Problem A. A Boring Number Game

Problem Setter: Rumman Mahmud
Tester: Raihat Zaman Neloy, Sharif Minhazul Islam
Alternate Writer: Irfanur Rahman Rafio
Category: Math, Combinatorics
Total Solved: 41
First to Solve: IOI1

The expected value of the length of the final list is: $n - \frac{n-1}{k}$. How?

After some mathematical calculations, we can derive a formula for the expected value that is not hard to find.

$$\frac{\sum_{l=1}^{n} l \cdot \binom{n-1}{l-1} \cdot k \cdot (k-1)^{(l-1)}}{k^n}$$

Let's focus on the numerator:

$$S_n = k \cdot \sum_{l=1}^{n} l \cdot \binom{n-1}{l-1} \cdot (k-1)^{(l-1)} \quad \dots \dots \dots \quad \text{(I)}$$

Now, let's consider the binomial expansion of $(x+1)^n$:

$$(x+1)^n = \sum_{p=0}^{n} \binom{n}{p} \cdot x^p$$

Let's differentiate both sides with respect to x:

$$n \cdot (x+1)^{n-1} = \sum_{p=1}^{n} p \cdot \binom{n}{p} \cdot x^{p-1}$$

If we substitute x by $k-1$ and p by l we get:

$$n \cdot k^{n-1} = \sum_{l=1}^{n} l \cdot \binom{n}{l} \cdot (k-1)^{l-1} \quad \dots \dots \dots \quad \text{(II)}$$

This is quite close to our numerator. But not close enough! Let's try to restructure $\binom{n}{l}$ and see what happens.

We know that:

$$\binom{n}{l} = \binom{n-1}{l-1} + \binom{n-1}{l}$$

We can rewrite it to:

$$\binom{n-1}{l-1} = \binom{n}{l} - \binom{n-1}{l}$$

Let's substitute $\binom{n-1}{l-1}$ with $\binom{n}{l} - \binom{n-1}{l}$ in equation (I).

$$S_n = k \cdot \sum_{l=1}^{n} l \cdot \binom{n}{l} \cdot (k-1)^{(l-1)} - k \cdot \sum_{l=1}^{n} l \cdot \binom{n-1}{l} \cdot (k-1)^{(l-1)} \quad \dots \dots \dots \quad \text{(III)}$$

Using equation (II) we can derive that:

$$k \cdot \sum_{l=1}^{n} l \cdot \binom{n}{l} \cdot (k-1)^{(l-1)} = k \cdot n \cdot k^{n-1} = n \cdot k^n$$

Let's substitute this back to equation (III):

$$S_n = n \cdot k^n - k \cdot \sum_{l=1}^{n} l \cdot \binom{n-1}{l} \cdot (k-1)^{(l-1)}$$

Since $\binom{n-1}{n} = 0$, we can rewrite the second term of the above equation as:

$$k \cdot \sum_{l=1}^{n-1} l \cdot \binom{n-1}{l} \cdot (k-1)^{(l-1)}$$

Let's substitute the second term:

$$S_n = n \cdot k^n - k \cdot \sum_{l=1}^{n-1} l \cdot \binom{n-1}{l} \cdot (k-1)^{(l-1)} \quad \dots \dots \dots \quad \text{(IV)}$$

The second term is very close to equation (II). Let's substitute n with $n-1$ in equation (II):

$$(n-1) \cdot k^{n-2} = \sum_{l=1}^{n-1} l \cdot \binom{n-1}{l} \cdot (k-1)^{l-1}$$

Multiply both sides by k:

$$(n-1) \cdot k^{n-1} = k \cdot \sum_{l=1}^{n-1} l \cdot \binom{n-1}{l} \cdot (k-1)^{l-1}$$

Which is exactly equal to the second term of the equation (IV). Let's substitute this in equation (IV):

$$S_n = n \cdot k^n - (n-1) \cdot k^{n-1}$$

Now that we found our numerator sum, let's simplify the expected value:

$$E = \frac{S_n}{k^n} = \frac{n \cdot k^n - (n-1) \cdot k^{n-1}}{k^n} = n - \frac{n-1}{k}$$

What remains is to find the mod 998244353 part, which can be done in $\mathcal{O}(1)$.

Problem B. Bijgonit

Problem Setter: Irfanur Rahman Rafio
Tester: Raihat Zaman Neloy, Hasinur Rahman
Alt Writer: Arnob Sarker, Rumman Mahmud
Category: Math
Total Solved: 130
First to Solve: NSU_FiNal_sTraw

Let $A = X^6 - Y^6$ and $B = X^4 - Y^4$. The maximum possible value of A is $5000^6 - 0^6 = 5^6 \times 10^{18} = 1.5625 \times 10^{22}$, which is outside the range of `long long int` data type.

Boring Solution

The most dull way to solve this is by using large data types such as `__int128` from C++.

Cool Solution

Let's do some bijgonit (algebra) to factorize A and B.

$A = X^6 - Y^6$
$= (X^3 + Y^3)(X^3 - Y^3)$
$= (X + Y)(X^2 + XY + Y^2)(X - Y)(X^2 - XY + Y^2)$
$= (X^2 - Y^2)(X^4 + X^2Y^2 + Y^4)$

$B = X^4 - Y^4$
$= (X^2 + Y^2)(X^2 - Y^2)$

It may look like $gcd(A, B) = X^2 - Y^2$, but there's a catch!
In algebraic factorization, X and Y are different expressions that are considered co-prime. Here, this assumption may not be true. Since X and Y are not guaranteed to be co-prime, $(X^2 + Y^2)$ and $(X^4 + X^2Y^2 + Y^4)$ are not guaranteed to be co-prime either.

$\therefore gcd(A, B) = (X^2 - Y^2) \times gcd(X^2 + Y^2, X^4 + X^2Y^2 + Y^4)$

Here, the value of $X^4 + X^2Y^2 + Y^4$ must be less than $3 \times 5000^4 = 1.875 \times 10^{15}$. The value of $gcd(X^2 + Y^2, X^4 + X^2Y^2 + Y^4)$ cannot exceed $X^2 + Y^2$, so $gcd(A, B)$ cannot exceed $(X^2 + Y^2)(X^2 - Y^2) = X^4 - Y^4$. The maximum possible value for this is $5000^4 - 0^4 = 6.25 \times 10^{14}$. All of these values fall inside the range of `long long int` data type.

Cooler Solution

Let, $g = gcd(X, Y)$, $x = X/g$ and $y = Y/g$. Now, x and y are co-prime. Let's factorize A and B again.

$A = (X^2 - Y^2)(X^4 + X^2Y^2 + Y^4)$
$A = (g^2x^2 - g^2y^2)(g^4x^4 + g^4x^2y^2 + g^4y^4)$
$A = g^6(x^2 - y^2)(x^4 + x^2y^2 + y^4)$

$B = (X^2 + Y^2)(X^2 - Y^2)$
$= (g^2x^2 + g^2y^2)(g^2x^2 - g^2y^2)$
$= g^4(x^2 + y^2)(x^2 - y^2)$

Now that x and y are co-prime, it might be tempting to assume that $gcd(A, B) = g^4(x^2 - y^2)$. But there's another catch!
x and y are co-prime with each other, but not guaranteed to be co-prime with g.

$\therefore gcd(A, B) = g^4(x^2 - y^2) \times gcd(x^2 + y^2, g^2(x^4 + x^2y^2 + y^4))$
When x and y are co-prime, it is algebraically guaranteed that $(x^2 + y^2)$ and $(x^4 + x^2y^2 + y^4)$ are co-prime.
$\therefore gcd(A, B) = g^4(x^2 - y^2) \times gcd(x^2 + y^2, g^2)$

Since this solution uses the smallest arguments for the `gcd` function, this is the fastest solution.

Tricky Examples

Sample Input	Sample Output
1 6 4	80
1 10 6	512

Problem C. Magnetic Bond Optimization

Problem Setter: Mohammad Ashraful Islam Shovon
Tester: Muhiminul Islam Osim, Raihat Zaman Neloy
Category: Adhoc, Greedy
Total Solved: 141
First to Solve: Code_monkeys!

We can easily calculate the bond strength contribution of each pair of two adjacent particles. For two adjacent particles with opposite polarities, we get the minimum of the absolute values of their strengths as the bond strength and for two adjacent particles with the same polarity, we can consider 0 as the bond strength between them.

Suppose, the array of the ordered bond strengths is B. Now, if we get the prefix sum array C ($C_i = C_{i-1} + B_i$) from the array B, we can easily get the sum of values within a range of C with $\mathcal{O}(1)$ time complexity. For a range $[l, r]$, the sum of the values is actually $C_r - C_{l-1}$. By using the technique, for each query, we traverse through the array C, and get the sum of values for each possible range of length $K - 1$ and get the maximum as the optimal answer for that query.

Time complexity: $\mathcal{O}(T \cdot Q \cdot N)$

Problem D. Daripalla

Problem Setter: Sabbir Rahman
Alter: Irfanur Rahman Rafio
Category: Ad-hoc, STL, Math
Total Solved: 0
First to Solve: N/A

Note that all combinations are just linear sum of batkhara weights using coefficients $-1, 0, 1$. Input has the positive sums. So add the negatives of them to get all positive and negative possible target weights. We still do not know number of target weights which are 0. But we'll handle that later. Now take the maximum input, which is sum of the batkhara weights. Add this to all the batkharas. The multiset has now turned to all sums using coefficients $0, 1, 2$.

Now start with a known batkhara set which is empty and take the weights from the target weights ascending. Once you pick a weight add it to known set and erase all possible combinations of known batkhara set (using $0, 1, 2$) from the target weights. Continue and you'll get the full original weights.

Now for the unknown count of 0 issue. Note that after the offset adding, 0 has turned to half of sum of original batkhara weights. Unless there is only 1 batkhara, this value will never be a batkhara weight itself. So we can assume it as missing from the multiset always and ignore. And handle single input case separately. We could also calculate the lowest larger or equal power of three from the target set and deduce that we need to add the value until target set size is power of three.

For implementation, one can use stl sets. But they are a bit slow. Quicker solution is to use priority queue to

maintain which weights to delete. Both will pass. But the set solution might be too strict.

Problem E. Escape Plan II

Problem Setter: Arnob Sarker
Tester: Muhiminul Islam Osim, Sharif Minhazul Islam, Hasinur Rahman
Alternate writer: Irfanur Rahman Rafio
Category: DP, Tree, Binary Search
Total Solved: 17
First to Solve: DU_Primordius

The first part of the problem is to calculate the minimum amount of initial fuel needed for each answer, i.e., the minimum amount of fuel needed to escape using stops $[0, N-2]$. This can be solved using DP; for each node, calculate and store the minimum amount of fuel needed to escape using stops $[0, N-2]$.
Complexity: $\mathcal{O}(N^2)$

The rest is easy; use binary search over the answer.
Complexity: $\mathcal{O}(Q \log N)$

Problem F. Flowers II

Problem Setter: Md Hasinur Rahman
Tester: Md Nafis Sadique
Category: Hashing, Divide and Conquer, Heavy-Light Trick, Square Root Decomposition
Total Solved: 0
First to Solve: N/A

We can represent the unique shape of a Flower by representing the preorder traversal as LRU pathstring. LRU pathstring defines the tour as below.

- L: Left child of the current node.
- R: Right child of the current node.
- U: Up from the current node.

For example, the LRU pathstring of preorder traversal of the left flower in the figure below is LLURUU while pathstring of right flower is RRULUU. These LRU pathstrings will be unique for every unique flower shape.

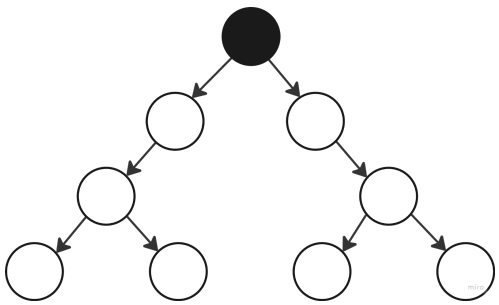

Figure 1: Flowers

If we can find the hash of the LRU pathstring of the preorder traversal of the flower, we can easily find the number of unique flowers in the tree. After that merging two trees will need merging two unique sets of hashes.

It can be done by always merging the light sets to the heavy sets which will reduce the complexity of overall merging complexity to amortized $O(N)$.

But we have a special case for query of type 1. While merging two trees, it can lead to merging two flowers. The main challenge here is to find the hash of the merged flower efficiently. Let's say the pathstring for the flower at top has pathstring S_T and bottom is S_B. We are connecting R_X as a child of Y. We have to figure out a position in the string S_T where we can insert the pathstring of R_X to get the pathstring of the merged flower. Let's say the position is i and we can write $S_T = S_T[1...i-1] + S_T[i, ..., |S|]$. Merging the flower will result in a pathstring of $S_{merged} = S_T[1...i-1] + S_B + S_T[i, |S|]$. But it requires $O(N)$ for each query. Can we reduce it? The answer is yes.

We imagine S_T as an array of blocks of size $\sqrt{N}$. The target position i will exist in exactly one block. Inserting S_B in this block will require $O(\sqrt{N})$. But inserting pathstrings in a block may make the blocks fat and lead to $O(N)$ merging complexity. That's where we will introduce a mechanism for keeping the element size of a block no more than $\sqrt{N}$. We define Entropy as total number of single character insertions on all the blocks of pathstring of a flower root. Initially $Entropy = 0$. If $Entropy \geq \sqrt{N}$, then we reconstruct all the blocks by distributing the characters of the pathstring, ensuring that each block contains at most $\sqrt{N}$ characters. As this balancing requires $O(N)$, but it happens only after Entropy becomes greater than $\sqrt{N}$, the balancing of $O(N)$ will be done at most $\sqrt{N}$ times. After modifying each block, we will recalculate the hashes for that block. We can find the hash of the merged flower from the hashes of each block. Still we have a challenge remaining. If we always merge pathstring of bottom flower to the top, it will lead to $O(N^2)$ time complexity overall. This problem can be solved by always merging the lightweight flower to the heavyweight flower.

Time Complexity: $\mathcal{O}(N\sqrt{N})$
Space Complexity: $\mathcal{O}(N\sqrt{N})$

Problem G. Glitch in the Multiverse

Problem Setter: Muhiminul Islam Osim
Tester: Nayeemul Islam Swad, Md Sabbir Rahman
Category: Data Structures
Total Solved: 0
First to Solve: N/A

In this type of problem, one way to handle query ranges is to apply the Mo's algorithm. In this way, while traversing through the ranges, our task is to add or remove the contribution of a particular version's occurrence to a data structure so that we can query for the current considering time s. An easy way to do that is maintaining some binary indexed trees (BIT) so that for a value of s of a query, we can get the total contribution of a particular meeting scenario from some of these BITs.

Now, how can we calculate the contribution of a version for a query? There are three types of scenarios to meet with another version.

1. **Walking in the same direction**: In this way, a version will meet another version if and only if they start at the same cell. We can just maintain the frequency of occurrence at each grid to calculate the contribution.
2. **Walking in the opposite direction on the same row or column**: As two versions move towards each other, for time s, a version can meet or pass another version if the distance between them doesn't exceed $min(2 \cdot s, n)$. So, for the direction of version i of a query, we need to get the summation of occurrences

of all versions with the opposite direction within the distance limit. We can maintain the contribution of the occurrences in BITs for four types of move (column-up, column-down, row-left, row-right).

3. **Walking in the perpendicular direction**: If a version walks on a row, they can meet another version walking on a column and vice versa. The potential starting cells of versions which a query version i can meet are the cells of the two diagonals (for two different perpendicular directions) going through the starting cell of the query version i. So, for each direction, we need to maintain the occurrence of the versions in two BITs for two different diagonals.

Finally, for each query, we get the total contribution from the 2D frequency array, and the range queries of some BITs according to the position and direction of the query version i.

The overall time complexity of the solution is $\mathcal{O}(T \cdot (N+Q) \cdot sqrt(Q) \cdot \log n)$.

Bonus Problem: Solve it with $\mathcal{O}(T \cdot (N+Q) \cdot \log^2 n)$ or $\mathcal{O}(T \cdot (N+Q) \cdot \log n)$.

Problem H. Array Apocalypse

Problem Setter: Arnob Sarker
Tester: Md Sabbir Rahman
Alternate Writer: Rumman Mahmud, Irfanur Rahman Rafio
Category: Probability, Number theory
Total Solved: 0
First to Solve: N/A

The key idea of solving this problem is to calculate the probability of choosing one number into the set S. For example:

- For array $\{1, 2\}$: $P(1) = 1$ and $P(2) = \frac{2}{3}$, because 1 will be chosen for sure and 2 will be chosen if it's picked before 1(probability of picking 2 when 1 and 2 is present in array $= \frac{2}{1+2}$).
- For array $\{2, 3\}$: probability of choosing 2 and 3 is 1 because both will be chosen regardless of the order.

An element will be chosen if and only if, among all its factors, it is the first one to be chosen. Probability of choosing A_i:

$$P(A_i) = \frac{A_i \times cnt(A_i)}{\sum_{d|A_i} d \times cnt(d)}$$

GPS calculation:

$$GPS(N) = GPS(N-1) + 2 \times \sum_{d|N} phi(d) \times \frac{N}{d} - N$$

So, the expected value would be:

$$E(A) = \sum_{i=1}^{n} P(A_i) \times GPS(A_i)$$

Complexity: $\mathcal{O}(N \log N)$

Problem I. Let's Learn Multiplication Table

Problem Setter: Raihat Zaman Neloy
Tester: Rumman Mahmud
Alt Writer: Irfanur Rahman Rafio
Category: Ad-hoc
Total Solved: 152
First to Solve: -... .-. .- -.-. ..- / ..- .– ..-

This problem can be solved in various ways. A few of the solutions are:
- You can try to find the equation which is written like: $1 \times B = C$ or $A \times 1 = C$. In that case, the answer will be C.
- You can keep a counter of all the numbers that occurred in A or B. This way, the number that occurred more than 9 times will be the answer.
- You can keep an array of all the values of C, sort the array and print the first value.
- You can find the gcd of all the C_i and the gcd will be the answer.

Problem J. T5

Problem Setter: Irfanur Rahman Rafio
Tester: Sharif Minhazul Islam
Alt Writer: Arnob Sarker
Category: Game Theory, Implementation
Total Solved: 33
First to Solve: Code_monkeys!

Let's first analyze one board and try to understand what's happening.

Observations

- If the first move is made on the center, no second move can be made on this board without allowing the opponent to win on the third move.
- If the first move is made on a corner or an edge, a second move can be made on this board without allowing the opponent to win on the third move. However, this is only possible if the first two moves are placed in cells that are one knight-move apart. A knight-move refers to the movement of a knight in chess: from its current position, it can move to any cell that is two steps in one direction and one step perpendicular to it. If this criterion is not met, a cell will exist where the third move can result in a win..
- If more than two moves have been made and the game is not yet decided, there must exist a cell where making the next move will result in a win.

States

Any board can be classified into one of the following states:

1. **Solved**: The game on this board is already decided.
2. **Open**: The game can be won with the next move.
3. **Blocked**: Any move on this board will make it `Open` for the opponent to win on their subsequent turn.
4. **Half-Blocked**: Any move on this board will either make it `Blocked` or `Open`.
5. **Empty**: No move has been made on this board yet.

A board can enter a `Blocked` state in two ways:

1. The first move was made on the center cell.
2. Two moves have been made on this board, and these moves are one knight-move apart.

A board can enter a `Half-Blocked` state when a single move is made, and that move is either on a corner or an edge.

From a `Half-Blocked` state, making a second move can transition the board to either a `Blocked` or `Open` state, depending on the placement of that move.

From an `Empty` state, the board can only enter either a `Blocked` state or a `Half-Blocked` state.

Solution

The constraint of the problem guarantees that none of the boards will be in `Solved` state.

If at least one board is in `Open` state, the player to make the next move can win immediately. This case can be handled separately. Now let's work with the cases where none of the boards are in `Open` state.

Each board in the game is either `Empty`, `Half-Blocked`, or `Blocked`. In an optimal play, when a player receives a state where all boards are `Blocked`, he will lose the game.

Since `Blocked` boards cannot be touched, let's think about the `Empty` boards and the `Half-Blocked` boards. The available transitions are:

- Empty to Blocked
- Empty to Half-Blocked
- Half-Blocked to Blocked

Now, if a player receives a state where the number of `Empty` boards and the number of `Half-Blocked` boards are both even, he can be defeated using an optimal strategy. Whatever move he makes, his opponent needs to mirror that transition on a different board. Eventually, the number of `Empty` boards and `Half-Blocked` boards will go down to 0, and he has to make a `Blocked` board `Open`. So a state with an even number of `Empty` boards and an even number of `Half-Blocked` boards is a losing state.

A state with even `Empty` boards and odd `Half-Blocked` boards is a winning state. Because from here, the next player can make a `Half-Blocked` board `Blocked`. Now the number of `Empty` boards and the number of `Half-Blocked` boards are both even, so the opponent needs to play from a losing state.

A state with odd `Empty` boards and even `Half-Blocked` boards is a winning state. Because from here, the next player can make an `Empty` board `Blocked`. Now the number of `Empty` boards and the number of `Half-Blocked` boards are both even, so the opponent needs to play from a losing state.

A state with odd `Empty` boards and odd `Half-Blocked` boards is also a winning state. Because from here, the next player can make an `Empty` board `Half-Blocked`. Now the number of `Empty` boards and the number of `Half-Blocked` boards are both even, so the opponent needs to play from a losing state.

Alternate Solution

Handle the case where `Open` board(s) exist separately.

Since both players are using the same symbol, this is a combinatorial game. Now, consider the `Empty`, `Half-Blocked` and `Blocked` boards as nim piles with 2, 1 and 0 stones respectively. Here, the size of nim pile represents the maximum number of moves that can be played on that board without blocking it. Solve the game of nim by taking the xor of the nim values. If the current state is a winning state, the next player to move wins; otherwise he loses.

Problem K. Total Distance Over All Trees

Problem Setter: Nayeemul Islam Swad
Tester: Raihat Zaman Neloy, Md Nafis Sadique
Alter: Sabbir Rahman
Category: Combinatorics, Contribution technique, Prüfer sequence
Total Solved: 4
First to Solve: BRACU_Randomized_Approximation_Algorithm

$\mathcal{O}(N \log N)$ Solution

We use the contribution technique: Suppose we're given a tree where $d(1, n) = k$. If for each of those k edges we add 1 to our final answer, then in the end this tree will have contributed k to the final answer. So instead of fixing a tree and finding out $d(1, n)$ for it, we fix an edge (u, v) and find out $f(u, v) =$ the number of trees where this edge contributes to $d(1, n)$. Then the sum of $f(u, v)$ over all possible edges (u, v) is our final answer to the problem.

Suppose that an edge (u, v) has k nodes at one side of it (the u side) and $n - k$ nodes at the other side (the v side). Since the edge must lie on the path from nodes 1 to n, the nodes 1 and n must belong to different sides. Let's assume that 1 lies on the side with k nodes and n lies on the side with $n - k$ nodes.

Let's fix k and find the number of trees such that there is an edge (u, v) with k nodes at the u-subtree, $n - k$ nodes at the v-subtree, and 1 and n lie on the u and v-subtrees, respectively.

For the u-subtree: there are k nodes in total, one of which is 1. The remaining $k - 1$ nodes can be chosen in $\binom{n-2}{k-1}$ ways (node n cannot be here) and u can be chosen in k ways from this subtree. From Cayley's formula, the number of rooted (at node u) non-labelled trees of k nodes is k^{k-2}.

For the v-subtree: there are $n - k$ nodes in total, all remaining after constructing the u-subtree. Among these, v can be chosen in $n - k$ ways. From Cayley's formula, the number of rooted (at node v) non-labelled trees of $n - k$ nodes is $(n - k)^{n-k-2}$.

So for a fixed k, we should add $\binom{n-1}{k-1} \cdot k \cdot k^{k-2} \cdot (n - k) \cdot (n - k)^{n-k-2} = \binom{n-1}{k-1} \cdot k^{k-1} \cdot (n - k)^{n-k-1}$ to our final answer. Iterating over $k = 1, \ldots, n - 1$ and adding the corresponding values we get the final answer.

Note that we need to precompute the factorials, inverse factorials, and the values of the form i^{i-1}.

$\mathcal{O}(N)$ Solution

This is the intended solution.

Note that the answer stays the same if we sum up the values for $d(n - 1, n)$ instead of $d(1, n)$.

Consider the Prüfer sequence of the given tree — $p_1, p_2, \ldots, p_{n-2}$. Here, p_1 is the parent of the *smallest* leaf of the whole tree. Just to make the explanation simpler, let's assume that $p_{n-1} = n$ since n would always be the parent of the last removed leaf.

Now, imagine the process of reconstructing the tree from this Prüfer sequence. Since all the nodes except n gets removed during the reconstruction process, $n - 1$ must also be removed at some point and thus it would be the smallest leaf at that time. Define i as the position in the sequence where node $n - 1$ gets removed (i.e. p_i is the parent of $n - 1$ when $(n - 1)$ is being removed).

Observation: $[(n - 1), p_i, p_{i+1}, \ldots, p_{n-2}, p_{n-1} = n]$ is actually the path from the node $n - 1$ to the node n. So, the distance between node $n - 1$ and node n is $n - i$.

So given a Prüfer sequence for a tree, if we know how to find i quickly from the sequence then we can immediately calculate $d(n - 1, n) = n - i$ for the corresponding tree. For this, we need to observe two properties that i satisfies.

Property 1: All the integers $p_i, p_{i+1}, \ldots, p_{n-1}$ are pairwise distinct.
Proof: This is true, because we just noticed earlier that $[(n-1), p_i, p_{i+1}, ..., p_{n-2}, p_{n-1} = n]$ is the path from node $n-1$ to node n. So the nodes in the path must be distinct.

Property 2: $n-1$ does not occur among $p_i, p_{i+1}, ..., p_{n-1}$.
Proof: Again, this is true because $[(n-1), p_i, p_{i+1}, ..., p_{n-2}, p_{n-1} = n]$ is the path from node $n-1$ to node n. So $n-1$ cannot occur twice in the path.

Observation: i is actually the smallest possible index in the Prüfer sequence that satisfies both of these properties.
Proof: Suppose the contrary, then $i-1$ also satisfies both of the above properties. In that case, during the reconstruction process of the tree from the Prüfer sequence, once we remove a leaf for the term $i-1$ the node p_{i-1} becomes a leaf (according to property 1). Also, by property 2, $p_{i-1} < n-1$. Thus, $n-1$ cannot be the smallest leaf at p_i, because p_{i-1} is also a leaf and it is smaller. Thus, contradiction.

Now armed with these observations, suppose we're given a tree and a non-negative integer k. Let $f(k)$ be the number of trees that satisfy $d(n-1, n) \geq k$. Then notice that, $d(n-1, n) \geq k$ if and only if the last k terms in the Prüfer sequence (including $p_{n-1} = n$) are all pairwise distinct and none of them are equal to $n-1$. So, $f(k) = \binom{n-2}{k-1} \cdot (k-1)! \cdot n^{n-1-k}$. By precomputing all the factorials, inverse factorials, and powers of n, we can find the answer in $\mathcal{O}(N)$ by adding up $f(k)$ for $k = 1, \ldots, n-1$.