{16} - (2^{16} - 3)) \dots (q \cdot 2^{16} + (2^{16} - 1))]$$

And that can be divided into pairs $(q \cdot 2^{16} - x)(q \cdot 2^{16} + x) = (q^2 \cdot 2^{32} - x^2)$

$$block = [(q^2 \cdot 2^{32} - 1^2)(q^2 \cdot 2^{32} - 3^2) \dots (q^2 \cdot 2^{32} - (2^{16} - 1)^2)]$$

Which can be expressed as $A + B \cdot q^2 \cdot 2^{32}$ for some fixed A and B (which can be calculated).

$$(2m-1)!! = block[k=1] \cdot block[k=2] \cdot ... \cdot block[k=...] \cdot [\text{incomplete block}]$$

The last one (incomplete block) can be calculated manually. The product of first several blocks can be easily calculated using A, B and formula

$$1^2 + 3^2 + 5^2 + ... + (2k-1)^2 = \frac{k(4k^2-1)}{3}$$

There is another way to calculate $(2k+1)!!$. Let $P_k(x) = (2x+1) \cdot (2x+3) \cdots (2x+2k-1) = \prod_{i=0}^{k-1}(2x+2i+1)$. $(2k+1)!! = P(0,k)$. Clearly, $P_k(x)$ is a polynomial.

$P_0(x) = 1$, $P_{2k}(x) = P_k(x) \cdot P_k(x+k)$, $P_{2k+1}(x) = (2x + 4k + 1) \cdot P_{2k}(x)$.

We can calculate coefficients of $P_k(x)$ using these formulas. Note that x everywhere appears with coefficient 2, and we calculate coefficients modulo 2^{64}. Therefore, all coefficients before x^i where $i \geq 64$ are equal to zero.

Each polynomial has length $w = 64$, so computing $(2n + 1)!!$ works in $O(w^2 \log n)$ time.

Problem Tutorial: "Fresh Matrix"

This task looks like a standard dynamic programming by profile. You can start with writing dynamic programming with state "which cells are blocked in column, and which non-blocked cells in column are connected with each other".

Actually, when you wrote this dynamic, you encounter with one problem: there are too much (aroun 2k) states to directly multiply matrices, which is common way to deal with such problems.

As you may know, almost every time when you have problem which can be solved by matrix exponentiation, there is linear recurrence on the answer values. You can calculate first few thousands of answers for small m (it can be done fast enough, since there are not so many transitions in dynamic), and ensure that there is linear recurrence on the answer.

If there were not so many states, you could use gauss elimination algorithm to solve linear system and find coefficients of linear recurrence, but linear recurrence has length around 900, and gauss shouldn't fit in time. Instead of this, you can notice that linear system has very specific type, where each row is just shift of previous row. This system can be solved faster, e.g. you can use Berlekamp-Messy algorithm, which works in $\mathcal{O}(n^2)$ time.

After you got coefficients of linear recurrence, you can calculate m-th term of sequence easily by $\mathcal{O}(n^2 \log m)$ (or even $\mathcal{O}(n \log n \log m)$ with some FFTs) time. You just have to calculate $x^m \mod M(x)$, where coefficients of $M(x)$ are coefficients of recurrence in right order.

Problem Tutorial: "Game of Chairs"

Let's compute array d: $d[i]$ is the sum of distances from position i to every color. The answer is $\frac{min\ d[i]}{c}$.

Let's fix some color z and for each i add to $d[i]$ the distance from i to the nearest chair of color z. Let $x_1 < x_2 < \cdots < x_m$ be positions of chairs of color z. We should add some arithmetic progressions to d:

1. Add $x_1 - 1, x_1 - 2, \ldots, 0$ to $d[1 \ldots x_1]$
2. Add $0, 1, 2, \ldots$ to $d[x_k \ldots \lfloor \frac{x_k + x_{k+1}}{2} \rfloor]$
3. Add $\ldots, 2, 1, 0$ to $d[\lfloor \frac{x_k + x_{k+1}}{2} \rfloor + 1 \ldots x_{k+1}]$
4. Add $0, 1, 2, \ldots$ to $d[x_m \ldots n]$

It can be done in $O(n)$ time for each color, so total time is $O(n^2)$.

Let's improve this solution. We will compute an array d' such that $d[i] = \sum_{k=1}^{i} d'[k]$. To add arithmetic progression $b, b + y, b + 2y, \ldots$ to $d[l \ldots r]$, we should add b to $d'[l]$, add y to $d'[l + 1], \ldots, d'[r]$ and add $-(b + (r - l)y)$ to $d'[r + 1]$.

The solution is still $O(n^2)$. But d' can be obtained using the same approach.

We will compute an array d'' such that $d'[i] = \sum_{k=1}^{i} d''[k]$. To add b to $d'[l \ldots r]$, we should add b to $d''[l]$ and add $-b$ to $d''[r + 1]$.

Now we can process every color in $O(m)$ where m is the number of chairs of that color. Obtaining d' from d'' and d from d' is done only once in the end, so the solution works in $O(n)$ time.

Problem Tutorial: "Hung Fu"

If for some $x = p_i$ we have $y = p_j$ $(i \neq j)$ as the optimal element in minimum part of the formula, let's call y as the parent of x, or x has -1 parent if $i = j$. Since $j \leq i$, parent links will produce an oriented tree with -1 as the root. So if for each ordered pair (i, j) we determine the cost of making j the parent of i and find the cheapest oriented subtree of this complete graph, we'll end up with some solution. And if there is a better solution, it would produce a better tree, so we already have the best one.

There is a simple implementation of algorithm that finds the cost of the minimal oriented subtree in $\mathcal{O}(n^3)$ time. To find the smallest permutation you can try to extend the fixed prefix and find the cost of the subtree with additional constraints. So the resulting time will be $\mathcal{O}(n^5)$ (which is already fine). You can improve time complexity by either using the quicker algorithm for searching minimal subtree or by replacing one loop with binary search.

Problem Tutorial: "Is It a p-drome?"

To solve this problem you can use hashes. For example, we can come up with the function f so $f(s)$ is equal to 0 if s is p-drome and $f(s) \neq 0$ if s is not p-drome with high probability. We should be able to compute f for all substrings of given length.

Such function $f(s)$ can be scalar product with some vector: $f(s) = \sum f_i s_i$. We should ensure that $s_i = s_{p_i}$. We can see that $s_i = s_{p_i}$ is almost equivalent to $y \cdot s_i \equiv y \cdot s_{p_i} \mod P$, where P is prime modulo and y is some constant.

Let's for all i add generate random y and add $+y$ to f_i and $-y$ to f_{p_i}. Now if s is p-drome, obviously, $f(s) = \sum f_i s_i = 0$. If s is not p-drome, $f(s)$ will be somewhat random number in $[0; P-1]$, and we will get a false-positive result with low probability.

To calculate all $f(s)$ you can use one `FFT` multiplication.

Problem Tutorial: "Journey"

First, let's understand what a path is. The process of motion can be divided into segments, within we move, jumping to the same number. Hence let's consider $dp[i]$, the number of ways to get to the cell i, and change jumping number in this cell. From the state $dp[i]$ let's make transitions to the states $dp[i + a[i] * k]$ where $(1 \leq k)$ and $(i + a[i] * k \leq n)$ However, there are situations that we came to the cell y with number h and $a[x] = h$. So we have a problem, we don't want to count transition from x to $x + a[x]$ twice. Let's think if this two number are equal then we will definitely change it. It means we have to break iterating when we reach a cell with the same number.

Let's prove that this algorithm works in O $(n\sqrt{n})$. If $a[i] > \sqrt{n}$, then for each position we do no more than $\sqrt{n}$ transitions. If $a[i] \leq \sqrt{n}$, then it is obvious that in each cell we can go no more than once for each jump size. It follows that the total number of transitions is again no more than $O(n\sqrt{n})$.

Problem Tutorial: "Kid's Nightmare"

Graphs G in the statement are usually called `chordal graphs` (because every cycle has a chorde). They have a lot of interesting properties.

The most important property is that such graphs always have `PEO` (Perfect Elimination Order). `PEO` is such a permutation of vertices, so each vertice v and all its neighbours, which come after v in permutation, form a clique.

It is easy to see, that these graphs can have only linear number of maximal by inclusion cliques (in every clique there is vertice v which comes earlier than others in PEO, so clique should be vertice v and all its neighbours which are after v).

There is algorithm to construct `clique tree` of given graph. `Clique tree` T is tree, vertices of which are maximal by inclusion cliques of G. As each vertice u of T is clique, we can associate with it vertices $v_1, \ldots, v_{k_u}$ from G. Important property of `clique tree` is that every vertice v from G belongs to connected

subtree of vertices in T. It can be proved that summary count of vertices in all maximal cliques is linear by number of edges if G.

Now, we have a tree T with some sets of colors (actually by "color" we mean vertice v in old graph) in vertices, and every color belongs to connected subtree of T. We can reformulate the problem on T: set of vertices $v_1, \ldots, v_k$ in G has no cycles if and only if in every vertice of T there is at most two of chosen vertices from G (it is obvious: if there is a cycle in G, it belongs to some clique, and if there are three vertices in clique, there is a cycle).

This problem can be solved by dynamic programming on tree T. One of ways is to use state "maximal number of colors we can choose in a subtree of u, if we have chosen colors v_1 and v_2 in vertice u". There are around $\mathcal{O}(m\sqrt{m})$ of such states, because sum of colors in all vertices is $\mathcal{O}(m)$, and there is no more than $\sqrt{m}$ colors in each vertice.

This dynamic programming can be calculated in $\mathcal{O}(m\sqrt{m})$ time. To do that, for each state you should determine which colors are contained in parent of this vertice, and add value of dynamic to corresponding states in parent.

You can read "Chordal graphs and their clique graphs" article for understanding how everything works in chordal graphs.

Problem Tutorial: "Lazy Student"

Let $f(k)$ be the answer to the problem.

$$f(1) = 1$$
$$f(k+1) = \min_t \left(t \cdot t + (1-t) \cdot f(k)\right)$$

where t is the amount of topics Raccoon will learn before the first attempt. This gives us

$$f(k+1) = f(k) - \frac{f(k)^2}{4}$$

This recurrent sequence has a relatively small period modulo 10^9+7. (Intuition is that it should be around $\sqrt{mod}$)

Outline. Calculate $\frac{P}{Q}$ modulo 10^9+7. Calculate Q modulo 10^9+7 (can be done relatively easy). Use $P = \frac{P}{Q} \cdot Q$ to calculate P.