Problem A. Quantum Corridor

Author: Mohidul Haque Mridul
Alter: Nayeemul Islam Swad, Arman Ferdous
Tester: Ruhan Habib
Solve Count vs Expected Solve Count: 0 / 3
First Solve: N/A

Abridged Statement

You are given a linear corridor with n chambers indexed 1 to n. A particle starts at chamber i. From any chamber x, the particle can move to $x \pm a$ or $x \pm b$, provided the destination is within $[1, n]$. Calculate the number of chambers the particle can never reach.

Solution

Let $\mathcal{R}$ be the set of reachable chambers. We want to compute $n - |\mathcal{R}|$. Without loss of generality, assume $a \leq b$. If $a > b$, we swap them.

Analysis

Since the particle can always perform moves of size a provided it stays within bounds, the reachability of a chamber is largely determined by its residue modulo a. If a chamber x is reachable, any other chamber $y \in [1, n]$ such that $y \equiv x \pmod{a}$ is likely also reachable, forming a contiguous interval of "reachable"indices for that specific residue.

Thus, the problem reduces to identifying the set of **reachable residues** modulo a. Let $S = (i - 1) \pmod{a}$ be the starting residue (using 0-based indexing).

We consider two cases based on the relationship between n and the step sizes.

Case 1: Wide Corridor ($n \geq a + b$)

If $n \geq a + b$, the corridor is large enough to allow the particle to mix the two step sizes freely without being permanently blocked by the boundaries. Specifically, one can always find a position to perform a shift of $\pm b$ and correct the position using shifts of $\pm a$.

In this case, the reachable residues are exactly those congruent to S modulo $\gcd(a, b)$. The number of reachable chambers is simply the count of integers in $[1, n]$ satisfying this modular condition.

Case 2: Narrow Corridor ($n < a + b$)

Since $a \leq b$ and $n < a + b$, we have $n - b < a$. This implies that moves of size b are strictly limited.

- A move $x \to x + b$ is valid only if $x \leq n - b$.
- A move $x \to x - b$ is valid only if $x \geq 1 + b$.

In terms of residues modulo a, let $u \in [0, a - 1]$. The smallest physical chamber corresponding to residue u is $u + 1$ (in 1-based indexing). For 0-based residues:

- **Safe Residues:** $u \in [0, n - b - 1]$. A particle at such a residue can jump forward by $+b$.
- **Forbidden Residues:** $u \in [n - b, a - 1]$. A particle here cannot jump forward by $+b$ because the destination would exceed n.

Visualizing with Islands

Imagine a islands arranged in a circle, numbered 0 to $a-1$. There is a potential bridge connecting every island u to island $(u+D) \pmod a$, where $D = b \pmod a$. However, the bridge originating at island u is only built if island u is **Safe**. If island u is **Forbidden**, the bridge between u and $(u+D)$ does not exist (the gap is impassable).

The frog starts at island S. Since the bridges are bidirectional, the frog can move:

- **Clockwise $(+D)$:** From u to $u+D$, provided the bridge at u exists (i.e., u is Safe).
- **Counter-Clockwise $(-D)$:** From v to $v-D$. This requires crossing the bridge that originates at $v-D$. Thus, the island $v-D$ must be Safe.

The set of reachable residues is simply the continuous chain of islands accessible from S via these existing bridges. The chain ends whenever we encounter a missing bridge in either direction.

We solve this by addressing two subproblems using Euclidean-like recursions:

1. Finding the Reachable Sequence (First Collision)

We must determine how many steps the frog can take forward before hitting a missing bridge. This is equivalent to finding the smallest integer $k \geq 1$ such that the residue lands in the forbidden range $[L, R]$:

$$(S + k \cdot D) \pmod a \in [L, R]$$

This is a standard "First Hit"problem. Let the target range relative to 0 be $[l', r']$.

- **Direct Check:** If the step size D is small enough that a multiple lands in $[l', r']$ without wrapping around the modulus a, we are done. The smallest such multiple is $D \cdot \lceil l'/D \rceil$.
- **Recursive Step:** If D is too large, every step wraps around a. We write $k \cdot D = q \cdot a + y$ (where $y \in [l', r']$). Rearranging, we get $q \cdot a \equiv -y \pmod D$. This effectively swaps the modulus and the step size ($a \pmod D$), reducing the problem size logarithmically ($O(\log a)$).

We calculate k_{fwd} (steps forward) and k_{back} (steps backward) using this logic.

The sequence repeats after CycleLength $= a/\gcd(a, D)$ steps. The total number of unique reachable residues is:

$$K = \min(\text{CycleLength}, k_{fwd} + k_{back} + 1)$$

2. Counting the Physical Chambers

Knowing the reachable residues form an arithmetic progression modulo a with length K, we must count how many actual indices in $[1, n]$ belong to these residues. Let $q = \lfloor (n-1)/a \rfloor$ and $r = (n-1) \pmod a$.

- Residues in range $[0, r]$ appear exactly $q+1$ times in the corridor.
- Residues in range $[r+1, a-1]$ appear exactly q times.

The total count $|\mathcal{R}|$ is given by:

$$|\mathcal{R}| = K \cdot q + (\text{Count of reachable residues } x \in \{v_0, \ldots, v_{K-1}\} \text{ such that } x \leq r)$$

The second term requires counting how many terms of the arithmetic progression lie in the range $[0, r]$. This summation can be computed using the **floor_sum** algorithm, which calculates $\sum_{i=0}^{K-1} \lfloor \frac{A \cdot i + B}{M} \rfloor$ in $O(\log M)$. By using the identity $x \pmod m = x - m\lfloor x/m \rfloor$, we can express the count as:

$$\text{Count} = \texttt{floor\_sum}(K, a, D, S+a) - \texttt{floor\_sum}(K, a, D, S+a-(r+1))$$

Alternative Solution ($O(\log^2 a)$)

Instead of deriving the first collision point analytically, one can use **Binary Search** on the number of steps k. To verify if a sequence of length k hits the forbidden range $[L, R]$, we simply count how many terms of the sequence $\{(S + i \cdot D) \pmod a\}_{i=1}^{k}$ fall into $[L, R]$. If this count is greater than 0, the particle has hit the wall.

This subproblem is identical to the counting problem in Case 2. We use `floor_sum` to count elements $\leq R$ and subtract the count of elements $\leq L - 1$:

$$\text{Hits} = \text{Count}_{\leq R}(k) - \text{Count}_{\leq L-1}(k)$$

Since each check takes $O(\log a)$ using `floor_sum` and the binary search adds a factor of $O(\log a)$, the total complexity is $O(\log^2 a)$, which fits comfortably within the time limits.

Complexity

- **Time Complexity:** $O(\log a)$ per test case due to the Euclidean-like recursions in both collision detection and counting.
- **Space Complexity:** $O(1)$.

Problem B. Easy Composite

Author: Ruhan Habib
Alter: Mohidul Haque Mridul
Tester: Mainul Hasan Mohsin
Solve Count vs Expected Solve Count: 97 / 90
First Solve: BUET_MST (0:06:38)

We make us of the fact that for any number divisible by 3, its sum of digits must also be divisible by 3.

Let s equal the sum of digits of p. Since $p \geq 10$ and p is prime, we can not have $s \equiv 0 \pmod 3$.

We can simply prepend 1 first.

If $s \equiv 2 \pmod 3$, then our new integer must be composite as $s + 1 \equiv 0 \pmod 3$. So we output 1 again.

Otherwise, we must have $s \equiv 1 \pmod 3$. In this case, $s + 2 \equiv 0 \pmod 3$. So prepending 2 to p gives us a composite number. We can simply output 2 this time.

Problem C. Path of Crows

Author: MD Mazed Hossain
Alter: Mohidul Haque Mridul, Arman Ferdous
Tester: Sakib Hassan
Solve Count vs Expected Solve Count: 61 / 40
First Solve: DIU_465UVaProblemsSolvedSir (0:54:14)

Suppose nodes u_1 and u_k are connected via the path $u_1 \to u_2 \to \cdots \to u_k$. Then we can make u_k directly connected to u_1 by the following series of $k - 2$ moves.

- $(a_1, b_1, c_1) = (u_k, u_{k-1}, u_{k-2})$
- $(a_2, b_2, c_2) = (u_k, u_{k-2}, u_{k-3})$

- $\cdots$

- $(a_{k-2}, b_{k-2}, c_{k-2}) = (u_k, u_2, u_1)$

This process only removes the edge (u_k, u_{k-1}) and adds a new edge (u_k, u_1) in the initial tree in $k-2$ moves.

So the procedure to create the chain is simple now. We start from p_1 and check if p_2 is already connected or not. If it is already connected we can proceed to connect p_3 with p_2. Otherwise, we use the above method to connect p_2 with p_1. Since any node is at most $n-1$ edge distance away from a specific node at any step, we require at most $n-2$ moves in one iteration, and there are $n-1$ iterations. So the total number of moves is bounded by $(n-2)(n-1) < n^2$.

Complexity: $\mathcal{O}(n^2)$, but $\mathcal{O}(n^3)$ solutions will also pass (e.g. simulating every single move individually and running a dfs after each move).

Problem D. Dual Star

Author: Mainul Hasan Mohsin
Alter: Nayeemul Islam Swad
Tester: Ruhan Habib
Solve Count vs Expected Solve Count: 25 / 40
First Solve: BUET_Supernova (1:08:20)

1) Reduce to 2D. The problem describes a Dual Star System where two identical planets C_1 and C_2 rotate in a circular orbit. The center of rotation is defined as the midpoint M of the two planet centers. Since the problem guarantees that the space station at the origin $(0, 0, 0)$ lies on the same plane as the orbit, we can treat this as a 2D geometry problem. Throughout the motion, the planets remain on opposite sides of the orbit, meaning C_1, M, and C_2 are always collinear.

2) Geometry setup. Let

$$d_1 = |OC_1|, \quad d_2 = |OC_2|.$$

Only the nearer planet matters, because any line from O that touches/intersects the nearer planet will also meet the other (opposite) planet on the same line. So choose

$$C = \begin{cases} C_1, & d_1 \le d_2, \\ C_2, & \text{otherwise.} \end{cases}$$

Define the orbital radius

$$R = |MC|.$$

3) "Touch angle"around M. Consider the line OM. By symmetry, this direction is the best candidate.

The planet is touchable by the line OM when OM is tangent to the planet. That happens when the angle between OM and MC is θ, where

$$\sin\theta = \frac{r}{R} \quad \Rightarrow \quad \theta = \arcsin\left(\frac{r}{R}\right).$$

4) Current angle α. Let

$$\vec{u} = \vec{MC} = \vec{C} - \vec{M}, \qquad \vec{v} = \vec{MO} = \vec{O} - \vec{M}.$$

Then the current angle is

$$\alpha = \arccos\left(\frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}\right).$$

Here, you can alternatively use cosine formula or any other method to find the angle α.

5) Final answer.

- If $\alpha \leq \theta$, alignment is possible at time 0.
- Otherwise, we need to reduce α down to θ, i.e. rotate by $(\alpha - \theta)$ radians. With angular speed ω (rad/s),

$$t = \frac{\alpha - \theta}{\omega}.$$

6) Precision notes. Use $\pi =$ `acos(-1.0)`. Clamp inputs of `acos` and `asin` to $[-1, 1]$ to avoid precision issues. Use a small epsilon (e.g. 10^{-9}) for comparisons.

Complexity: $O(1)$ per test case.

Problem E. Chorki

Author: Sakib Hassan
Alter: Arman Ferdous
Tester: Ahanaf Akif Pathan
Solve Count vs Expected Solve Count: 15 / 25
First Solve: DU_Rumbling (1:15:51)

Solution 1

We want to find a shift k such that the string T, where $T[i] = S[i] \oplus S[(i+k) \pmod n]$, is lexicographically as large as possible.

Lexicographical comparison works bit-by-bit from left to right. To make T maximal, we want $T[0]$ to be 1. If we have multiple shifts that make $T[0] = 1$, we look at $T[1]$, and so on.

Key observation: When comparing two different shifts k_1 and k_2 to see which produces a larger T:

1. Find the first index i where the two resulting strings differ: $T_{k_1}[i] \neq T_{k_2}[i]$.
2. This difference occurs if and only if $S[(i + k_1) \pmod n] \neq S[(i + k_2) \pmod n]$.
3. At this first point of difference, the shift that results in $T[i] = 1$ is the better one. Since $T[i] = S[i] \oplus S[i + k]$, we want $S[i + k]$ to be the opposite of $S[i]$.

The Suffix Array Approach:
A Suffix Array (and its accompanying LCP array) allows us to find the first index where two rotations of a string differ in $O(1)$ time.

The Algorithm:

1. Create a new string $W = S + S$ to handle cyclic shifts as linear substrings.
2. Construct the Suffix Array for W and the LCP (Longest Common Prefix) array.
3. Build a Sparse Table over the LCP array to answer Range Minimum Queries (RMQ). This allows us to find the LCP of any two suffixes in $O(1)$.
4. Initialize $best\_k = 0$. Iterate through all possible shifts k from 1 to $n - 1$:
   - Find $L = LCP(suffix\ best\_k,\ suffix\ k)$.
   - If $L < n$ (the rotations are not identical):
     - The first difference is at index $i = L$.
     - Compare $T_{best\_k}[L]$ and $T_k[L]$.
     - Since $T_x[L] = S[L] \oplus S[x + L]$, we check if $T_k[L] > T_{best\_k}[L]$.

– If it is, update $best\_k = k$.

5. Once the optimal k is found, calculate the final XORed string.

Time complexity: $O(n \log n)$

Hashing + Binary Search Approach:
One can use String Hashing combined with Binary Search. To find the first difference L between two shifts a and b:

1. Binary search over the range $[0, n-1]$ for the largest value L such that the hashes of $S[a \dots a+L-1]$ and $S[b \dots b+L-1]$ are equal.
2. This approach yields a complexity of $O(n \log n)$ depending on hash precomputation.

Solution 2

The problem can also be solved using **Suffix Automaton**.

To account for the cyclic nature of the shifts, we build the SAM on the concatenated string $SS = S + S$. This string of length $2n$ contains every cyclic shift of S as a substring of length n.

Once the SAM is constructed, we can find the optimal shift by performing a greedy walk starting from the root (initial state t_0):

1. Define the target string as the bitwise inverse of S: $\bar{S}[i] = S[i] \oplus 1$.
2. Starting at the SAM root, for each bit $i = 0, 1, \dots, n-1$:
   - Check if a transition exists for the target bit $\bar{S}[i]$.
   - **If exists:** Follow the transition for $\bar{S}[i]$. The XOR result for this bit $T[i]$ becomes 1.
   - **If not exists:** We are forced to follow the transition for $S[i]$. The XOR result for this bit $T[i]$ becomes 0.
3. Because the SAM is built on $S + S$, any path of length n starting from the root is guaranteed to correspond to at least one valid cyclic shift.

Multiple cyclic shifts that share the same prefix will end up at the same SAM state. By traversing the SAM transitions, we are simultaneously evaluating all valid shifts. The "greedy"choice at each step is optimal because lexicographical order prioritizes the earliest possible difference.

Time complexity: $O(n)$

Problem F. Mandatory XOR Problem

Author: Nayeemul Islam Swad
Alter & Tester: Ruhan Habib
Solve Count vs Expected Solve Count: 0 / 3
First Solve: N/A

Abridged Statement

Given n and S, compute

$$\sum_{\substack{a_1+\cdots+a_n=S \\ a_i \geq 0}} (a_1 \oplus a_2 \oplus \cdots \oplus a_n) \pmod{998244353}.$$

Step 1: Column-by-column view of addition

The constraint $a_1 + a_2 + \cdots + a_n = S$ is an arithmetic (not bitwise) constraint, while the quantity we're summing is bitwise. The key idea is to bridge these two worlds by thinking about addition **column by column in binary** similar to how you'd add numbers by hand.

Write each a_i in binary. At bit position b, each a_i contributes either a 0 or a 1. Let k_b denote the number of elements among $a_1, \ldots, a_n$ that have a 1-bit at position b. Then:

- The **XOR** at bit b is 1 if and only if k_b is odd.
- The **column sum** at bit b is k_b, which contributes $k_b \cdot 2^b$ to the arithmetic total.

In binary addition, the column sum k_b at position b doesn't just produce the b-th bit of S. It also produces a **carry** to higher bit positions. Formally, if c_b is the carry *into* column b (from column $b-1$), then:

$$k_b + c_b = s_b + 2\,c_{b+1}$$

where s_b is the b-th bit of S and c_{b+1} is the carry *out* of column b. We have boundary conditions $c_0 = 0$ (no carry into the least significant bit) and $c_B = 0$ (no carry left over after the most significant bit), where B is the number of bits needed to represent S.

The number of ways to choose *which* k_b of the n elements receive a 1-bit at position b is $\binom{n}{k_b}$. Since the choices at different bit positions are coupled only through the carries, this gives us the structure for a DP on carries.

Step 2: DP formulation

We process bit positions $b = 0, 1, 2, \ldots, B$ from the **least significant** to the **most significant**. (Here B is chosen large enough that $s_b = 0$ for all $b \geq B$. Concretely, $B = \lceil \log_2(S+1) \rceil$ suffices.)

State. A natural choice of state would be the carry c_b entering bit position b. But let's use a slightly different (and more convenient) combined state. Define:

$$h_b = k_{b-1} + c_{b-1}$$

That is, h_b is the sum of the bits distributed at position $b-1$ and the carry *into* position $b-1$. (Think of it as the "total column activity" at position $b-1$.) Since $k_{b-1} \leq n$ and one can show $c_{b-1} \leq n$ by induction, we have $0 \leq h_b \leq 2n$.

The carry is recovered from h_b via $c_b = (h_b - s_{b-1})/2$ (which is the carry relation $c_b = (k_{b-1} + c_{b-1} - s_{b-1})/2$ rewritten). For c_b to be a non-negative integer, we need $h_b \geq s_{b-1}$ and $h_b \equiv s_{b-1} \pmod 2$. This parity constraint is enforced naturally by the recurrence, as we'll see.

We maintain two DP tables:

- $f[b][h]$ = the number of ways to choose $(k_0, k_1, \ldots, k_{b-1})$ and their corresponding element assignments such that $h_b = h$.
- $g[b][h]$ = the sum of $\big($XOR contributed by bits $0, 1, \ldots, b-1\big)$ over all such ways, conditioned on $h_b = h$.

Base case: $f[0][0] = 1$, $g[0][0] = 0$ (before processing any bit, $h_0 = k_{-1} + c_{-1} = 0$, and no XOR has been accumulated).

Transition. At level b (corresponding to choosing k_{b-1} at bit position $b-1$), we pick $k \in \{0, 1, \ldots, n\}$ elements to receive a 1-bit. There are $\binom{n}{k}$ ways to choose which elements. The XOR contribution from bit $b-1$ is $2^{b-1} \cdot [k \text{ is odd}]$. The state update is:

$$h_b = k + c_{b-1} = k + \frac{h_{b-1} - s_{b-2}}{2}$$

where we use the convention $s_{-1} = 0$ (no bit of S below the LSB). Rearranging, $h_{b-1} = 2(h_b - k) + s_{b-2}$. This gives:

$$f[b][h] = \sum_{k=0}^{n} \binom{n}{k} f[b-1]\big[2(h-k) + s_{b-2}\big]$$

$$g[b][h] = \sum_{k=0}^{n} \binom{n}{k} \Big(g[b-1]\big[2(h-k) + s_{b-2}\big] + [k \text{ odd}] \cdot 2^{b-1} \cdot f[b-1]\big[2(h-k) + s_{b-2}\big] \Big)$$

(Terms where the index into $f[b-1]$ or $g[b-1]$ falls outside $[0, 2n]$ are zero.)

Answer: $g[B+1][0]$, where B is the bit-length of S. The condition $h_{B+1} = 0$ means $k_B + c_B = 0$: no bits are distributed at the virtual position above the MSB, and no carry remains. This ensures $a_1 + \cdots + a_n = S$ exactly.

Step 3: Recognizing the convolution

The transition above, as written, looks like it might require $O(n^2)$ time per bit position (iterating over both h and k). The crucial optimization is to recognize that the transition has the structure of a **polynomial multiplication** (convolution).

Recall the recurrence:

$$f[b][h] = \sum_{k=0}^{n} \binom{n}{k} \cdot f[b-1]\big[2(h-k) + s_{b-2}\big]$$

Define two polynomials:

$$P(x) = \sum_{k=0}^{n} \binom{n}{k} x^k$$

$$Q_b(x) = \sum_{m \geq 0} f[b-1][2m + s_{b-2}] \cdot x^m$$

The polynomial Q_b extracts every other entry of $f[b-1]$ (those matching the parity s_{b-2}) and packs them contiguously. Then:

$$f[b][h] = [x^h] \left(P(x) \cdot Q_b(x) \right)$$

This is a standard polynomial multiplication of two polynomials of degree $O(n)$.

For g (the XOR-sum table): The transition splits into two parts:

1. **Carrying forward the accumulated XOR sum:** Convolve $P(x)$ with the polynomial formed from $g[b-1]$ (same parity extraction as above).

2. **Adding the new XOR contribution at bit $b-1$:** This contributes 2^{b-1} only when k is odd. Convolve the "odd-only" version of P, namely $P_{\text{odd}}(x) = \sum_{k \text{ odd}} \binom{n}{k} x^k$, with $Q_b(x)$ (formed from $f[b-1]$), and multiply the result by 2^{b-1}.

In full:

$$g[b][h] = [x^h]\Big(P(x) \cdot Q_b^{(g)}(x)\Big) + 2^{b-1} \cdot [x^h]\Big(P_{\text{odd}}(x) \cdot Q_b^{(f)}(x)\Big)$$

where $Q_b^{(g)}$ and $Q_b^{(f)}$ are the parity-extracted polynomials from $g[b-1]$ and $f[b-1]$ respectively.

Step 4: NTT-based computation

Each transition requires a constant number of polynomial multiplications of degree-$O(n)$ polynomials. Using NTT over $\mathbb{Z}/998244353\mathbb{Z}$, each multiplication takes $O(n \log n)$ time.

There are $B+1 = O(\log S)$ bit positions to process. So the total time per test case is $O(n \log n \cdot \log S)$.

Problem G. Genome Evolution

Author: Ahanaf Akif Pathan
Alter & Tester: Mohidul Haque Mridul
Solve Count vs Expected Solve Count: 110 / 114
First Solve: MBSTU_UdoMada (0:03:25)

From the statement, it follows that for $t \geq 3$ the genome value satisfies

$$G(t) = G(t-2) \,\&\, G(t-1)$$

Thus, the recurrence becomes

$$G(t) = \begin{cases} a, & \text{if } t = 1, \\ b, & \text{if } t = 2, \\ G(t-1) \,\&\, G(t-2), & \text{if } t \geq 3. \end{cases}$$

Let us compute the first few terms of $G(t)$:

- $G(1) = a$
- $G(2) = b$
- $G(3) = G(1) \,\&\, G(2) = a \,\&\, b$
- $G(4) = G(2) \,\&\, G(3) = b \,\&\, (a \,\&\, b) = a \,\&\, b$
- $G(5) = G(3) \,\&\, G(4) = (a \,\&\, b) \,\&\, (a \,\&\, b) = a \,\&\, b$
- $\ldots$

Hence, for all $t \geq 3$, the value stabilizes at $a \,\&\, b$. Therefore, the recurrence simplifies to

$$G(t) = \begin{cases} a, & \text{if } t = 1, \\ b, & \text{if } t = 2, \\ a \,\&\, b, & \text{if } t \geq 3. \end{cases}$$

Time complexity: $O(1)$ per testcase.

Problem H. Pothchola

Author: Sakib Hassan
Alter: Ahanaf Akif Pathan
Tester: Mainul Hasan Mohsin
Solve Count vs Expected Solve Count: 63 / 30
First Solve: BUET_Supernova (0:32:41)

Bitwise Observations

Let $msb(x)$ be the index of the most significant bit of x. We analyze the edge conditions based on the relative values of $msb(a_u)$ and $msb(a_v)$.

Case 1: $msb(a_u) > msb(a_v)$

If $msb(a_u) = m$ and $msb(a_v) = k$ with $m > k$, then $msb(a_u \oplus a_v) = m$. The value $a_u \oplus a_v$ is at least 2^m, while $a_v < 2^k < 2^m$. Thus $a_u \oplus a_v > a_v$, which violates the second condition.

Case 2: $msb(a_u) = msb(a_v)$

Let $msb(a_u) = msb(a_v) = k$. In the XOR sum $a_u \oplus a_v$, bit k becomes $1 \oplus 1 = 0$. This means $msb(a_u \oplus a_v) < k$. Consequently, $a_u \oplus a_v < 2^k \leq a_u$, which violates the first condition.

Case 3: $msb(a_u) < msb(a_v)$

Let $msb(a_u) = j$ and $msb(a_v) = k$ where $j < k$.

1. $msb(a_u \oplus a_v) = k$ (since bit k is set in a_v and unset in a_u). Since $k > j$, $a_u \oplus a_v > a_u$ is always true.
2. For $a_u \oplus a_v < a_v$, consider the bits from k down to j. Bits k through $j+1$ are identical in both. At bit j, a_u has a 1. The bit j of $a_u \oplus a_v$ is $1 \oplus bit_j(a_v)$. To satisfy $a_u \oplus a_v < a_v$, bit j of $a_u \oplus a_v$ must be 0, which implies $bit_j(a_v) = 1$.

So, an edge $u \to v$ exists if and only if $msb(a_u) < msb(a_v)$ and the $msb(a_u)$-th bit of a_v is set to 1.

The condition $msb(a_u) < msb(a_v)$ ensures the graph is a Directed Acyclic Graph (DAG). We can compute the longest path using DP.

DP Approach

Let $dp[k]$ be the maximum path length ending at a node whose value has $msb = k$.

For each a_i in the input:

1. Let $k = msb(a_i)$.
2. Find the best previous path that can connect to this node:

$$\text{best\_prev} = \max(\{dp[j] \mid 0 \leq j < k \text{ and bit } j \text{ of } a_i \text{ is } 1\} \cup \{0\})$$

3. Update the DP state for bit k: $dp[k] = \max(dp[k], best\_prev + 1)$.

Complexity

- **Time Complexity:** $O(n \cdot \log(\max A_i))$.

Problem I. Cosmic Bit Flip

Author: Arman Ferdous
Alter: Ruhan Habib
Tester: Mohidul Haque Mridul
Solve Count vs Expected Solve Count: 0 / 5
First Solve: N/A

The idea is to use Gaussian elimination. Let x_i be the number of lay-lines in the ith city. Also, $a_{ij} = 1$ if jth city was visited on the ith tour and 0 otherwise. Then we can create this system of liner equations -

$$AX \equiv P \pmod 2,$$

where

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \in \mathbb{Z}_2^{m \times n}, \quad X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{Z}_2^{n \times 1}, \quad P = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_m \end{pmatrix} \in \mathbb{Z}_2^{m \times 1}.$$

As $\mathbb{Z}_2^k$ is a vector space with XOR as it's operation, we can apply Gaussian elimination to solve this system for $x_i \pmod 2$. To make this faster across m updates we use some optimizations. First we describe a $\mathcal{O}(\frac{nm^2}{64})$ solution and then we show how to speed it up.

$\mathcal{O}(\frac{nm^2}{64})$ **Idea**: Instead of starting with the augmented matrix $[A \mid P]$ for the row reduction process, we will instead start with $[A \mid I_{m \times m}]$ (I is identity matrix). Using bitsets, we can reduce A into row reduced echelon form within $\mathcal{O}(\frac{nm^2}{64})$. After reaching this step, let A' be the modified left matrix and I' be the modified right matrix.

Now, suppose the ith row of A' contains all zeros. Then $\sum_{j=1}^{m} I'_{ij} \times p_j \equiv 0 \pmod 2$ must hold. If this is not the case for any zero row, then the answer is impossible for that query.

If the answer is not impossible, we then consider the rank (number of non-zero rows) of A'. If it is less than n, then it is not possible to uniquely determine all node values $x_i \pmod 2$ and the answer will be unknown.

Otherwise, the rank will be exactly n, and we can uniquely determine all $x_i \pmod 2$ and find the sum. This part of the solution takes $\mathcal{O}(m^2)$ because for each p_i update, we need to update m row results in P'.

Speed-up to $\mathcal{O}(\frac{n^2m}{64})$: We will insert the rows of A and I one by one instead of starting with the whole $[A \mid I_{m \times m}]$ at once. After inserting a row, we immediately do all the elimination operations with the previous rows (this is the same technique used in calculating the XOR-basis). If we see that the newly inserted row is a zero row, then we do not insert it in the augmenting matrix, but store it separately (we need the right portion of it to check impossible cases). Also, don't create a new column in P for any zero rows, otherwise P will be of dimension $n \times m$, we need it to be $n \times n$. This optimization will let us build the row reduced echelon matrix in $\mathcal{O}(\frac{n^2m}{64})$.

The rest of the evaluation is straight forward with some preprocessing involved. As the number of rows in A' and I' are now n instead of m, we can use any $\mathcal{O}(nm)$ method for it to pass.

Problem J. Prefix Reversal

Author: Ahanaf Akif Pathan
Alter: Mainul Hasan Mohsin

Tester: MD Mazed Hossain
Solve Count vs Expected Solve Count: 91 / 65
First Solve: BUET_GreyMatter (0:12:16)

Let us illustrate the idea with a concrete example. Suppose $n = 5$ and $A = [a_1,\ a_2,\ a_3,\ a_4,\ a_5]$.

By definition, the values of $F(i, 5, A)$ are:

- $F(1, 5, A) = 1 \times a_1 + 2 \times a_2 + 3 \times a_3 + 4 \times a_4 + 5 \times a_5$
- $F(2, 5, A) = 1 \times a_2 + 2 \times a_1 + 3 \times a_3 + 4 \times a_4 + 5 \times a_5$
- $F(3, 5, A) = 1 \times a_3 + 2 \times a_2 + 3 \times a_1 + 4 \times a_4 + 5 \times a_5$
- $F(4, 5, A) = 1 \times a_4 + 2 \times a_3 + 3 \times a_2 + 4 \times a_1 + 5 \times a_5$
- $F(5, 5, A) = 1 \times a_5 + 2 \times a_4 + 3 \times a_3 + 4 \times a_2 + 5 \times a_1$

Now observe how the value changes when we move from $F(i-1, 5, A)$ to $F(i, 5, A)$:

- $F(2, 5, A) - F(1, 5, A) = a_1 - a_2 = (\sum_{i=1}^{1} a_i) - 1 \times a_2$
- $F(3, 5, A) - F(2, 5, A) = a_1 + a_2 - 2 \times a_3 = (\sum_{i=1}^{2} a_i) - 2 \times a_3$
- $F(4, 5, A) - F(3, 5, A) = a_1 + a_2 + a_3 - 3 \times a_4 = (\sum_{i=1}^{3} a_i) - 3 \times a_4$
- $F(5, 5, A) - F(4, 5, A) = a_1 + a_2 + a_3 + a_4 - 4 \times a_5 = (\sum_{i=1}^{4} a_i) - 4 \times a_5$

From the pattern above, we can write the recurrence relation.

Formally, for $i > 1$,

$$F(i, n, A) = F(i-1, n, A) + \sum_{j=1}^{i-1} a_j - (i-1) \times a_i$$

The prefix sum array allows us to compute $\sum_{j=1}^{i-1} a_j$ in constant time, so each value $F(i, n, A)$ can be computed efficiently. After evaluating $F(i, n, A)$ for all $i = 1 \ldots n$, we can sort the permutation according to these values.

Time complexity: $O(n)$ per testcase.

Problem K. Lottery

Author: Ruhan Habib
Alter & Tester: Nayeemul Islam Swad
Solve Count vs Expected Solve Count: 0 / 2
First Solve: N/A

First, we let $T = \sum_{j=1}^{n} t_j$ and let L denote the maximum number of tickets a regular person could have bought. In our case, $L = 5$. Now, let us rewrite the formula for $f(x)$:

$$f(x) = \sum_{\substack{A \subseteq \{1,\ldots,n\} \\ |A| \le m-1}} (-1)^{m-1-|A|} \binom{n-|A|}{m-1-|A|} \left(\frac{x}{x + T - \sum_{j \in A} t_j} \right) \tag{1}$$

Observation 1. Getting the kth prize with m draws is equivalent to getting first prize with $m - (k-1)$ draws. So the given formula is enough for us.

Observation 2. The inner sum only depends on $|A|$ and $\sum_{j\in A} t_j$. We also note that $0 \le |A| \le m-1$ implies that $0 \le \sum_{j\in A} t_j \le (m-1)L$.

For all $a \in [0..m-1]$ and $s \in [0..(m-1)L]$, we define $\varphi(a,s)$ and $\psi(a,s)$ as follows.

$$\varphi(a,s) = \left|\{A : |A| = a \wedge \sum_{j\in A} t_j = s\}\right| \tag{2}$$

$$\psi(s) = \sum_{a=0}^{m-1} (-1)^{m-1-a} \binom{n-a}{m-1-a} \varphi(a,s) \tag{3}$$

With some rearrangment, our sum becomes

$$f(x) = \sum_{s=0}^{(m-1)L} \left(\frac{x}{x+T-s}\right) \psi(s) \tag{4}$$

Observation 3. We can compute φ using 2D convolution. Let p and q be indeterminates. Then

$$g(p,q) = \left(\prod_{j=1}^{n} \left(1 + p^1 q^{t_j}\right) \pmod{p^m}\right) \tag{5}$$

$$\varphi(a,s) = [p^a q^s]\, g(p,q) \tag{6}$$

Observation 4. Since $\sum_{j\in A} t_j \le (m-1)L < mL$ when $|A| \le m-1$, the exponent of q in the polynomial g can never be greater than or equal to mL. So we can "merge" p and q into a single variable y, where we represent p^1 using y^{mL} and $p^1 q^{t_j}$ using y^{mL+t_j}. So,

$$h(y) = \left(\prod_{j=1}^{n} \left(1 + y^{mL+t_j}\right) \pmod{y^{m\cdot mL}}\right) \tag{7}$$

$$\varphi(a,s) = \left[y^{amL+s}\right] h(y) \tag{8}$$

Observation 5. $h(y)$ is a product of at most L unique polynomials. For $j \in [1..L]$, let $c_j = |\{z : t_z = j\}|$. Then

$$h(y) = \prod_{j=1}^{L} \left(1 + y^{mL+j}\right)^{c_j} \pmod{y^{m\cdot mL}} \tag{9}$$

$$= \prod_{j=1}^{L} \left(\sum_{r=0}^{c_j} \binom{c_j}{r} y^{r(mL+j)}\right) \pmod{y^{m^2 L}} \tag{10}$$

We can now compute the polynomial $h(y)$ using NTT in $O\left(L \cdot m^2 L \log\left(m^2 L\right)\right)$.

We can now pre-compute φ using equation (8). Since we have φ, we can pre-compute ψ as well, using equation (3). This pre-computation takes $O\left(m^2 L\right)$.

Finally, for each value of $x \in \{1, \ldots, l\}$, we can now simply use equation (4) to compute $f(x)$ in total time $O\left(lmL\right)$.

The overall time complexity of our solution is $O\left(m^2 L^2 \log\left(m^2 L\right) \log n + lmL\right)$.

Bonus

In this section, we will give a rough sketch showing why the given formula for $f(x)$ is correct. The following calculations are quite classic (as in well-known).

Let $\mathbb{N} = \{1, 2, 3, \dots\}$. For integers $n \geq 0$, $1 \leq m \leq n+1$, $t = (t_1, \dots, t_{n+1}) \in \mathbb{N}^{n+1}$, let $g(n, m, t)$ denote the probability that among n people, the mth draw is won by the $n+1$-th participant.

Lemma. Suppose that we have integers $n, m, t_1, \dots, t_{n+1}$ with $n \geq 0$, $1 \leq m \leq n+1$, and $t_i \geq 1 \forall i$. Let $T_1, \dots, T_{n+1}$ be random variables with distribution $T_i \sim \text{Exp}(t_i)$ for all $i \in [1..n]$. Then

$$g(n, m, t) = \mathbb{P}\left[T_{n+1} \text{ is the } m\text{-th smallest among } \{T_1, \dots, T_{n+1}\}\right]$$

Proof. We proceed by induction. The base-case $n = 0$ is obvious. Now, let $n, m, t_1, \dots, t_{n+1} \in \mathbb{N}$ such that $1 \leq m \leq n$. Suppose that our lemma is true for smaller values of n. If $m = 1$, then

$$\begin{aligned}
\mathbb{P}\left[T_{n+1} = \min_j T_j\right] &= \mathbb{P}\left[T_{n+1} \leq T_j \forall j \in [1..n]\right] \\
&= \int_0^\infty t_{n+1} e^{-t_{n+1}s} \prod_{j \in [1..n]} e^{-t_j s} \, \mathrm{d}s \\
&= \int_0^\infty t_{n+1} \exp\left(-s \sum_{j=1}^{n+1} t_j\right) \mathrm{d}s \\
&= \frac{t_{n+1}}{\sum_{j \in [1..n+1]} t_j} \\
&= g(n, 1, t)
\end{aligned}$$

If $m \geq 2$, then

$$\begin{aligned}
&\mathbb{P}\left[T_{n+1} \text{ is the } m\text{-th of } \{T_1, \dots, T_{n+1}\}\right] \\
&= \sum_{i=1}^{n} \mathbb{P}\left[T_{n+1} \text{ is the } (m-1)\text{-th of } \{T_1, \dots, T_{i-1}, T_{i+1}, \dots, T_{n+1}\} | T_i = \min_j T_j\right] \cdot \mathbb{P}\left[T_i = \min_j T_j\right] \\
&= \sum_{i=1}^{n} \mathbb{P}\left[T_{n+1} \text{ is the } (m-1)\text{-th of } \{T_1, \dots, T_{i-1}, T_{i+1}, \dots, T_{n+1}\}\right] \cdot \mathbb{P}\left[T_i = \min_j T_j\right] \\
&= \sum_{i=1}^{n} g(n-1, m-1, (t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_{n+1})) \cdot \frac{t_i}{\sum_j t_j} \\
&= g(n, m, t)
\end{aligned}$$

So by induction, our claim holds true. □

So,

$$
\begin{aligned}
& f(x) \\
& = g\left(m, n, \left(t_1, \ldots, t_n, x\right)\right) \\
& = \mathbb{P}\left[T_{n+1} \text{ is the } m\text{-th of } \{T_1, \ldots, T_{n+1}\}\right] \\
& = \sum_{\substack{S \subseteq [1..n] \\ |S|=m-1}} \int_0^\infty x e^{-xs} \mathbb{P}\left[T_j < s \forall j \notin S\right] \cdot \mathbb{P}\left[T_j > s \forall j \in S\right] \, \mathrm{d}s \\
& = \sum_{\substack{S \subseteq [1..n] \\ |S|=m-1}} \int_0^\infty x e^{-xs} \left(\prod_{j \notin S} \mathbb{P}\left[T_j < s\right]\right) \left(\prod_{j \in S} \mathbb{P}\left[T_j > s\right]\right) \, \mathrm{d}s \\
& = \sum_{\substack{S \subseteq [1..n] \\ |S|=m-1}} \int_0^\infty x e^{-xs} \left(\prod_{j \notin S} e^{-t_j s}\right) \left(\prod_{j \in S} \left(1 - e^{-t_j s}\right)\right) \, \mathrm{d}s \\
& = \sum_{\substack{S \subseteq [1..n] \\ |S|=m-1}} \int_0^\infty \sum_{A \subseteq S} x e^{-xs} \left(-1\right)^{m-1-|A|} \prod_{j \notin A} e^{-t_j s} \, \mathrm{d}s \\
& = \sum_{\substack{A \subseteq [1..n] \\ |A| \leq m-1}} \int_0^\infty \sum_{\substack{S \supseteq A \\ |S|=m-1}} x e^{-xs} \left(-1\right)^{m-1-|A|} \prod_{j \notin A} e^{-t_j s} \, \mathrm{d}s \\
& = \sum_{\substack{A \subseteq [1..n] \\ |A| \leq m-1}} \int_0^\infty \binom{n-|A|}{m-1-|A|} x e^{-xs} \left(-1\right)^{m-1-|A|} \prod_{j \notin A} e^{-t_j s} \, \mathrm{d}s \\
& = \sum_{\substack{A \subseteq [1..n] \\ |A| \leq m-1}} \binom{n-|A|}{m-1-|A|} \left(-1\right)^{m-1-|A|} x \exp\left(-\left(x + \sum_{j \notin A} t_j\right)\right) \, \mathrm{d}s \\
& = \sum_{\substack{A \subseteq [1..n] \\ |A| \leq m-1}} \binom{n-|A|}{m-1-|A|} \left(-1\right)^{m-1-|A|} \left(\frac{x}{x + \sum_{j \notin A} t_j}\right)
\end{aligned}
$$

Problem L. Sapure

Author: MD Mazed Hossain
Alter: Sakib Hassan
Tester: Mohidul Haque Mridul
Solve Count vs Expected Solve Count: 93 / 70
First Solve: GodFathers Inc. (0:31:29)

One can simulate the snake moves easily by maintaining a queue structure of the cells the snake is at. If the snake is simply moving through a cell, only the tail gets removed (pop) and the head moves in the direction it is going (push). If the snake ate some food in that move, then we don't move the tail in that step.

To check for collisions, we need to maintain a set/map like structure of the cells the snake currently occupies.

Complexity: $\mathcal{O}(k \lg m)$.