Problem Analysis

The 2025 ICPC Asia Jakarta Regional Contest

November 24, 2025

Problem A

A. Nihilation

Total solves:	48
First solver:	std_abs, National Taiwan University (0:05)
Author:	Albert Yulius Ramahalim

Problem A: A. Nihilation

Problem

Given an array A, find the minimum number of operations $A_i := (A_i \times k) \bmod m$ to make all $A_i = 0$.

Problem A: A. Nihilation

Problem

Given an array A, find the minimum number of operations $A_i := (A_i \times k) \bmod m$ to make all $A_i = 0$.

- The upper bound is 2 operations: $A_i := 2A_i \bmod 4$ and $A_i := A_i \bmod 2$.
- We can only use 1 operation when $\gcd(A_i) > 1$, say G. In this case, we can just do $A_i := A_i \bmod G$.

Problem B

Burning Blocks

Total solves:	42
First solver:	GMAC, Universitas Gadjah Mada (0:17)
Author:	William Justin

Problem B: Burning Blocks

Problem

There are N stacks of $W_1, \ldots, W_N$ identical blocks from left to right.

Initially, the outermost blocks are burned. Each block takes one minute to burn, after which the fire will spread to all adjacent blocks.

How long does it take to burn all blocks?

Problem B: Burning Blocks

Problem

There are N stacks of $W_1, \ldots, W_N$ identical blocks from left to right.

Initially, the outermost blocks are burned. Each block takes one minute to burn, after which the fire will spread to all adjacent blocks.

How long does it take to burn all blocks?

Use DP. Define:

- $L(i)$: time required to burn columns $1..i$ if fire spreads from the **left** and top.
- $R(i)$: time required to burn columns $i..N$ if fire spreads from the **right** and top.

DP Transitions

The transitions are:

- $L(i) = \min(L(i-1), W_i)$
- $R(i) = \min(R(i+1), W_i)$

The answer is the maximum value of $\min(L(i), R(i))$ across all i.

Time complexity: $O(N)$.

Problem J

AND and/or OR

Total solves:	36
First solver:	Selamat! Berjuang! Sukses! *handclap, Universitas Indonesia (0:24)
Author:	Pikatan Arya Bramajati

Problem J: AND and/or OR

Problem

You are given three integers N, A, and B. You may do the following operations on N:

1. $N := N \text{ AND } 2N$
2. $N := N \text{ OR } 2N$

Is there a sequence of A type 1 and B type 2 operations such that you end up with $N \cdot 2^k$ for some integer k?

Binary Segments

- Write N in binary, then view the consecutive 0s and 1s as 'segments'. If we ignore trailing zeros, ending with $N \cdot 2^k$ is equivalent to ending with the same lengths of segments.

Binary Segments

- Write N in binary, then view the consecutive 0s and 1s as 'segments'. If we ignore trailing zeros, ending with $N \cdot 2^k$ is equivalent to ending with the same lengths of segments.
- A type 1 operation will shorten each 0 segment and lengthen each 1 segment by 1. A type 2 operation will do the opposite.

Binary Segments

- Write N in binary, then view the consecutive 0s and 1s as 'segments'. If we ignore trailing zeros, ending with $N \cdot 2^k$ is equivalent to ending with the same lengths of segments.
- A type 1 operation will shorten each 0 segment and lengthen each 1 segment by 1. A type 2 operation will do the opposite.
- So, to make sure that the segments will be equal, we need that $A = B$, and there are no segments of length 1 (so it doesn't get removed).

Binary Segments

- Write N in binary, then view the consecutive 0s and 1s as 'segments'. If we ignore trailing zeros, ending with $N \cdot 2^k$ is equivalent to ending with the same lengths of segments.
- A type 1 operation will shorten each 0 segment and lengthen each 1 segment by 1. A type 2 operation will do the opposite.
- So, to make sure that the segments will be equal, we need that $A = B$, and there are no segments of length 1 (so it doesn't get removed).

Time complexity: $O(\log N)$.

Problem M

Predisposed

Total solves:	31
First solver:	193637, National University of Singapore (0:09)
Author:	Albert Yulius Ramahalim

Problem M: Predisposed

Problem

A sequence $A_1, \ldots, A_N$ is **predisposed** if $0 \leq A_i < M$ and for all constraints (X_j, Y_j), we have that $\sum_{X_j | k} A_k \equiv Y_j \pmod{M}$.

Count the number of predisposed sequences.

Problem M: Predisposed

Problem

A sequence $A_1, \ldots, A_N$ is **predisposed** if $0 \leq A_i < M$ and for all constraints (X_j, Y_j), we have that $\sum_{X_j | k} A_k \equiv Y_j \pmod{M}$.

Count the number of predisposed sequences.

- Construct A from index N to 1. For index i:
  - If there is a constraint (i, y), then there is only one possible value for A_i: $y - (A_{2i} + A_{3i} + \ldots) \bmod M$.
  - If there is no such constraint, there are M possible values for A_i.
- Thus, the answer is M^{N-Q}.

Problem K

Travelling Taro Trains

Total solves:	19
First solver:	std_abs, National Taiwan University (0:49)
Author:	Hibiki Nishiwaki

Problem K: Travelling Taro Trains

Problem

You are given an initially empty directed graph with N vertices. Edges will be in the form of (u, v, k): from u to v with label k. Initially, you are at vertex 1. Process Q queries:

1. Add an edge (u, v, k).
2. Remove the edge (u, v, k).
3. You may choose to stay at your current vertex, or move to an adjacent vertex via an edge with label k.

For each type-3 query, output the number of different vertices you can possibly be in.

Greedy

We only need to examine each edge (u, v, k) at most once, i.e. at the earliest possible timing when u is visitable and there is a type-3 query with that value of k.

Greedy

We only need to examine each edge (u, v, k) at most once, i.e. at the earliest possible timing when u is visitable and there is a type-3 query with that value of k.

Thus, we can maintain:

- A_i: the set of edges with label i whose starting point is a **visitable** vertex.
- B_j: the set of **unexamined** edges outgoing from an unvisited vertex j.

Greedy

We only need to examine each edge (u, v, k) at most once, i.e. at the earliest possible timing when u is visitable and there is a type-3 query with that value of k.

Thus, we can maintain:

- A_i: the set of edges with label i whose starting point is a **visitable** vertex.
- B_j: the set of **unexamined** edges outgoing from an unvisited vertex j.

Then, for each query of type 3, we can examine all the edges in A_k. If we end up being able to visit a new vertex t, insert the edges from B_t to the appropriate A's.

Time complexity: $O(M + Q)$ or $O((M + Q) \log(M + Q))$.

Problem C

Binary Grid

Total solves:	5
First solver:	193637, National University of Singapore (1:29)
Author:	Maximilliano Utomo Quok

Problem C: Binary Grid

Problem

Construct a binary grid of size $N \times M$ such that:

- there are no adjacent cells with value 1,
- there are exactly A_i cells with value 1 in row i.

Problem C: Binary Grid

Problem

Construct a binary grid of size $N \times M$ such that:

- there are no adjacent cells with value 1,
- there are exactly A_i cells with value 1 in row i.

Let's handle some easy cases:

- All A_i must be $\leq \lceil M/2 \rceil$.
- If M is even, we can just make a checkerboard pattern and remove 1s as needed.

Odd M Case

If M is odd, note that for each row where $A_i = (M+1)/2$, there is only one way to configure that row.

Let's focus on the way to configure the gaps between two rows with $A_i = (M+1)/2$.

Odd M Case

If M is odd, note that for each row where $A_i = (M+1)/2$, there is only one way to configure that row.

Let's focus on the way to configure the gaps between two rows with $A_i = (M+1)/2$.

- If the number of rows in the gap is odd, we can use the checkerboard pattern like when M is even.
- If the number of rows in the gap is even, we can fill it greedily, then omit the 1s as needed. The result is the grid on the right (`.` $= 0$, `X` $= 1$).

```
X.X.X.X
.X.X.X.
..X.X.X
X..X.X.
.X..X.X
X.X..X.
.X.X..X
X.X.X..
.X.X.X.
X.X.X.X
```

Problem D

Count DFS Graph

Total solves:	5
First solver:	193637, National University of Singapore (1:40)
Author:	Muhammad Ayaz Dzulfikar

Problem D: Count DFS Graph

Problem

Given a permutation P of length N, where $P_1 = 1$.

Count the number of graphs such that when doing DFS starting from vertex 1 and visiting unvisited neighbors in ascending indices, the list of vertices visited in order is equal to P.

Problem D: Count DFS Graph

Problem

Given a permutation P of length N, where $P_1 = 1$.

Count the number of graphs such that when doing DFS starting from vertex 1 and visiting unvisited neighbors in ascending indices, the list of vertices visited in order is equal to P.

Suppose that we are also given the DFS tree. How many graphs are there such that the DFS will give us that tree?

Problem D: Count DFS Graph

Problem

Given a permutation P of length N, where $P_1 = 1$.

Count the number of graphs such that when doing DFS starting from vertex 1 and visiting unvisited neighbors in ascending indices, the list of vertices visited in order is equal to P.

Suppose that we are also given the DFS tree. How many graphs are there such that the DFS will give us that tree?

Observation

For each $2 \leq i \leq N$, let $S_i := \{j : j \text{ descendant of } i, j > i\}$ be the vertices that may have an edge with i's parent. Then, there are $\prod_{i=2}^{N} 2^{|S_i|}$ ways to add back edges.

Counting over DFS Trees

Then, do DP to count over all possible DFS trees that produce P.

- $\mathrm{dp}(l, r) =$ number of DFS trees from $A[l..r]$ rooted at some vertex $u < l$.

Counting over DFS Trees

Then, do DP to count over all possible DFS trees that produce P.

- $\mathrm{dp}(l, r) =$ number of DFS trees from $A[l..r]$ rooted at some vertex $u < l$.
- Transitions:
  - Split into siblings: $\mathrm{dp}(l, r) = \sum_t \left(\mathrm{dp}(l+1, t) \cdot \mathrm{dp}(t+1, r) \cdot 2^{|S_l|}\right)$.
  - Don't split: $\mathrm{dp}(l, r) = \mathrm{dp}(l+1, r) \cdot 2^{|S_l|}$.

  Here, each $|S_l|$ considers the subtree in the appropriate context.
- Answer: $\mathrm{dp}(2, N)$.

Time complexity: $O(N^3)$.

Problem F

Two Sets

Total solves:	4
First solver:	std_abs, National Taiwan University (1:13)
Author:	Maximilliano Utomo Quok

Problem F: Two Sets

Problem

Given an undirected graph $G = (V, E)$ with N vertices and M edges. Find two integers p, q and two sets S_1, S_2 such that:

- $N < (p+1)(q+1)$
- For every $u \in S_1$, there are at least p vertices in S_1 such that $(u, p) \in E$.
- $|S_2| \geq q$ and S_2 is an independent set.

Selecting p

Let's select p as follows:

1. Start from $S_1 = \{1, 2, \ldots, N\}$.
2. Repeatedly remove a vertex $v \in S_i$ with minimum degree from S_1. Remove from the graph all edges and nodes incident to v.
3. Pick a step of S_1 that maximizes the value of p.

Next, we want an independent set of $q \geq \lfloor N/(p+1) \rfloor$.

Constructing the Independent Set

Let's do the following:

1. Start from the initial graph and $S_2 = \{\}$.
2. Repeatedly add $v \notin S_2$ with the minimum degree to S_2. Remove from the graph all edges and nodes incident to v.

Note that the order of picking v's while constructing S_2 is the same with while constructing S_1. So, each step removes at most p vertices, yielding $\lceil n/(p+1) \rceil$ iterations.

Time complexity: $O(N^2)$.

Problem G

Grid Game 2×2

Total solves:	3
First solver:	std_abs, National Taiwan University (2:02)
Author:	Prabowo Djonatan

Problem G: Grid Game 2×2

Problem

There is a $10^9 \times 10^9$ grid. N cells are black and the rest are white. Two players will play a game, with the following happening each turn.

1. Choose a black cell (r, c).
2. Pick a (possibly empty) subset $K \subseteq \{1, ..., t\}$ where $t = \lfloor \log_2 \min(r, c) \rfloor$.
3. For each $k \in K$, toggle the colors of all $(r - i, c - j)$ such that $0 \leq i, j < 2^k$ and $(i, j) \neq (0, 0)$.
4. Toggle the color of (r, c).

The player who cannot move loses. Find the winner.

Magic

Group all the cells. Group k contains all (x, y) such that $x \operatorname{OR} y$ has k trailing zeroes.

Solution

First player wins iff there is a group with an odd number of black cells.

Magic

Group all the cells. Group k contains all (x, y) such that $x \text{ OR } y$ has k trailing zeroes.

Solution

First player wins iff there is a group with an odd number of black cells.

Proof. Say (x, y) satisfies $x \equiv y \equiv 0 \pmod{2^k}$ (i.e. it is in group $\geq k$). So, by moving with the set $K = \{k\}$, the parity of the number of black cells in group $k - 1$ is changed.

Proof (Cont.)

- If there is a group with an odd number of black cells, choose the largest numbered among such groups, and choose

$$K = \{k : \text{group } (k-1) \text{ has an even number of black cells}\}.$$

- If all groups have an even number of black cells, any move will result in a group having an odd number of black cells.

Proof (Cont.)

- If there is a group with an odd number of black cells, choose the largest numbered among such groups, and choose

$$K = \{k : \text{group } (k-1) \text{ has an even number of black cells}\}.$$

- If all groups have an even number of black cells, any move will result in a group having an odd number of black cells.

Thus, states where there is a group with an odd number of black cells are winning states. ■

Time complexity: $O(N)$.

Problem I

International Olympiad in ICPC

Total solves:	3
First solver:	NYCU_CartesianTreeII, National Yang Ming Chiao Tung University (1:52)
Author:	Ashar Fuadi

Problem I: International Olympiad in ICPC

Problem

Given a grid of characters `.` and `#` of size $3 \times N$. Cells with `#` are blocked, while cells with `.` are white. You want to print the word IOI by painting some cells black, with the following rules:

- The first letter `I` is a rectangle of size $3 \times p$ ($p \geq 1$).
- The letter `O` is the boundary of a rectangle of size $3 \times q$ ($q \geq \max(3, p + r)$).
- The second letter `I` is a rectangle of size $3 \times r$ ($r \geq 1$).
- The `O` is between the two `I`'s and there must be at least one column between the letters.

What is the maximum number of cells you can paint black?

Main Observation

Any solution can be split into two groups. Each group has an `I` and a "half-`O`":

```
           |
.III..OOOO | OO.II.
.III..O... | .O.II.
.III..OOOO | OO.II.
           |
```

So, we can solve for each part separately.

Main Observation

Any solution can be split into two groups. Each group has an `I` and a "half-`O`":

```
           |
.III..OOOO | OO.II.
.III..O... | .O.II.
.III..OOOO | OO.II.
           |
```

So, we can solve for each part separately.

Let $L(i)$ be the maximum number of painted cells if column i is the rightmost column of the half-`O` (i.e. considering all columns $\leq i$). Define $R(i)$ similarly for the columns $\geq i$. Our answer will be $\max(L(i) + R(i+1))$.

Solving a Half-Problem

Classify the columns into three types:

- Type 0: the entire column is unblocked.
- Type 1: only the top and bottom cells are unblocked (it cannot be part of an `I` or the edge of an `O`).
- Type 2: all other columns (no letter can include these columns).

Solving a Half-Problem

Classify the columns into three types:

- Type 0: the entire column is unblocked.
- Type 1: only the top and bottom cells are unblocked (it cannot be part of an `I` or the edge of an `O`).
- Type 2: all other columns (no letter can include these columns).

Let $W(i)$ be the largest width of an `I` to the left of column i, if column i is the leftmost column of the half-`O`.

$W(i)$ can be calculated in $O(N)$ with prefix maximums.

Sweeping to Calculate $L(i)$

Next, define S_i as the set of all columns j such that j is a type-0 column, and columns j to i can be part of a half-0.

Sweeping to Calculate $L(i)$

Next, define S_i as the set of all columns j such that j is a type-0 column, and columns j to i can be part of a half-0.

Thus,

$$L(i) = \max_{j \in S_i} \left(\underbrace{(3\min(W_j, i-j+1))}_{\text{from I}} + \underbrace{(2(i-j+1)+1)}_{\text{from half-0}} \right).$$

Sweeping to Calculate $L(i)$

Next, define S_i as the set of all columns j such that j is a type-0 column, and columns j to i can be part of a half-0.

Thus,

$$L(i) = \max_{j \in S_i} \left(\underbrace{(3 \min(W_j, i - j + 1))}_{\text{from I}} + \underbrace{(2(i - j + 1) + 1)}_{\text{from half-0}} \right).$$

We can sweep i from left to right and maintain S_i easily:

- If we meet a type-0 column, add it to the current set.
- If we meet a type-2 column, clear the set.

Sweeping to Calculate $L(i)$

Next, define S_i as the set of all columns j such that j is a type-0 column, and columns j to i can be part of a half-0.

Thus,

$$L(i) = \max_{j \in S_i} \left(\underbrace{(3 \min(W_j, i - j + 1))}_{\text{from I}} + \underbrace{(2(i - j + 1) + 1)}_{\text{from half-0}} \right).$$

We can sweep i from left to right and maintain S_i easily:

- If we meet a type-0 column, add it to the current set.
- If we meet a type-2 column, clear the set.

Next, we just need a way to maintain the inner expression across all values of $j \in S_i$.

Maintaining the Inner Expression

Note that the inner expression has two possible values depending on whether $(i - j + 1)$ is less than W_j or not.

Maintaining the Inner Expression

Note that the inner expression has two possible values depending on whether $(i - j + 1)$ is less than W_j or not.

So, we can maintain two sets, one for each possibility. As we sweep from left to right, an index j will initially have $(i - j + 1) \leq W_j$.

We can keep track on which value of i the sign changes, and we move the contribution of an index j from one set to the other.

Maintaining the Inner Expression

Note that the inner expression has two possible values depending on whether $(i - j + 1)$ is less than W_j or not.

So, we can maintain two sets, one for each possibility. As we sweep from left to right, an index j will initially have $(i - j + 1) \leq W_j$.

We can keep track on which value of i the sign changes, and we move the contribution of an index j from one set to the other.

Don't forget to handle some special cases, such as when the sets need to be cleared (because of a type-2 column), or when the total width of the `O` is only 2 (in which case the answer is -1).

Time complexity: $O(N \log N)$.

Problem H

Maximeter

Total solves:	2
First solver:	193637, National University of Singapore (2:18)
Author:	Pikatan Arya Bramajati

Problem H: Maximeter

Problem

Given two integers M and D. Find the maximum number of vertices in a rooted weighted tree satisfying:

- All weights are positive integers.
- For each vertex x and integer w, there is at most M children c of x such that the weight of the edge (c, x) is w.
- The diameter of the tree is at most D.

Solving the Easier Version

Define $f_M(d)$ as the maximum number of vertices in a rooted tree such that the distance between each node and the root is at most d, and each vertex has at most M children with the same weight.

Solving the Easier Version

Define $f_M(d)$ as the maximum number of vertices in a rooted tree such that the distance between each node and the root is at most d, and each vertex has at most M children with the same weight.

Then, we have $f_M(0) = 1$, and

$$f_M(d) = 1 + M \sum_{i=0}^{d-1} f_M(i) = (M+1)^d.$$

Back to the Problem

Let's solve if D is even:

Back to the Problem

Let's solve if D is even:

- Start from a center c, build $f_M(D/2)$ from it.

Back to the Problem

Let's solve if D is even:

- Start from a center c, build $f_M(D/2)$ from it.
- Add $D/2$ edges of length 1 above c, and for each ancestor with distance k, build $f_M(D/2-k) - f_M(D/2-k-1)$ from it.

Back to the Problem

Let's solve if D is even:

- Start from a center c, build $f_M(D/2)$ from it.
- Add $D/2$ edges of length 1 above c, and for each ancestor with distance k, build $f_M(D/2-k)-f_M(D/2-k-1)$ from it.
- So, the total number of vertices is

$$\begin{aligned}&f_M\left(\frac{D}{2}\right)+f_M\left(\frac{D}{2}-1\right)-f_M\left(\frac{D}{2}-2\right)+f_M\left(\frac{D}{2}-2\right)-f_M\left(\frac{D}{2}-3\right)+\ldots\\=&f_M\left(\frac{D}{2}\right)+f_M\left(\frac{D}{2}-1\right).\end{aligned}$$

Back to the Problem

Let's solve if D is even:

- Start from a center c, build $f_M(D/2)$ from it.
- Add $D/2$ edges of length 1 above c, and for each ancestor with distance k, build $f_M(D/2-k) - f_M(D/2-k-1)$ from it.
- So, the total number of vertices is

$$\begin{aligned} & f_M\left(\frac{D}{2}\right) + f_M\left(\frac{D}{2}-1\right) - f_M\left(\frac{D}{2}-2\right) + f_M\left(\frac{D}{2}-2\right) - f_M\left(\frac{D}{2}-3\right) + \ldots \\ =& f_M\left(\frac{D}{2}\right) + f_M\left(\frac{D}{2}-1\right). \end{aligned}$$

We can solve similarly if D is odd. The answer is $2f_M((D-1)/2)$.

Time complexity: $O(\log D)$ per test case.

Problem L

Down Right Chips

Total solves:	1
First solver:	std_abs, National Taiwan University (3:36)
Author:	Pikatan Arya Bramajati

Problem L: Down Right Chips

Problem

You have an $N \times M$ grid. There are $A_{i,j}$ chips in cell (i, j). You can do two kinds of moves:

1. Choose a tile (r, c) not in the bottommost row. Remove 2 chips from (r, c) and add 1 chip to $(r + 1, c)$.
2. Choose a tile (r, c) not in the rightmost row. Remove 1 chip from (r, c) and add 1 chip to $(r, c + 1)$.

You want to do a number of moves until you can't do any more moves. What is the minimum number of moves you need to make?

There are Q updates. On each update, a number of chips will be added to a cell. Answer the question above after each update.

Greedy Protocol

Since a *down* move removes a chip, we want to do *down* moves ASAP.

So, greedily, we can iterate over all columns from left to right, do as many *down* moves as possible, then move the chips to the next column with *right* moves.

Segment Tree

Because of this, each row of each column will contribute at most one chip to the right. So (ignoring the bottommost row), there are 2^{N-1} ways this can happen.

Segment Tree

Because of this, each row of each column will contribute at most one chip to the right. So (ignoring the bottommost row), there are 2^{N-1} ways this can happen.

Thus, we can use a segment tree. For a range $[l, r]$, and for each 2^{N-1} ways to receive a chip from the column to the left, the segtree node will store:

- The number of moves needed to move all chips to the rightmost column.
- The final configuration of chips on the rightmost column.

The nodes can then be merged in $O(2^N)$.

Time complexity: $O(Q2^N \log M)$.

Problem E

Mugen E

Total solves:	1
First solver:	std_abs, National Taiwan University (3:12)
Author:	Albert Yulius Ramahalim

Problem E: Mugen E

Problem

Given K strings $S_1, \ldots, S_K$ and a string T.

Define $f(n)$ as E_n/n. Here, E_n the EV of number of occurrences of T in U as a substring, where U is formed by picking a string at random among S_i for n times and concatenating them.

Find the value of $\lim_{n \to \infty} f(n)$.

Calculating $f(n)$

Let's calculate $f(n)$ using contribution. For an array of indices A, define $t(A)$ as the number of occurrences of U in $S_{A_1} + \cdots + S_{A_{|A|}}$ which spans the entire A (that is, it won't get removed if A_1 or $A_{|A|}$ was erased).

Calculating $f(n)$

Let's calculate $f(n)$ using contribution. For an array of indices A, define $t(A)$ as the number of occurrences of U in $S_{A_1} + \cdots + S_{A_{|A|}}$ which spans the entire A (that is, it won't get removed if A_1 or $A_{|A|}$ was erased).

Then,

$$f(n) = \frac{E_n}{n} = \frac{1}{n} \sum_{|A| \le n} \left((n - |A| + 1) \left(\frac{1}{K} \right)^{|A|} \times t(A) \right)$$

Calculating $f(n)$

Let's calculate $f(n)$ using contribution. For an array of indices A, define $t(A)$ as the number of occurrences of U in $S_{A_1} + \cdots + S_{A_{|A|}}$ which spans the entire A (that is, it won't get removed if A_1 or $A_{|A|}$ was erased).

Then,

$$f(n) = \frac{E_n}{n} = \frac{1}{n} \sum_{|A| \le n} \left((n - |A| + 1) \left(\frac{1}{K} \right)^{|A|} \times t(A) \right)$$
$$= \sum_{x=1}^{n} \left(\frac{n - x + 1}{n} \left(\frac{1}{K} \right)^{x} \sum_{|A|=x} t(A) \right).$$

Taking the Limit

Thus,

$$\lim_{n\to\infty} f(n) = \sum_{x=1}^{\infty} \left(\left(\frac{1}{K}\right)^x \sum_{|A|=x} t(A) \right).$$

(The bound of the sum is actually $|T|$, not ∞.)

Taking the Limit

Thus,

$$\lim_{n\to\infty} f(n) = \sum_{x=1}^{\infty} \left(\left(\frac{1}{K}\right)^x \sum_{|A|=x} t(A) \right).$$

(The bound of the sum is actually $|T|$, not ∞.)

To count the sum of $t(A)$ over all A, we can use DP. Let $\mathrm{dp}(l, i)$ as the number of ways to concatenate l strings such that the resulting suffix coincides with the prefix of length i of T.

Taking the Limit

Thus,

$$\lim_{n\to\infty} f(n) = \sum_{x=1}^{\infty} \left(\left(\frac{1}{K}\right)^x \sum_{|A|=x} t(A) \right).$$

(The bound of the sum is actually $|T|$, not ∞.)

To count the sum of $t(A)$ over all A, we can use DP. Let $\mathrm{dp}(l, i)$ as the number of ways to concatenate l strings such that the resulting suffix coincides with the prefix of length i of T.

There are $O(|T|^2)$ states and $O(K)$ transitions, which can be precomputed with hashing/KMP/Z-function in $O(\sum |S_i| + K|T|)$.

Speeding Up the DP

To speed up the DP, notice that we can drop the first dimension of the DP, and instead multiply the previous DP value with $1/K$ every time we add a string to the concatenation. There are now $O(|T|)$ states.

Time complexity: $O(\sum |S_i| + K|T|)$.

Thank you!

We hope you enjoyed the contest!