Problem Tutorial: "Numerous Elimination"

We introduce two solutions: an $O(N)$ time solution using dynamic programming and an $O(\log N)$ time solution by cleverly transforming the formula.

$O(N)$ Solution

Let $dp[i]$ be the number of matches played when i players participate in the tournament. Clearly, $dp[1] = 0$. Since the number of matches doesn't change based on the order of matches, we consider adding one player after $i-1$ players have played matches.

Initially, after $i-1$ players have played $dp[i-1]$ matches, there is one player in each column from column 0 to column $i-2$. We then add one player to column 0 and proceed to play $i-1$ matches sequentially from column 0 to column $i-2$. As a result, there will be $i-1$ players in column 0 and one player in column $i-1$. Finally, after playing $dp[i-1]$ matches, every column will have one player, and the tournament ends.

Therefore, when conducting a tournament with i players, $2 \times dp[i-1] + i - 1$ matches are played, and we obtain the formula:

$$dp[i] = 2 \times dp[i-1] + i - 1$$

By calculating the values of $dp[i]$ for $i = 2, \ldots, N$ in order, we can solve this problem in $O(N)$ time.

$O(\log N)$ Solution

By cleverly transforming the formula obtained earlier, we can find the answer more efficiently.

$$\begin{aligned}
dp[n] &= 2 \times dp[n-1] + n - 1 \\
&= dp[1] \times 2^{n-1} + \sum_{i=2}^{n} (i-1) \times 2^{n-i} \\
&= \sum_{i=2}^{n} \sum_{j=i}^{n} 2^{n-j} \\
&= \sum_{i=2}^{n} 2^{n-i+1} - 1 \\
&= (2^n - 2) - (n-1) \\
&= 2^n - n - 1
\end{aligned}$$

Using exponentiation by squaring, we can calculate this in $O(\log N)$ time.

Problem Tutorial: "Almost Large"

Consider a graph with N vertices. If we can change S_i to S_j in one operation, add an edge $i \to j$. We just need to check whether it is possible to reach from 1 to N to answer the problem. However, since there are $O(N^2)$ possible edges to consider, it is not suitable.

Imagine the operation as the following sequence:

- Choose a digit and set its value to 0.
- Then, repeat increasing any digit by one (if the digit's value is less than 2).

- End the operation if it matches any S_i.

Considering the above operation, you can create the following graph. Let $M = 12$ as the maximum number of digits.

- Prepare 3^M vertices $a_0, a_1, \ldots, a_{3^M-1}$ and N vertices $b_1, \ldots, b_N$.
- For $i = 1, 2, \ldots, N$, add an edge from a_{S_i} to b_i.
- For S_i in ternary representation as $n_1 n_2 \ldots n_M$, for $j = 1, 2, \ldots, M$, add an edge from b_i to a_x where $x = n_1 n_2 \ldots n_{j-1} 0 n_{j+1} \ldots n_M$.
- For $i = 0, 1, \ldots, 3^M - 1$, if the ternary representation of i is $n_1 n_2 \ldots n_M$, and for $j = 1, 2, \ldots, M$, if $n_j < 2$, add an edge from a_i to a_x where $x = n_1 n_2 \ldots n_{j-1}(n_j + 1) n_{j+1} \ldots n_M$.

By determining whether it is possible to reach from b_1 to b_N in this graph, you can solve the problem. By using DFS, the time complexity is $O(M(3^M + N))$, where it is suitable for this problem.

Problem Tutorial: "Yet Another Simple Math Problem"

For pairs of positive integers satisfying the conditions (a, b), we can prove that there is only one pair of positive integers (x, y) satisfying $x + y^2 = a$ and $x^2 + y = b$.

To prove it, assume there are 2 positive integers (x_1, y_1) and (x_2, y_2) satisfiying $x_1 + y_1^2 = x_2 + y_2^2 = a$ and $x_1^2 + y_1 = x_2^2 + y_2 = b$, then obtain $x_1 = x_2$ and $y_1 = y_2$.

Consequently, the problem can be reformulated as follows: Find the number of pairs of positive integers (x, y) satisfying $1 \leq x + y^2, x^2 + y \leq N$.

Once the problem is reduced, various methods can be employed to solve it.

Solution

Let k be the maximum non-negative integer satisfying $k^2 + k \leq N$. The answer is then given by $k^2 + 2 \times \max\{0, N - (k+1)^2\}$. Therefore, finding k using binary search or other methods allows solving the problem in $O(\log N)$ time per test case.

In the following, we provide the proof for this solution. When actually solving the problem, you can write a straightforward code to output the answers and observe the results for $N = 1, 2, 3, \ldots$ to predict the answer formula.

Proof

We define "condition" as $1 \leq x + y^2, x^2 + y \leq N$.

observation 1

All pairs of positive integers (x, y) satisfying $1 \leq x, y \leq k$ satisfy the condition, and there are a total of k^2 such pairs.

This follows from $1 \leq x + y^2, x^2 + y \leq k^2 + k \leq N$.

observation 2

Any pair of positive integers (x, y) satisfying $x \geq k+2$ or $y \geq k+2$ does not satisfy the conditions.

Without loss of generality, we can assume $x \geq y$ due to symmetry in the condition. In this case,

$$\begin{aligned} x - y &\leq (x-y)(x+y) \\ &= x^2 - y^2 \end{aligned}$$

implies $x + y^2 \leq x^2 + y$. Therefore,

$$\begin{aligned} x^2 + y &\geq (k+2)^2 \\ &= (k+1)^2 + 2(k+1) + 1 \\ &\geq (k+1)^2 + (k+1) \\ &> N. \end{aligned}$$

We used the assumption, "k is the maximum non-negative integer satisfying $k^2 + k \leq N$", in the last inequality. Since $x^2 + y > N$, the pair (x, y) does not satisfy the condition.

observation 3

The number of pairs (x, y) satisfying $x = k+1$ or $y = k+1$ is $2 \times \max\{0, N - (k+1)^2\}$.

We would like to find the number of pairs of positive integers (x, y) satisfying the condition, where $x = k+1$ or $y = k+1$.

Due to symmetry in the condition, without loss of generality, we can assume $x \geq y$. Then we only need to consider cases where $x = k+1$ and $y \leq k+1$. In this case, $x + y^2 \leq x^2 + y$ holds. Therefore, the only crucial factor is whether $x^2 + y \leq N$.

Since $x = k+1$,

$$\begin{aligned} y &\leq N - x^2 \\ &= N - (k+1)^2. \end{aligned}$$

As $N < (k+1)^2 + (k+1)$ based on the assumption about k, we have $N - (k+1)^2 < k+1$. Thus, the number of positive integers y satisfying the condition is $\max\{0, N - (k+1)^2\}$.

Since we automatically determined $x = k+1 > y$, we don't need to consider cases where $x = y$. Therefore, the number of pairs (x, y) satisfying $x = k+1$ or $y = k+1$ is $2 \times \max\{0, N - (k+1)^2\}$.

Problem Tutorial: "Spacecraft"

Consider a spaceship, represented by a sphere with the center at the origin and radius R, denoted as B. When considering the region enclosed by half-lines with endpoints P_i that are tangent to B, the shape of this region forms a cone with P_i as its vertex.

Let λ_i be the result of translating this cone so that its vertex is at the origin O. Additionally, if we cut the original cone, before translation, along the intersection with the surface of the sphere, and retain the side that does not contain P_i, we obtain the region denoted as Λ_i. This region corresponds to the area where the i-th star cannot be seen.

However, the figures are considered as sets of points in $\mathbb{R}^3$. Also, λ_i and Λ_i include their boundaries.

This problem seeks to determine the number of regions into which the space S, represented by a sphere with a radius of 1 centered at the origin, is divided when excluding the union of all the translated cones, i.e., $S \setminus \bigcup_i \lambda_i$.

To establish the validity of this reduction, we provide a proof.

Proof

Points in 3-dimensional space are identified with 3-dimensional vectors. If a point p is lovely, then for any real number k greater than 1, the point kp is also lovely (i.e., moving in the direction away from the origin is also considered lovely). Therefore, consider the direction $\frac{p}{|p|}$ as seen from the origin, for any point p excluding the origin. At this point, the following two conditions are equivalent for a direction d ($|d| = 1$).

- There exists a lovely point with direction d.
- $d \in (S \setminus \bigcup_i \lambda_i)$.

The above condition can be rephrased as $d \notin \bigcup_i \lambda_i$ for all i, since $|d| = 1$. As λ_i is completely contained in the region where the i-th star cannot be seen, the equivalence between the conditions above is evident. The reverse implication is also satisfied because the "side" of λ_i and Λ_i is parallel, allowing the condition $d \notin \Lambda_i$ to be met by choosing a sufficiently large real number $F(d)$. Therefore, the reduction is valid.

Before confirming the validity of the reduction, we introduce some notation. We define points that are not in $\bigcup_i \lambda_i$ on the sphere S as "lovely points" on the sphere S. The set of points that are lovely on the sphere S is denoted as L_S. Furthermore, two points p_1 and p_2 belonging to L_S are considered to be in the same connected component if there exists a curve on L_S connecting p_1 and p_2, denoted as $p_1 \sim_S p_2$.

Now, let's confirm the validity of the reduction. It is sufficient to show that $p_1 \sim p_2$ if and only if $p_1/|p_1| \sim_S p_2/|p_2|$ for lovely points p_1 and p_2.

- First, we demonstrate $p_1 \sim p_2 \to p_1/|p_1| \sim_S p_2/|p_2|$. Take a curve on L connecting p_1 and p_2, and let $f : L \to L_S$ be defined as $f(p) = \frac{p}{|p|}$. Since f is clearly continuous, the curve becomes a curve on L_S, and immediately, $p_1/|p_1| \sim_S p_2/|p_2|$ follows.
- Next, we show $p_1 \sim p_2 \leftarrow p_1/|p_1| \sim_S p_2/|p_2|$. Take a curve on L_S connecting $p_1/|p_1|$ and $p_2/|p_2|$, and let $g : L_S \to L$ be defined as $g(d) = F(d)d$. Since g is also clearly continuous, the curve becomes a finite-length curve on L, and immediately, $p_1 \sim p_2$ follows.

Reduced Problem Analysis

The intersection of S and λ_i forms circles. The figure in the next page shows these circles and S in one of the sample test cases.

Next, carefully observe the reduced problem. When several circles are drawn on the sphere, the problem becomes how many regions are formed outside all these circles on the sphere. Here, pay attention to the fact that these circles drawn on the sphere each lie on a plane, and rephrase the problem as follows.

Cut the sphere S with several planes and retain only the parts on the same side as the origin concerning the planes. How many regions are formed?

Let σ_i be the plane that includes the circle that serves as the boundary between S and λ_i, and denote the half-space on the same side as the origin with respect to σ_i as H_i. Consider the convex hull C defined as the intersection of half-spaces $H_1, H_2, ..., H_N$. Additionally, add half-spaces as necessary to ensure that C is finite. If we can determine how many regions $S \cap C'$ is divided into, where C' is the part of C whose distance from the origin is greater than 1, then we obtain the answer to the original problem.

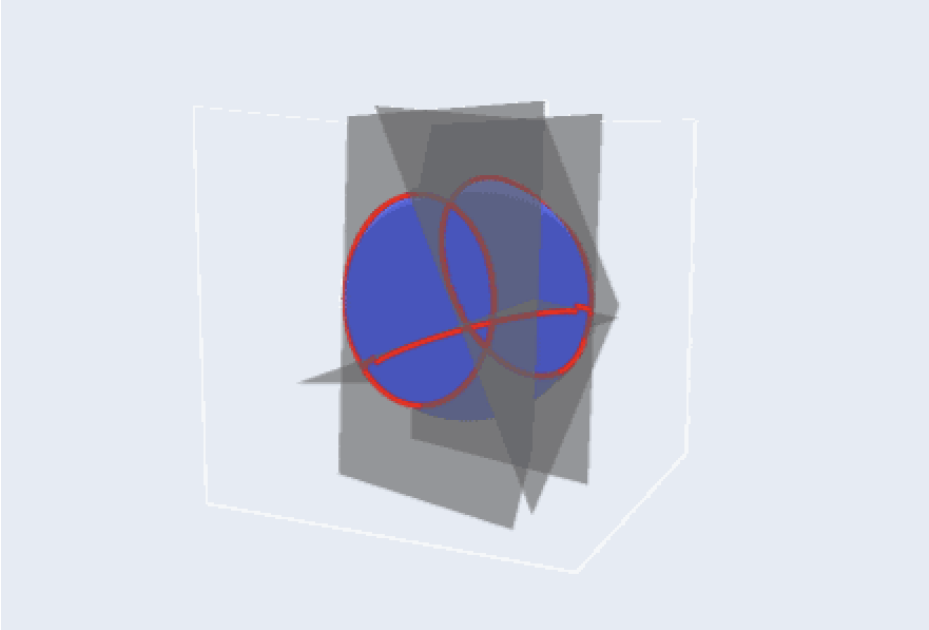

Рис. 1: The intersection of S and λ_i forms a circle, which lies on a plane.

Since the points on C that maximally distance from the origin are only its vertices, to ensure that two different vertices are in the same region, it is necessary and sufficient for the region to be reachable by traversing along the edges of C' on the sphere. From these considerations, the following algorithm is derived:

- Compute the planes σ_i for $i = 1, 2, ..., N$, and calculate the convex hull C as the intersection of the half-spaces defined by σ_i.
- Add half-spaces as necessary to ensure that C is finite, including S entirely if needed.
- Determine the number of connected components in the graph formed by points and edges

Several algorithms are known for computing the convex hull as the intersection of half-spaces. For example, by examining all planes in N iterations and calculating the intersection of half-planes on those planes using angular sorting, it is possible to compute the half-plane intersection in $O(N \log N)$ time. Consequently, the overall time complexity for computing the convex hull is $O(N^2 \log N)$ per case. Since the number of edges and vertices is $O(N)$, determining the number of connected components does not become the bottleneck in the algorithm. This part can be efficiently handled using data structures such as UnionFind.

Problem Tutorial: "R-Connected Components"

Solution

We can find the correct value of the answer with restricting the range of vertices from $\mathbb{Z}^2$ to about $2\sqrt{R} \times 2\sqrt{R}$. While the solution results in TLE, comparing the prime factorization results of an integer R with it, we can expect the answer is as follows.

- If any a prime number of form $4n + 3$ occurs an odd number of times in the prime factorization of R, the answer is `inf`.
- Otherwise, the answer is obtained by removing all prime nubmers of form $4n + 1$ from the prime factorization of R.

In fact, this is correct. By factorizing R in $O(\sqrt{R})$ time, this problem can be solved.

Let's proceed with proving this.

Proof

Introducing the imaginary unit i, we can factorize $x^2 + y^2 = (x + yi)(x - yi)$. This problem seems to be related to factorization over Gaussian integers.

Denote the Gaussian integer ring as $\mathbb{Z}[i]$ and consider the vertex set as $\mathbb{Z}[i]$.

Define $W(R) := \{w \in \mathbb{Z}[i] \mid w\overline{w} = R\}$, where $\overline{w}$ denotes the complex conjugate of w. $W(R)$ represents the relative positions of vertices that can be directly connected by edges.

When we label the elements of $W(R)$ as $W(R) = \{w_1, w_2, \ldots, w_k\}$, then a vertex x being connected to the vertex 0 is equivalent to the existence of a sequence of non-negative integer $n_1, n_2, \ldots, n_k$ such that $x = \left(\sum_{i=1}^{k} n_i w_i\right)$.

Here, since $w \in W(R) \implies iw, -w, -iw \in W(R)$ holds, it is acceptable to relax the sequence of non-negative integers $n_1, n_2, \ldots, n_k$ to Gaussian integers.

Now, considering the properties of the Gaussian integer ring:

- Primes of the form $4n + 3$ are Gaussian primes.
- Uniqueness of prime factorization.
- For $a, b \in \mathbb{Z}[i]$, there exists $g \in \mathbb{Z}[i]$ such that $na + mb \mid n, m \in \mathbb{Z}[i] = ng \mid n \in \mathbb{Z}[i]$. In particular, g is the greatest common divisor of a and b.

We acknowledge these properties. For more details, please refer to Gaussian integers - Wikipedia[1]

When a prime q of form $4n+3$ occurs an odd number of times in the prime factorization of R

Assume that there exists $w \in \mathbb{Z}[i]$ such that $w\overline{w} = R$. For any $x \in \mathbb{Z}[i]$, w is a multiple of x if and only if $\overline{w}$ is a multiple of $\overline{x}$. Therefore, when factoring w and $\overline{w}$, the prime factor q should appear the same number of times, and the prime factorization of R should have q occurring an even number of times. This contradicts the assumption that q occurs an odd number of times in the prime factorization of R, so $W(R) = \emptyset$, and the answer is inf.

From now on, it is assumed that R contains an even number of each $4n + 3$ type prime in its prime factorization.

Preparing

From now on, assume that R contains an even number of each prime of form $4n + 3$ in its prime factorization.

We define the following notations.

- For $w \in \mathbb{Z}[i]$ and $A \subset \mathbb{Z}[i]$, $wA := \{aw \mid a \in A\}$
- For $A, B \subset \mathbb{Z}[i]$, $A + B := \{a + b \mid a \in A, b \in B\}$
- For $A, B \subset \mathbb{Z}[i]$, $A \times B := \{ab \mid a \in A, b \in B\}$

[1] https://en.wikipedia.org/wiki/Gaussian_integer

- When we label the elements of $W(R)$ as $W(R) = w_1, w_2, \ldots, w_k$, let $S(R) = \left\{ \sum_{i=1}^{k} n_i w_i \;\middle|\; n_1, n_2, \ldots, n_k \in \mathbb{Z}[i] \right\} = w_1\mathbb{Z}[i] + w_2\mathbb{Z}[i] + \cdots + w_k\mathbb{Z}[i]$ be the set of vertices connected to vertex 0.

If the following shown, we can complete the proof by mathematical induction.

- $S(1) = \mathbb{Z}[i]$
- $S(2) = (1+i)\mathbb{Z}[i]$
- For a prime p of form $4n+1$, $S(p) = \mathbb{Z}[i]$
- For a prime q of form $4n+3$, $S(q^2) = q\mathbb{Z}[i]$
- S is multiplicative: $(S(A) = a\mathbb{Z}[i] \land S(B) = b\mathbb{Z}[i]) \implies S(AB) = ab\mathbb{Z}[i]$

We will show these in the following.

When $R = 1, 2$

As demonstrated in the samples, $S(1) = \mathbb{Z}[i]$ and $S(2) = (1+i)\mathbb{Z}[i]$.

When $R = p$, where p is a prime of form $4n+1$

It is known that primes of form $4n+1$ can be expressed (uniquely) as the sum of two squares.

Therefore, there exists $w \in \mathbb{Z}[i]$ such that $w\overline{w} = R$. Here, w is a Gaussian prime. (If w can be factored, then $\overline{w}$ can also be factored, thus R can be factored. This contradicts.)

By the properties acknowledged above, $S(R) = w\mathbb{Z}[i] + \overline{w}\mathbb{Z}[i] = \gcd(w, \overline{w})\mathbb{Z}[i]$. Since $w, \overline{w}$ are distinct primes (if they were the same, R would be even), $\gcd(w, \overline{w}) = 1$ (considering multiplying by identity elements as equivalent), and $S(R) = \mathbb{Z}[i]$.

When $R = q^2$, where q is a prime of form $4n+3$

When factorizing q^2 over Gaussian integers, it cannot be further factored beyond $q \times q$. Due to the uniqueness of prime factorization over Gaussian integers, for $w\overline{w} = R$, the only possibilities for $w \in \mathbb{Z}[i]$ are $w = q, iq, -q, -iq$. Therefore, $S(R) = q\mathbb{Z}[i]$.

S is multiplicative: $(S(A) = a\mathbb{Z}[i] \land S(B) = b\mathbb{Z}[i]) \implies S(AB) = ab\mathbb{Z}[i]$

By the uniqueness of prime factorization over Gaussian integers, $W(AB) = W(A) \times W(B)$ holds.

If we label $W(A) = \{a_1, a_2, \ldots\}$, $W(B) = \{b_1, b_2, \ldots\}$,

$$\begin{aligned}
S(AB) &= a_1b_1\mathbb{Z}[i] + a_1b_2\mathbb{Z}[i] + \cdots + a_2b_1\mathbb{Z}[i] + a_2b_2\mathbb{Z}[i] + \cdots \\
&= a_1(b_1\mathbb{Z}[i] + b_2\mathbb{Z}[i] + \cdots) + a_2(b_1\mathbb{Z}[i] + b_2\mathbb{Z}[i] + \cdots) + \cdots \\
&= a_1b\mathbb{Z}[i] + a_2b\mathbb{Z}[i] + \cdots \\
&= b(a_1\mathbb{Z}[i] + a_2\mathbb{Z}[i] + \cdots) \\
&= ab\mathbb{Z}[i].
\end{aligned}$$

We proved the answer for this problem.

Problem Tutorial: "N^a (log N)^b"

Observation

The condition that the limit $\lim_{N\to\infty} \frac{F(N)}{N^a(\log N)^b}$ converges to a finite value is similar to Landau's big O notation. In other words, the pair (a, b) intuitively represents "the rate of divergence to infinity."

For $N, \log N, \log(\log N)$, the following holds for any positive real number $c > 0$:

$$\lim_{N\to\infty} \frac{(\log N)^c}{N} = 0$$
$$\lim_{N\to\infty} \frac{(\log(\log N))^c}{\log N} = 0$$

While the proof are not easy, using these equations allows us to prove the following solution.

Solution

The states that should be considered as $F(N)$ can be expressed as tuples (a, b, ε), where a and b are non-negative integers, and ε is either 0 or 1. Intuitively, ε represents "the rate of divergence not reaching $\log(N)$ but diverging to infinity as $N \to \infty$." The solution to the problem is (a, b) if $\varepsilon = 0$, and $(a, b + 1)$ if $\varepsilon = 1$.

We define the order of states lexicographically: $(a_1, b_1, \varepsilon_1) \geq (a_2, b_2, \varepsilon_2)$ if and only if $a_1 > a_2$ or ($a_1 = a_2$ and $b_1 > b_2$) or ($a_1 = a_2$ and $b_1 = b_2$ and $\varepsilon_1 \geq \varepsilon_2$).

1. Firstly, for a positive integer m, the state of N^m is $(m, 0, 0)$.
2. Let the states of $F_1(N)$ and $F_2(N)$ be $(a_1, b_1, \varepsilon_1)$ and $(a_2, b_2, \varepsilon_2)$, respectively. The states of the sum and product of F_1 and F_2 are as follows:
   - The state of $F_1(N) \times F_2(N)$ is $(a_1 + a_2, b_1 + b_2, \varepsilon_1 \text{ OR } \varepsilon_2)$.
   - The state of $F_1(N) + F_2(N)$ is $\max\{(a_1, b_1, \varepsilon_1), (a_2, b_2, \varepsilon_2)\}$.
3. Let the state of $F(N)$ be (a, b, ε), then for a positive integer m, the state of $(\log(F(N)))^m$ is as follows:
   - If $a > 0$, then $(0, m, 0)$.
   - If $a = 0$, then $(0, 0, 1)$.

Therefore, by recursively parsing the string while performing these operations, we can solve this problem. Note that the values of a and b may not fit within a 32-bit integer type according to the constraints of the problem (they will fit within a 64-bit integer type instead).

Problem Tutorial: "Cola"

Bob's optimal strategy

The information known about Bob's P always follows the pattern described below:

There exists a unique integer $1 \leq a \leq N$ such that:

- For $i < a$, the value of P_i is known.

- For $i = a$, there are some values that P_i cannot be.
- For $a < i \leq N$, no information is known about P_i.

Therefore, Bob's optimal strategy utilizes the mentioned a in the following way:

- For $i < a$, set $Q_i = P_i$.
- For $i = a$, choose one possible value x for P_i and set $Q_i = x$.
- For $a < i \leq N$, arbitrarily determine Q_i.

The probability that the minimum i that becomes a in this strategy is b (where $0 \leq b < N - a$) is $\frac{1}{N+1-a}$ and is independent of a.

We define the polynomial $f(x)$ as follows.

- $[x^a]f(x)$ is the probability that Bob receives cola on the $(a+1)$-th question.

The following equation then holds.

$$f(x) = \prod_{i=1}^{N} \left(\frac{1}{i} \sum_{j=0}^{i-1} x^j \right) = \frac{1}{N!} \prod_{i=1}^{N} \frac{1-x^i}{1-x} = \frac{1}{N!} \frac{1}{(1-x)^N} \prod_{i=1}^{N} (1-x^i)$$

Computing by using formal power series

Now, we proceed with formal power series.

Using $\sum_{i=0}^{M-1} [x^i]f(x) = [x^{M-1}] \left(\frac{f(x)}{1-x} \right)$, it is sufficient to calculate $[x^i] \left(\frac{1}{(1-x)^{N+1}} \right)$ and $[x^i] \prod_{j=1}^{N} (1-x^j)$ for any $0 \leq i < M$.

$[x^i] \left(\frac{1}{(1-x)^{N+1}} \right) = \binom{N+i}{N}$, so with preprocessing in $O(N+M)$, it can be computed in $O(1)$.

Now, let's find $\prod_{j=1}^{N} (1-x^j)$.

For $0 \leq i \leq M \leq N$, the following holds:

$$[x^i] \prod_{j=1}^{N} (1-x^j) = [x^i] \prod_{j=1}^{\infty} (1-x^j)$$

Therefore, by using Euler's Pentagonal Number Theorem, $O(\sqrt{M})$ preprocessing can calculate $[x^i] \prod_{j=1}^{N} (1-x^j)$ in $O(1)$.

Problem Tutorial: "404 Chotto Found"

Let S be the Suffix Array and LCP Array of the concatenated strings $S_1, S_2, \ldots S_N$ using '#' (any character smaller than lowercase English letters in lexicographical order) as a separator.

Next, for each string, consider removing the common substrings found in multiple strings from the set of all substrings.

If a suffix is derived from S_i, and L is the maximum value of LCP for all suffixes except those derived from S_i, then to avoid common substrings with different strings, the end position should be chosen such that the length of the substring is greater than or equal to $L+1$.

To account for duplicate substrings within the same string, in case of the same substring, we count the later occurrence on the Suffix Array. Let R be the LCP with the next occurrence of the same substring-derived suffix on the SA. To avoid common substrings, the end position should be chosen such that the length of the substring is greater than or equal to $R+1$.

The count of substrings corresponding to a particular suffix is given by (length of the suffix) - $\max(R, L)$.

For each suffix corresponding to $S_1, S_2, \ldots, S_N$, sum up the counts calculated above.

Let $M = \sum_{i=1}^{N} |S_i|$, then with appropriate precomputation, L and R can be computed in $O(M)$ time.

The construction of Suffix Array and LCP Array can also be done in $O(M)$ time.

Problem Tutorial: "T Tile Placement Counting"

Firstly, H and W both being multiples of 4 is a necessary and sufficient condition for the existence of a tiling method (hereinafter referred to as "tiling"). Throughout the explanation, let H and W both be multiples of 4.

It is important to note that all the proofs for facts related to tiling used in this explanation are provided in the following two papers. Here, we acknowledge these facts and focus on the properties of tiling and specific calculation methods.

- D.W. Walkup, Covering a rectangle with T-tetrominoes, Amer. Math. Monthly 72 (1965) 986-988 [2]
- M. Korn and I. Pak, Tilings of rectangles with T-tetrominoes, Theor. Comp. Sci. 319 (2004), 3-27 [3]
  - Especially Sections 3. Tiling rectangles with T-tetrominoes and 4. Chain graphs

Properties of Tiling

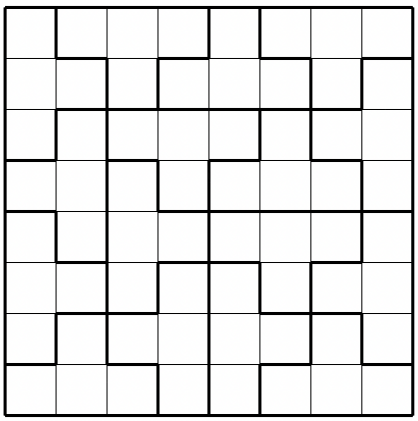

Рис. 2: Tiling example for $H = W = 8$.

In fact, the tiles do not have the following positional relationship.

Furthermore, the tiles, fitting together concavities and convexities, form columns, and these columns further create cycles. The tiles can be observed to form several cycles.

[2] https://www.jstor.org/stable/2313337
[3] https://doi.org/10.1016/j.tcs.2004.02.023

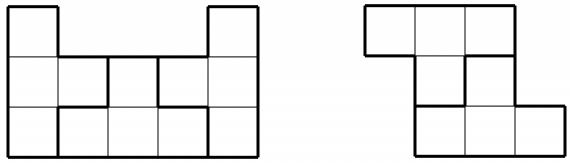

Рис. 3: Non-existent positional relationship between tiles.

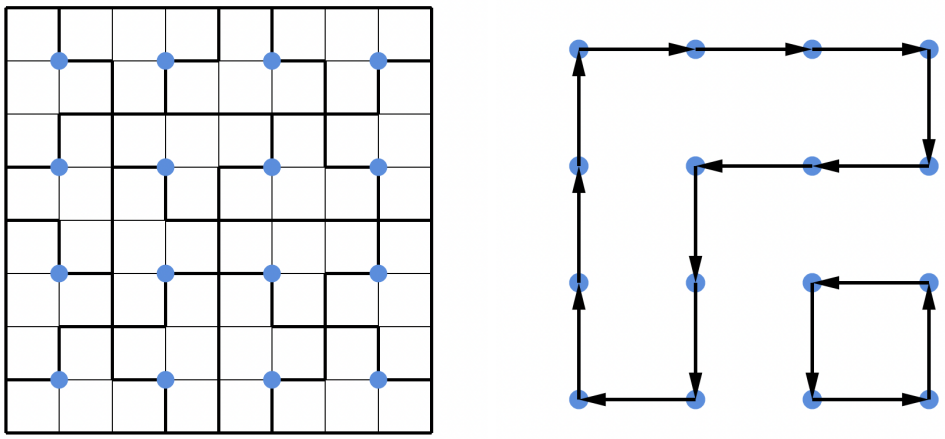

Рис. 4: Tiling cycles.

Now, let's consider the bottom right corner (the blue point in the figure) of a square with odd-row and odd-column indices, denoted as (i, j), where i and j are both odd.

The four squares around the blue point consist of three squares covered by the same tile, and the remaining one square covered by a different tile. Similarly, the four squares covered by the tile include three squares that are part of the four squares around another blue point, and the remaining one square is part of the four squares around a different blue point. Therefore, for a blue point u, let t be the tile that covers exactly one of the four squares around u, and let v be the blue point around whose four squares t covers three squares. We then consider a directed edge $u \to v$. The blue point v is uniquely determined for any blue point u. Moreover, the directed graph created in this way consists of vertices with in-degrees and out-degrees equal to 1, forming a set of directed cycles. We call this graph the **chain graph** of tiling. For any tiling, such a graph can be considered, and it is evident that the mapping from tiling to the chain graph is injective.

Properties of the Chain Graph

Let's summarize the characteristics of the chain graph. Firstly, all edges have a length of 2 squares. Furthermore, the graph consists of a set of cycles, and focusing on each cycle, the following conditions are satisfied:

- A change in direction towards the same direction can occur only when traversing an odd number of edges.
- A change in direction towards a different direction can occur only when traversing an even number of edges.

Graphs with these properties are all associated with tilings. Now, by selecting one cycle in the chain

graph and reversing its direction, we still get a chain graph. Therefore, for a chain graph with c cycles, it is sufficient to count the weighted number of chain graphs with a fixed orientation, where each cycle is assigned a weight of 2^c.

Up to this point, the problem of counting tilings has been reduced to the problem of counting chain graphs. By fixing the orientation of the edges and counting the number of ways to choose some edges on the next graph to make the in-degrees and out-degrees of all vertices equal to 1, the solution can be obtained. (This orientation reflects the constraints of direction changes.)

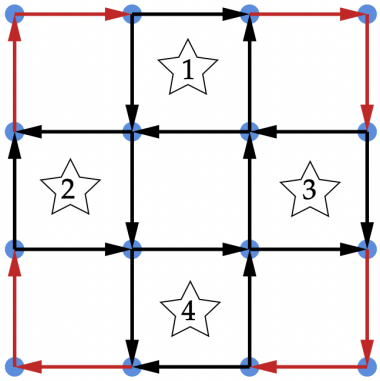

Рис. 5: Directed edges on a graph.

Note that the scale has been reduced from considering a grid of $H \times W$ squares to considering a grid of $H/2 \times W/2$ vertices.

In this graph, certain edges (depicted in red) are inevitably chosen due to the degree conditions, and for the remaining edges, choices are determined as follows:

- Independently choose either the left and right 2 edges or the upper and lower 2 edges of the marked sections.

For example, the tiling given earlier corresponds to choosing the upper and lower edges for marks 1 and 3, and the left and right edges for marks 2 and 4.

Counting

Dynamic programming is effective for counting such graphs. Consider selecting the marks from left to right and choosing edges one by one. Since the weight doubles each time a cycle is formed, it is necessary to keep track of how many cycles are formed. After selecting each column's mark, manage which point belongs to the same cycle, and the number of states can be expressed using Catalan numbers $C_n = \dfrac{(2n)!}{n!(n+1)!}$ as $C_{H/4}$. The figures include the corresponding bracket sequences for each state. Let C denote the number of states.

The transition of this dynamic programming does not depend on the indices (as the processing differs based on column parity, combining transitions for two columns), and it can be represented in a form suitable for matrix exponentiation. Therefore, for a fixed H, the sequence of answers has at most a C-degree linear recurrence. Considering this property, a solution that works faster than the $\Theta(C^3 \log W)$ matrix exponentiation-based approach can be used.

Firstly, calculate the first $2C$ terms by matrix multiplication in $\Theta(C^3)$ time. Then, use the Berlekamp–Massey algorithm to reconstruct the linear recurrence in $O(C^2)$ time and obtain the final answer using

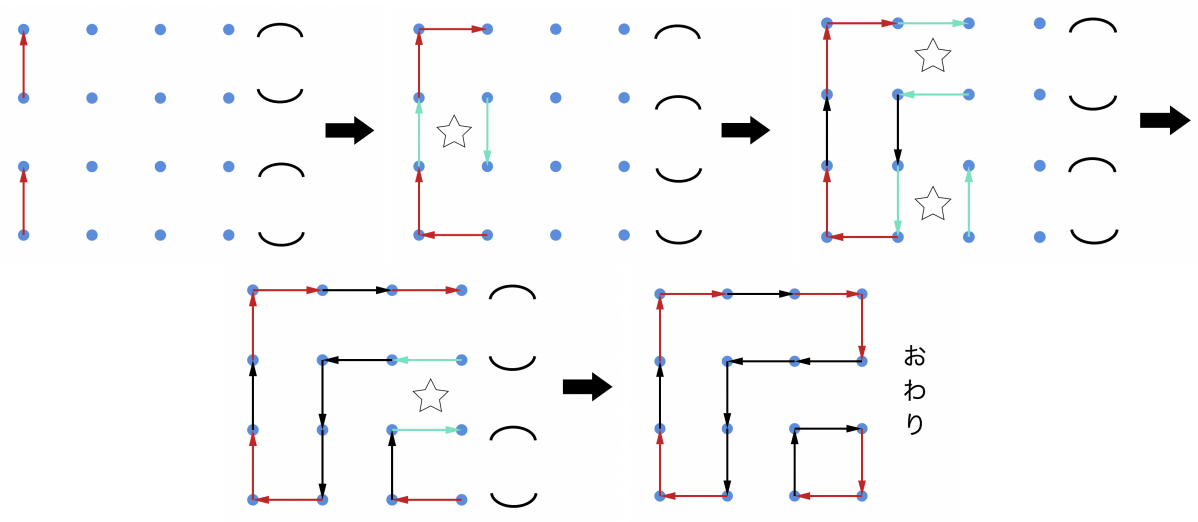

Рис. 6: Graphs used in counting.

the Bostan-Mori algorithm in $O(C \log C \log W)$ time. Note that embedding the two polynomials used in the Bostan-Mori algorithm for $H = 4, 8, \ldots, 28$ in advance can further reduce the execution time.

Problem Tutorial: "Set Construction"

Whether a given set satisfies the conditions can be determined with a time complexity of $O(M^2 \log M)$. Creating such a function allows us to verify whether the constructed set truly satisfies the conditions, and helps minimize unnecessary penalties.

However, note that submitting a solution that executes the $O(M^2 \log M)$ verification function might result in TLE.

Solution

Let's denote any set that satisfies the conditions for N, M as $f(N, M)$. This problem can be solved by using recursion to repeatedly reduce it to smaller cases, like several other construction problems.

When $M \leq N(N-1)/2$

We can reduce it to the problem of finding $f(N-1, M)$.

Let $A = f(N-1, M)$, and define

$$B = \{2a + (a\%2) \mid a \in A\}.$$

The set B satisfies the conditions, where $a\%2$ denotes the remainder when a is divided by 2.

When $N \geq 3$ and M is even

We can reduce it to the problem of finding $f(N-1, M/2)$.

Let $A = f(N-1, M/2)$ and $B = \{1\}$. By like taking the Cartesian product of the sets, define

$$C = \{2a + b \mid a \in A, b \in B\}.$$

The set C satisfies the conditions. The problem is reduced to finding $f(N-1, M/2)$.

To satisfy $M/2 \leq (N-1)N/2$, since $M \leq N(N+1)/2$, we need $N \geq 3$.

When $N \geq 6$ and M is odd

We can reduce it to the problem of finding $f(N-1, M-1)$ and then finding $f(N-2, \frac{M-1}{2})$.

Let $A = f(N-1, M-1)$. Define

$$C = \{2a+1 \mid a \in A, b \in B\} \cup \{0\}$$

, and the set C satisfies the conditions. The problem is reduced to finding $f(N-2, \frac{M-1}{2})$.

To satisfy $(M-1)/2 \leq (N-2)(N-1)/2$, we need $N \geq 6$.

When N is small

The missing case is $N \leq 5$.

The most challenging case is $f(5, 15)$. We can apply brute-force search to construct this case. Without brute-force search, we can construct as follows:

Focus on $15 = 3 \times 5$. Let $A = f(2, 3), B = f(3, 5)$. By like taking the Cartesian product of the sets, define

$$C = \{8a + b \mid a \in A, b \in B\}.$$

The set C satisfies the conditions.

Story

This problem is related to finite topological spaces, and this problem is equivalent to "Find a topological space on $0, 1, \cdots, N-1$ with M elements in its open set system."

Furthermore, it is possible to solve the decision problem in $O(N^2 M/w)$ time (w being the word size).

Problem Tutorial: "Dense Planting"

For any graph G, we can obtain the number of spanning trees in G by computing any first minor of the Laplacian matrix G by Kirchhoff's Theorem.

So let us consider about this graph.

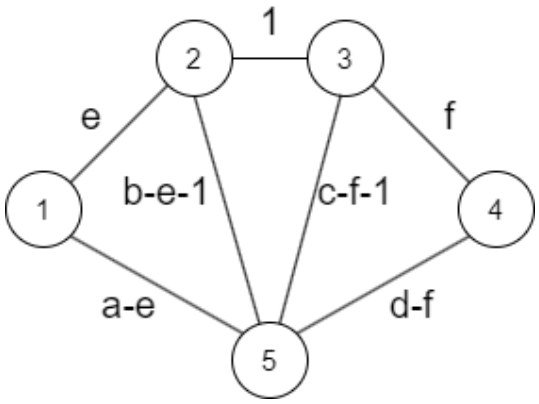

Рис. 7: a graph that have $(ab - e^2)(cd - f^2) - ad$ spanning trees.

the $(5, 5)$-minor of the Laplacian matrix of this graph is

$$\begin{pmatrix} a & -e & 0 & 0 \\ -e & b & -1 & 0 \\ 0 & -1 & c & -f \\ 0 & 0 & -f & d \end{pmatrix}$$

, so the number of spanning trees is $(ab-e^2)(cd-f^2)-ad$. and the number of edges is $a+b+c+d-e-f-1$.

Now all we need to do is to find a, b, c, d, e, f satisfy

- $K + ad = (ab - e^2)(cd - f^2)$
- $a \geq e \geq 0$
- $b \geq e + 1$
- $d \geq f \geq 0$
- $c \geq f + 1$
- $a + b + c + d - e - f \leq 1001$

.

You can find e, f that meet conditions by searching $a, b, c, d = K^{\frac{1}{4}} \pm 40$.

Also, by precalculating pairs of y, z that satisfy $xy - z^2 = P$ for $x \simeq K^{\frac{1}{4}}, P \simeq K^{\frac{1}{2}}$, you can quickly find b, c, e, f that satisfy the condition by searching for divisors P of $K + ad$ for$a, d \simeq K^{\frac{1}{4}}$

Problem Tutorial: "Next TTPC 3"

The name of the programming contest, denoted as T_x, repeats in a period of $\mathrm{lcm}(|S_1|, |S_2|, |S_3|, |S_4|)$. Denote C as the occurrences of T_x being `TTPC` in one period, and x as the $((N-1) \mod C) + 1$-th occurrence of T_x being `TTPC`. We can solve this problem by finding C and x. Therefore, we can assume $N \leq C$.

As a naive solution, enumerating T_x for one period is a possible method. However, in the constraints, it will be $\mathrm{lcm}(|S_1|, |S_2|, |S_3|, |S_4|) \leq 10^{12}$, so this solution is not suitable.

We imagine creating `TT` with strings S_1 and S_2, creating `PC` with strings S_3 and S_4, and combining these two to form `TTPC`. Let $M_1 = \mathrm{lcm}(|S_1|, |S_2|)$ and $M_2 = \mathrm{lcm}(|S_3|, |S_4|)$. Then, we can consider the periods of the first two characters and the last two characters of T_x are M_1 and M_2, respectively. Since $M_1, M_2 \leq 10^6$, we can enumerate the positions where S_1 and S_2 form the desired string in one period for both.

Let $k_1 < M_1$ be a non-negative integer such that on the $(k_1 + 1)$-th year the first two characters are `TT`, and let $k_2 < M_2$ be a non-negative integer such that on the $(k_2 + 1)$-th year the last two characters are `PC`. The non-negative integer x where the first two characters are `TT` can be expressed as $tM_1 + k_1 + 1$ using a non-negative integer t. Additionally, to have the last two characters become `PC`, we need $tM_1 + k_1 \equiv k_2 \pmod{M_2}$.

Here, for a fixed t, the number of pairs of k_1, k_2 satisfying the condition is given by

$$tM_1 \equiv -k_1 + k_2 \pmod{M_2}.$$

This represents the number of occurrences of `TTPC` in $N = tM_1, tM_1 + 1, \ldots, tM_1 + M_1 - 1$.

We can determine the number of pairs (k_1, k_2) satisfying $-k_1 + k_2 \equiv i \pmod{M_2}$ for $i = 0, 1, \ldots, M_2 - 1$ in a computational complexity of $O(M_2 \log(M_2))$. This can be achieved through convolution using Fast Fourier Transform (FFT).

By iterating t from 0, you can search for the value of t where the count of `TTPC` occurrences is N or more. After that, you can perform a brute-force search over $tM_1, tM_1 + 1, \ldots, tM_1 + M_1 - 1$ to find the N-th occurrence of `TTPC`.

The time complexity is $O(M_1 + M_2 \log(M_2))$.

Problem Tutorial: "Sum is Integer"

Decimal part of cumulative sum

Consider the cumulative sum $s_i = \sum_{j=1}^{i} \frac{p_i}{q_i}$ for $i = 0, 1, \ldots, N$, where $s_0 = 0$. This leads to the following problem:

> Count the number of pairs of integers (l, r) satisfying $0 \leq l < r \leq N$ such that $s_r - s_l$ is an integer.

The condition that the difference of two real numbers is an integer is equivalent to their decimal parts being equal. However, note that the decimal part of a real number x is defined as $x - \lfloor x \rfloor$.

Using this, we can solve the problem:

> Classify given real numbers based on their decimal parts.

Hashing using modulo

Let's use hashing. Since the values involved are rational numbers, we consider managing the decimal part using mod$M$. Here, M is a very large positive integer.

Calculating s_i modulo M is easy, but it contains information about not only the decimal part but also the integer part. Therefore, we perform calculations with floating-point numbers for the integer part and obtain the decimal part of s_i modulo M by subtracting its integer part.

The following points need consideration:

- How large should M be chosen to ensure that the decimal parts are likely to be equal modulo M due to the equality of the decimal parts modulo M?
- Can the integer part of the cumulative sum be accurately calculated with floating-point numbers?

Let's consider the former. If M is randomly chosen, the decimal parts will be distributed randomly modulo mod$M$. For all pairs of the $N + 1$ cumulative sums, if the decimal parts are different, then their decimal parts modulo mod$M$ must also be different. Therefore, by choosing M to be of the order of 10^{18}, we can estimate that it is sufficient to ensure a high probability of correctness. Thus, preparing two random prime numbers of the order of 10^9 as M will suffice for hashing the pairs of cumulative sums.

Bonus: If the M that the solution uses is known, you can create cases where the solution fails.

Now, consider the latter. Calculating the integer part of the cumulative sum with floating-point numbers may introduce errors and cause issues. However, by adding an appropriate constant c to the cumulative sum in advance, all integer parts of the cumulative sum can be accurately calculated with floating-point numbers. If errors seem to affect the calculation of the integer part when computing the cumulative sum, set c to avoid the issue. Although the value of cumulative sum modulo M may seem to change, it changes by the same amount, so there is no problem.

Bonus: The decimal part of the cumulative sum can be made as small as 10^{-40000}.

By managing the hash with an associative array, the problem can be solved in approximately $O(N \log N)$ time complexity.

Problem Tutorial: "Bracket Sequestion"

Counting Parentheses Sequences

The existence of a valid string is equivalent to replacing the '?' with '(' and ')' such that the number of occurrences of each is the same, resulting in a valid parentheses sequence.

```
(?()(??)??
 (   () ))
(((()(())))
```

Now, let's consider the inverse operation. For a parentheses sequence, we perform one of the following operations in order on the $1, 2, \ldots, 2N$ characters:

- Operation1 Replace '(' with '?'
- Operation2 Replace ')' with '?'
- Operation3 Do nothing

However, if Operation 2 is performed, Operation 1 cannot be executed.

The goal is to find the sum of the number of ways to replace characters for all parentheses sequences.

Let $C(x)$ be the generating function for Catalan numbers.

The number of ways to replace without performing Operation 1 for all strings is $2^N([x^N]C(x))$.

If Operation 1 is performed last, and considering the number of occurrences of '(' before the last ')' in the original parentheses sequence is greater by j $(1 \leq j \leq N)$, the number of ways for the parentheses sequence and replacement is $2^{N+j-1}([x^{N-j}]C^{2j+1}(x))$.

Here's an explanation of why this formula holds:

Let k be the number of ')' encountered before the last Operation 1. Up to the last Operation 1, the number of '(' encountered is $j+k$, and the number of subsequent ')' is $N-k$.

The total $N+k$ characters can be replaced with '?', and since one character is already determined to be replaced with '?', there are 2^{N+j-1} ways to perform replacements.

For the term $C^{2j+1}(x)$, it accounts for the fact that before the last Operation 1, there are $j-1$ more '(' than ')' to the left, and after that, there are j more ')' than '(' to the right.

The sequence is P-recursive

The sequence of Catalan numbers is P-recursive. Surprisingly, the sequence of answers is also P-recursive. Let's demonstrate this below.

Let a_i be the answer when $N = i$ $(a_0 = 1)$, $b_i = \frac{a_i}{2^i}$, $c_n = [x^n]C(x)$, and $f(x) = \sum b_i x^i$. According to the previous analysis, the following relation holds:

$$f = C + \frac{C^3 x}{1 - 2xC^2}$$

By using $xC^2 = C - 1$ or $C = \frac{1-\sqrt{1-4x}}{2x}$ in this equation, we obtain:

$$(9x-2)f = C(6x-1) - 1$$

By focusing on the terms x^n and x^{n+1} on both sides of this equation and further using $(4n+2)c_n = (n+2)c_{n+1}$, we get:

$$(9b_n - 2b_{n+1})\left(\frac{4n+2}{n+2}\right)\left(6 - \frac{4n+6}{n+3}\right) = (9b_{n+1} - 2b_{n+2})\left(6 - \frac{4n+6}{n+3}\right)$$

Expanding this equation and modifying it according to the formula for a_n, we obtain:

$$a_n(72n^2 + 468n + 216) + a_{n+1}(-17n^2 - 124n - 159) + a_{n+2}(n^2 + 8n + 15) = 0$$

Therefore, it has been demonstrated that the sequence $(a_0, a_1, \dots)$ is P-recursive.

The algorithm for computing the N -th term of a P-recursive sequence in $O(\sqrt{N}\log N)$ time is well known.

Problem Tutorial: "2D Parentheses"

Consider a graph consisting of N vertices corresponding to the N open parentheses and N vertices corresponding to the N closing parentheses. Add an edge between the i-th open parenthesis and the j-th closing parenthesis if $x_{1,i} < x_{2,j}$ and $y_{1,i} < y_{2,j}$.

To arrange N rectangles on the plane, it is necessary for this graph to have a perfect matching. If a perfect matching does not exist, the answer is `No`.

The possibility of arranging N rectangles on the plane in a way that satisfies the condition (hereinafter referred to as the condition) that the common area of any two different rectangles is either 0 or one is completely contained within the other depends on constructing a perfect matching.

The following greedy algorithm constructs a maximum matching, and if this algorithm constructs a perfect matching, it is the unique perfect matching that satisfies the condition.

- Sort the closing parentheses in ascending order of x, and if x coordinates are the same, sort them in ascending order of y.
- Match each open parenthesis with the closing parenthesis that can be matched. If there are multiple candidates, choose the one with the maximum y coordinate, and if the y coordinates are also the same, choose the one with the maximum x coordinate.

Proof of being the unique perfect matching

Assume there exists another perfect matching M_2 that can arrange N rectangles on the plane satisfying the condition, in addition to the perfect matching M_1 constructed by the above algorithm.

Then, there exists an open parenthesis i that satisfies the following condition:

- Let j_1 be the closing parenthesis matched with i in matching M_1, and let j_2 be the closing parenthesis matched with i in matching M_2.
- Then, $j_1 \neq j_2$, and either $x_{2,j_1} \leq x_{2,j_2}$ or $y_{2,j_1} \leq y_{2,j_2}$.

In that case, the open parenthesis matched with the closing parenthesis j_1 is outside the rectangle formed by i and j_2. This contradicts the fact that M_2 arranges N rectangles on the plane satisfying the condition.

Therefore, it has been proven that the perfect matching constructed by the above algorithm is the unique perfect matching that can arrange N rectangles on the plane satisfying the condition.

Solution

First, construct the maximum matching using the greedy algorithm and check if it is a perfect matching. If it is a perfect matching, check if it satisfies the condition. If it does, output `Yes`. If it doesn't, output `No`. The overall complexity is $O(N \log N)$ since the greedy algorithm can be implemented using `std::set`, and the condition check can be done using `std::set` or `std::multiset` or with a segment tree.

Problem Tutorial: "Bridge Elimination"

The score is defined as the "product of the sums of integers written on the vertices for each connected component of G."In other words, it is the "sum of the product of integers written on selected vertices from each connected component of G, considering all possible selections."

Therefore, if we denote $p(i)$ as "The total product sum of values for all cases of selecting i elements from $A_1, A_2, \ldots, A_N$,"and $q(i)$ as "The number of simple connected undirected graphs with i selected vertices, such that the number of connected components in G is i, and the chosen i vertices are in different connected components in G,"then the solution to the problem is given by:

$$\sum_{i=1}^{N} p(i) \times q(i)$$

Calculation of $p(i)$

$$p(i) = [x^i] \prod_{j=1}^{N} (1 + A_j x)$$

This allows us to calculate $p(i)$ using methods such as $O(N^2)$ or $O(N(\log N)^2)$.

Calculation of $q(i)$

Each connected component obtained by removing bridges from a simple connected undirected graph forms a biconnected component, which, due to the properties of biconnected components, results in a tree when aggregated into a single vertex.

Now, let's consider the following problem:

> For M simple connected undirected graphs with labeled vertices, each without bridges, what is the number of graphs obtained by adding $M - 1$ edges to make them connected?

The answer to this problem, when the number of vertices in the i-th graph without bridges is denoted as B_i, and N is the total number of vertices ($N = \sum_{i=1}^{M} B_i$), is given by:

$$\left(\prod_{i=1}^{M} B_i \right) N^{M-2}$$

This result can be derived using the Prüfer code.

Let B_i be the number of vertices in the i-th group, and $r(i)$ be the number of simple connected undirected graphs with i vertices and no bridges. Define $dp[i][j]$ as the "sum of $\prod_{k=1}^{j} B_k r(B_k)$ for all possible ways

of partitioning i vertices into non-empty j groups, each forming a simple connected undirected graph without bridges, and the selected j vertices are in different groups in G."Then, $q(i)$ is given by:

$$q(i) = dp[N][i] \times N^{i-2}$$

This dynamic programming approach has the following transitions:

$$dp[0][0] = 1$$

$$dp[i][j] = \sum_{k=1}^{i-j+1} \binom{i-j}{k-1} \times dp[i-k][j-1] \times k \times r(k)$$

This dynamic programming solution has a state space of $O(N^2)$ and transitions in $O(N)$, resulting in an overall complexity of $O(N^3)$.

Furthermore, $r(i)$ is obtained by subtracting the number of graphs with more than one connected component in G from the total number of graphs with i vertices. Let $dp2[i][j]$ be the sum of $\prod_{k=1}^{j} B_k r(B_k)$ for all possible ways of partitioning i vertices i vertices into non-empty j groups, each forming a simple connected undirected graph without bridges."Then, $dp2$ has the following transitions:

$$dp2[0][0] = 1$$

$$dp2[i][j] = \sum_{k=1}^{i} \binom{i-1}{k-1} \times dp2[i-k][j-1] \times k \times r(k)$$

Thus, $r(i)$ can be expressed as:

$$r(i) = s(i) - \sum_{j=2}^{i} dp2[i][j] \times i^{j-2}$$

Using $dp2$ with $O(N^3)$ complexity, $r(i)$ can be computed. Finally, $s(i)$ can be obtained using the inclusion-exclusion principle. With these calculations, $q(i)$ can be determined in $O(N^3)$.

Summary

By calculating $p(i)$ in $O(N^2)$ and $q(i)$ in $O(N^3)$, the overall solution can be achieved in $O(N^3)$.