Problem A. Kendama Challenge

Consider the probability that the condition is satisfied for the first time at the X-th trial.

The condition is satisfied for the first time at the X-th trial if and only if all of the following hold:

- The condition has not been satisfied up to the $(X-K-1)$-th trial
- The $(X-K)$-th trial is a failure
- The following K trials are all consecutive successes

By precomputing cumulative products of probabilities, each of these terms can be evaluated in $O(1)$ time.

Problem B. Cat Cut

For simplicity, for a string T, let $T[l, r)$ denote the contiguous substring consisting of the characters from the l-th to the $(r-1)$-th positions (0-indexed).

First, consider the structure of the strings X that can be constructed.

Instead of extracting a contiguous substring of length M from Y at each operation, we may assume that we extract a prefix of length at least M, and after all operations are finished, we take the last M characters. This does not change the set of strings that can be constructed.

Thus, we consider the process as follows: initially $X = S_1$, and in the i-th operation, we append a prefix of S_{i+1} (possibly empty) to the end of X. After $N-1$ operations, we take the last M characters of X.

Before taking the last M characters, the structure is

$$X = S_1 + \mathrm{prefix}(S_2) + \cdots + \mathrm{prefix}(S_N).$$

After extracting the last M characters, the final structure becomes one of the following:

- $\mathrm{suffix}(S_1) + \mathrm{prefix}(S_2) + \cdots + \mathrm{prefix}(S_N)$
- $\mathrm{substring}(S_i) + \mathrm{prefix}(S_{i+1}) + \cdots + \mathrm{prefix}(S_N)$ $(i \geq 2)$

1 Partial Solution

First, consider the part $\mathrm{prefix}(S_i) + \cdots + \mathrm{prefix}(S_N)$.

Let

$$dp[i][j] = \text{the lexicographically smallest string of length } j \text{ among } \mathrm{prefix}(S_i) + \cdots + \mathrm{prefix}(S_N).$$

This can be computed in $O(NM^3)$ time overall by naively checking all prefixes of S_i.

Then, by examining all suffixes of S_1 and all substrings of S_i $(i \geq 2)$, the answer can also be obtained in $O(NM^3)$ time overall.

2 Full Solution

First, for the part $\mathrm{prefix}(S_i) + \cdots + \mathrm{prefix}(S_N)$:

In the partial solution, we considered all lengths j, but it suffices to consider only strings of length M.

This is because for $s < t$, $dp[i][s]$ is always a prefix of $dp[i][t]$.

Indeed, if $dp[i][s]$ exists, we can extend it by adding more characters to obtain a string of length t whose prefix is $dp[i][s]$. Conversely, if $dp[i][t]$ exists, we can truncate it to obtain a prefix of length s.

Thus, define

$$dp[i] = \text{the lexicographically smallest string of length } M \text{ among } \text{prefix}(S_i) + \cdots + \text{prefix}(S_N).$$

Let $S = S_i$, $T = dp[i+1]$. Then

$$dp[i] = \min_j \big(S[0, j) + T[0, M - j)\big).$$

To compare $S[0, s) + T[0, M - s)$ and $S[0, t) + T[0, M - t)$ with $s < t$, it reduces to comparing

$$T[0, M - s) \quad \text{and} \quad S[s, t) + T[0, M - t).$$

This can be done by computing $\text{LCP}(S[s, M), T)$ and $\text{LCP}(T[t - s, M), T)$, which can be efficiently obtained using the Z-algorithm on $T + S$.

Next, consider adding substring(S_i) (for $i \geq 2$).

Let Z be the lexicographically smallest suffix of S_i (if one is a prefix of the other, take the longer one). Then it suffices to consider only strings of the form

$$Z[0, k) + dp[i+1][0, M - k).$$

This can be computed similarly to the prefix DP.

(Proof that considering only Z suffices)

Let $Z = S_i[j, M)$. For

$$S_i[L, R) + dp[i+1][0, M - (R - L)) \leq S_i[j, M) + dp[i+1][0, M - j),$$

one of the following must hold:

1. $S_i[L, R)$ is a prefix of $Z = S_i[j, M)$
2. $Z = S_i[j, M)$ is a prefix of $S_i[L, R)$
3. Neither (1) nor (2) holds, and $S_i[L, R) < S_i[j, M)$

However, in cases (2) and (3), $S_i[L, M)$ should have been chosen as Z, which is a contradiction.

This Z can be found in $O(M)$ time using a suffix array or Lyndon factorization.

For the suffix of S_1, it suffices to find the lexicographically smallest substring of length M in $S_1 + dp[2]$, which can also be done in $O(M)$ time using a suffix array or Lyndon factorization.

Thus, the entire solution runs in $O(NM)$ time.

Problem C. Partition AND/OR Aggregation

How to compute the maximum value

Case $K < M$

If we choose $B_1, \ldots, B_{M-1}$ to all have length 1, we can achieve the maximum value 1.

Case $K = M$

We reduce the problem to a decision problem via binary search: "Is it possible to partition so that the score of every subsequence is at least X?"

Since the score of a subsequence is monotonically non-increasing as the sequence is extended, we can greedily process from the left, repeatedly extending each segment as far as possible while maintaining that its score is at least X.

If the number of resulting subsequences is at most M, then we can further split some segments arbitrarily to obtain exactly M subsequences without decreasing the score, so the condition is satisfied.

If the number of subsequences exceeds M, then it is impossible to satisfy the condition.

This check can be done in $O(N)$ time.

—

How to compute the minimum value

We reduce the problem via binary search to the decision problem: "Does there exist a partition such that the K-th largest value (in descending order) is at most X?"

This can be rephrased as: "Can we partition into M subsequences such that at least $M-K+1$ subsequences have score at most X?"

For each i ($1 \leq i \leq N$), let R_i be the minimum right endpoint such that $S((A_i, \ldots, A_R)) \leq X$.

Since R_i is monotonically increasing with respect to i, all R_i can be computed in $O(N)$ time using the two-pointer (sliding window) technique.

The problem can then be formulated as follows:

- Select $M-K+1$ pairwise disjoint intervals from $[i, R_i]$. Can the remaining elements be partitioned appropriately so that the total number of segments becomes M?

The elements not covered by the selected $M-K+1$ intervals must be partitioned into $K-1$ non-empty segments. Therefore, if the total number of elements not included in the selected intervals is at least $K-1$, we can partition them appropriately and, if necessary, extend the selected intervals to fill gaps, thus constructing a valid partition into M segments.

Let $f(T)$ be the minimum total length $\sum(R_i - i + 1)$ when selecting T pairwise disjoint intervals from $[i, R_i]$ ($i = 1, \ldots, N$).

To satisfy the condition, it suffices to check whether

$$N - f(M-K+1) \geq K-1.$$

Here, f is a convex function (i.e., it satisfies $2f(T) \leq f(T-1) + f(T+1)$).

Proof:

Let $I_i = [i, R_i]$.

Suppose $f(T-1) = S$, $f(T+1) = S'$, and let $I_{i_1}, \ldots, I_{i_{T-1}}$ ($i_1 < \cdots < i_{T-1}$) and $I_{i'_1}, \ldots, I_{i'_{T+1}}$ ($i'_1 < \cdots < i'_{T+1}$) be optimal selections achieving these values.

Let j be the smallest index such that $i_j \leq i'_{j+1}$ (if no such j exists, let $j = T$). Then we have

$$i'_j < i_{j-1} < i_j \leq i'_{j+1}.$$

From this, we can construct two different selections of T disjoint intervals:

- $I_{i_1}, \ldots, I_{i_{j-1}}, I_{i'_{j+1}}, \ldots, I_{i'_{T+1}}$
- $I_{i'_1}, \ldots, I_{i'_j}, I_{i_j}, \ldots, I_{i_{T-1}}$

The total length of intervals in each of these selections is at least $f(T)$, and their sum equals $f(T-1) + f(T+1)$, hence

$$2f(T) \leq f(T-1) + f(T+1).$$

Since f is convex, we can apply Alien DP (Lagrangian relaxation).

For a real number λ, we find the T that minimizes $f(T) - \lambda T$. By performing binary search on λ such that the optimal T becomes at most $M - K + 1$, we can compute $f(M - K + 1)$.

The minimization of $f(T) - \lambda T$ can be formulated as a shortest path problem on the following graph, which can be solved in $O(N)$ time:

- Vertices are 1 through $N + 1$
- Add an edge from i to $R_i + 1$ with cost $R_i + 1 - i - \lambda$
- Add an edge from i to $i + 1$ with cost 0

The optimal T changes only when $\lambda = f(T + 1) - f(T)$, so it suffices to consider $\lambda = 1, \ldots, N$, and this subproblem can be solved in $O(N \log N)$ time.

—

Binary search over rational numbers

Since this problem requires computing the answer as a rational number, ordinary binary search over real numbers may suffer from precision issues.

Possible solutions include performing binary search on the Stern–Brocot tree, or binary searching over values of the form $\frac{k}{2^{60}}$ followed by rational reconstruction. Here, we introduce another approach.

In this problem, it is possible to enumerate all possible values of the score of contiguous subsequences in advance.

Let w be a non-negative integer such that $0 \leq A_i < 2^w$. Then the following holds:

- The number of possible scores of contiguous subsequences is at most $(2w + 1)N$

Indeed, if we fix the right endpoint i and extend the left endpoint leftward, the values of bitwise AND and OR change only when the popcount changes (i.e., when bits are set or cleared). Each of these events occurs at most w times.

Therefore, for each i, the number of possible $(\mathrm{AND}, \mathrm{OR})$ pairs is at most $2w + 1$.

By computing these for all $i = 1, \ldots, N$, we can reduce the candidates for the score to $O(wN)$ values. By enumerating and sorting them, we can perform binary search over indices.

Thus, after enumerating and sorting all possible scores in $O(wN \log(wN))$, we can compute the maximum value in $O(N \log(wN))$ time and the minimum value in $O(N \log N \log(wN))$ time.

Problem D. Campaign Speech

Let S be the perimeter of the polygon, and let $L_{\max}$ be the maximum distance between two consecutive speech locations along the perimeter. Then the desired answer is $S - L_{\max}$.

Since the edges of the simple polygon are given in order, we can compute $L_{\max}$ by traversing the edges in sequence and enumerating the speech locations on each edge in order.

3 Partial Solution

(constraint: speech locations exist only at the vertices of the polygon)

Since speech locations exist only at the vertices of the polygon, if we can determine whether each vertex is a speech location, we can compute the distances by updating the distance from the previously visited speech location.

This can be done efficiently by storing the coordinates of the speech locations in a C++ `set<pair<int,int>>` or a Python `set`.

The time complexity is $O((N+M)\log M)$ or $O(N+M)$.

4 Full Solution

We need to enumerate the speech locations on an edge $(x1, y1) \to (x2, y2)$ in order of increasing distance from $(x1, y1)$.

From the constraints, either $x1 = x2$ or $y1 = y2$ holds. Consider the case $x1 = x2$ (the case $y1 = y2$ is similar).

If we maintain a sorted array of all y-coordinates of speech locations with x-coordinate equal to $x1$, then we can use binary search to find all coordinates satisfying $y1 \le q_i \le y2$.

Since the coordinates range from -10^5 to 10^5, it is sufficiently efficient to maintain arrays for all possible x and y coordinates.

Such arrays can be constructed in preprocessing time $O(\max(|p_i|, |q_i|) + M\log M)$, so the overall time complexity is $O(\max(|p_i|, |q_i|) + (N+M)\log M)$, which is sufficiently fast.

Problem E. Ball Dumping Golf

Idea

Consider a directed graph with N vertices, where each box is regarded as a vertex. For each ball, add an edge from "the index of the box containing that ball"to "the number of that ball"(self-loops and multiple edges are allowed). Minimizing the number of operations is equivalent to asking: "How many **trails** are needed to cover all edges of this graph?"(It is allowed to visit the same vertex multiple times, but not the same edge.)

Also, since every vertex has both in-degree and out-degree equal to M, each connected component is an **Eulerian graph**, and thus can be traversed in a single trail (note that the entire graph is not necessarily Eulerian as a whole). Therefore, the number of operations is at most the number of connected components of this graph. Conversely, it is clearly impossible to achieve fewer operations than the number of connected components. Hence, the number of operations equals the number of connected components of this graph.

From the above, the desired value equals the expected number of connected components of this graph.

How to Compute

Define sequences a, b, c as follows (with M fixed and edge labels present):

- $a_i =$ (the number of such graphs when $N = i$)
- $b_i =$ (the number of such graphs when $N = i$ that are **connected**)
- $c_i =$ (the total number of connected components among all such graphs when $N = i$)

The desired value is $\frac{c_i}{a_i}$.

The sequence a corresponds to "the number of ways to assign balls into boxes."Each term can be computed directly using a multinomial coefficient:

$$a_i = \frac{(iM)!}{(M!)^i}.$$

From this, we compute b and c.

Let the exponential generating functions of a, b, c be $A(x), B(x), C(x)$, respectively.

Between A and B, we have the relation

$$A = 1 + B + \frac{B^2}{2!} + \frac{B^3}{3!} + \frac{B^4}{4!} + \cdots = \exp(B),$$

so we obtain $B = \log A$.

Using B, we can express C as

$$C = B + 2\left(\frac{B^2}{2!}\right) + 3\left(\frac{B^3}{3!}\right) + 4\left(\frac{B^4}{4!}\right) + \cdots = B\exp(B),$$

which can also be computed. (In fact, substituting $B = \log A$ into the above gives $C = A\log A$, so in implementation, one can compute C directly without explicitly computing B.)

Thus, the answer $\left(\frac{[x^N]C(x)}{[x^N]A(x)}\right)$ can be computed in $O(NM + N\log N)$ time.

Problem F. 1e16 Cities

Let $\gcd(i, j) = g$, and write $i = gx$, $j = gy$. Then the condition becomes

$$gxy = Ag + B, \quad \gcd(x, y) = 1.$$

In particular, we only need to consider g that are positive divisors of B.

All triples of positive integers (g, x, y) such that g is a positive divisor of B and $xy = A + B/g$ can be enumerated in $O(\sqrt{B} + d(B)\sqrt{A + B})$ time.

Among these, for those satisfying $\gcd(x, y) = 1$ and $gx, gy \le 10^{16}$, we add an edge (gx, gy).

Since the number of positive divisors of any integer up to 2×10^8 is at most 1000, the total number of enumerated triples is at most 10^6, so this can be done sufficiently fast.

By enumerating all edges in advance, each query can then be processed efficiently using Union-Find or similar data structures.

Problem G. The Symbolic Tree

In the following explanation, vertex indices are given in 0-indexed form.

Solution for $K = N$

Let $dp_{i,j}$:= the number of ways to assign integers in the subtree rooted at vertex i such that the integer written at vertex i is at most j.

Then the following recurrence holds:

$$dp_{i,j} = dp_{i,j-1} + \prod_{k \in \text{children}} dp_{k,j}.$$

(However, when $j = 0$, we have $dp_{ij} = 1$.)

The number of states is $O(NK)$, and the transitions occur only $(N-1)K$ times, so the total time complexity is $O(NK)$.

The desired answer is

$$dp_{0,K} - dp_{0,K-1}.$$

Solution for $K \le 10^9$

Summarizing the DP recurrence, we have

$$dp_{i,j} = \sum_{l=0}^{j} \prod_{k \in \text{children}} dp_{k,l}.$$

Here, regardless of the magnitude of j, there exist polynomials $f_0, f_1, ..., f_{N-1}$ such that $f_i(j) = dp_{i,j}$, and the degree of f_i is the size of the subtree rooted at vertex i.

(Proof: In the case of a leaf, the degree is 1, so the claim holds. Otherwise, inductively, the product of the children's polynomials has degree equal to subtree size -1, and taking its prefix sum increases the degree to the subtree size.)

Focusing on vertex 0, the degree of f_0 is N, and in the partial solution, we computed the $N+1$ values

$$f_0(0), f_0(1), ..., f_0(N).$$

Therefore, we can apply Lagrange interpolation to the polynomial f_0.

We only need to compute

$$f_0(K) - f_0(K-1)$$

using Lagrange interpolation.

Even a naive implementation of Lagrange interpolation takes $O(N^2)$ time, and in this case, since the values at the $N+1$ points $0, 1, \ldots, N$ are known, it can also be sped up to $O(N)$.

Since filling the DP table takes $O(N^2)$ time, the overall time complexity is $O(N^2)$ regardless of which interpolation method is used.

Problem H. How to Validate Such a Program

Let D be the distance matrix of a tree. Each query specifies a principal submatrix D' and returns $\operatorname{tr} D'^k_k$.

In this setting, $\det D'$ can be reconstructed using Newton's identities (the formulas expressing power sums in terms of elementary symmetric polynomials).

Here, a lemma known as the Graham-Pollak formula is known, whose statement is as follows.

- For the distance matrix M of a tree with N vertices, $\det M = (-1)^{N-1}(N-1)2^{N-2}$

Sketch of the proof: Take a leaf x and a vertex y adjacent to it, and perform elimination using $d(v,x) = d(v,y) + 1$ to obtain a nicely simplified matrix.

Using this, we can determine with high probability whether the induced subgraph of S is connected.

To determine whether a vertex v is a leaf, it suffices to take $S = V \setminus \{v\}$. Using the formula in `https://arxiv.org/pdf/2411.11488`, one can verify that when $P > N2^N$ and $S = V \setminus \{v\}$,

$$v \text{ is a leaf} \Leftrightarrow \det D' \bmod P = (-1)^N(N-2)2^{N-3}.$$

Problem I. Xor Magic Square

When N is even, the matrix in which all entries are 1 is a good matrix, and it is clearly the one with minimum total sum.

When $N = 1$, it is obvious that no good matrix exists.

Also, when $N = 3$, no good matrix exists either. Suppose, for contradiction, that there exists a good matrix of size 3. Let X be the XOR of the first row, the third row, the second column, and both diagonals. Using the fact that anything XORed an even number of times cancels out to 0, we obtain $X = A_{2,2}$. On the other hand, by the definition of a good matrix, we know that $X = 0$. Hence $A_{2,2} = 0$, which contradicts the definition of a good matrix, where all entries must be positive integers.

Below, we consider the case where N is odd and at least 5.

When $N = 5$, by some suitable method such as manual calculation or brute force search, we can find a good matrix with total sum $N^2 + 3N$, and this is minimal. One example is the following:

$$\begin{pmatrix} 1 & 2 & 3 & 1 & 1 \\ 1 & 1 & 1 & 2 & 3 \\ 2 & 3 & 1 & 1 & 1 \\ 1 & 1 & 2 & 3 & 1 \\ 3 & 1 & 1 & 1 & 2 \end{pmatrix}$$

When $N \geq 7$, we can obtain a minimum-sum good matrix of size N by surrounding a solution for $N-2$ with 1's, and setting

$$A_{1,1} = A_{N,N} = 2, \quad A_{1,N} = A_{N,1} = 3.$$

For example, when $N = 7$, we obtain the following:

$$\begin{pmatrix} 2 & 1 & 1 & 1 & 1 & 1 & 3 \\ 1 & 1 & 2 & 3 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 2 & 3 & 1 \\ 1 & 2 & 3 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 3 & 1 & 1 \\ 1 & 3 & 1 & 1 & 1 & 2 & 1 \\ 3 & 1 & 1 & 1 & 1 & 1 & 2 \end{pmatrix}$$

Problem J. Sum of max of iai

When the score is at most k, it is necessary and sufficient that for all $1 \leq i \leq N$,

$$a_i \leq \left\lfloor \frac{k}{i} \right\rfloor.$$

Thus, the number of permutations whose score is at most k can be computed as:

$$\prod_{i=1}^{N} \max\left(0, \min\left(N, \left\lfloor \frac{k}{i} \right\rfloor\right) - N + i\right).$$

From this, the answer is given by:

$$\begin{aligned} &\sum_{k=0}^{N^2-1} \left(N! - \prod_{i=1}^{N} \max\left(0, \min\left(N, \left\lfloor \frac{k}{i} \right\rfloor\right) - N + i\right)\right) \\ =&N^2 \cdot N! - \sum_{k=0}^{N^2-1} \left(\prod_{i=1}^{N} \max\left(0, \min\left(N, \left\lfloor \frac{k}{i} \right\rfloor\right) - N + i\right)\right). \end{aligned}$$

Therefore, it suffices to compute

$$\sum_{k=0}^{N^2-1} \left(\prod_{i=1}^{N} \max\left(0, \min\left(N, \left\lfloor \frac{k}{i} \right\rfloor\right) - N + i\right)\right)$$

efficiently.

Here, for each i, the value of $\min\left(N, \left\lfloor \frac{k}{i} \right\rfloor\right)$ changes at most N times. By precomputing the points at which these changes occur, the inner expression can be evaluated in $O(1)$ time, yielding an overall complexity of $O(N^2)$.

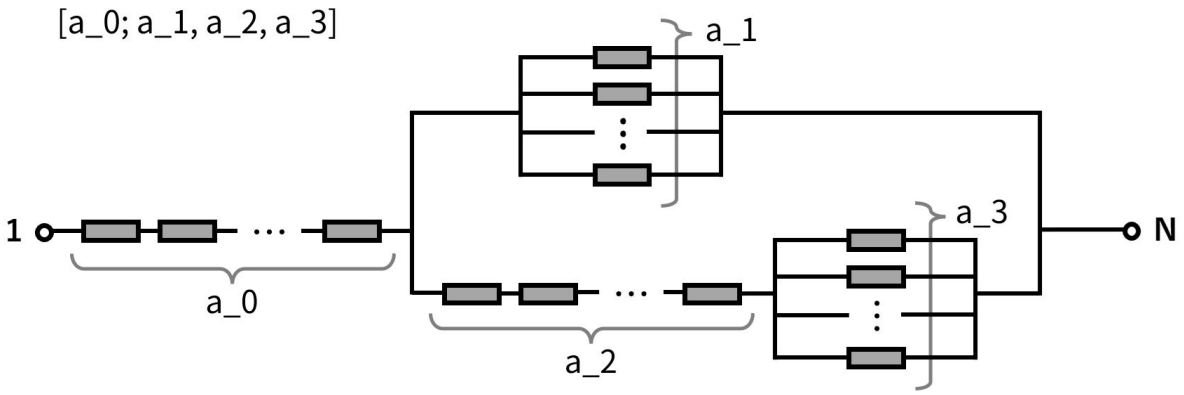
If we do not truncate the preprocessing of these change points, the complexity becomes $O(N^2 \log N)$, which is still sufficiently fast.

Problem K. Square Resistance Value

There are various possible approaches; here we introduce one example.

If we want to approximate $\sqrt{D}$, it is natural to consider its continued fraction expansion. By truncating the continued fraction expansion of $\sqrt{D}$ partway through, we can obtain a rational approximation.

A resistance network with value $[a_0; a_1, a_2, \ldots, a_k]$ can be constructed as follows. This can be realized using $a_0 + a_1 + a_2 + \cdots + a_k$ edges.



By maximizing the number of terms while keeping the total number of edges within 300, this approach is sufficient to solve the problem. In fact, the worst case for this method is $D = 3720$, where the error is about 5.5×10^{-7}.

There are also various alternative solutions, including ones that use different observations or heuristics.

Bonus! Given a graph, how can we determine **exactly**, without any numerical error, whether the difference between its equivalent resistance and $\sqrt{D}$ is at most 10^{-6}?

Problem L. Make Many KUPC

This problem can be solved optimally by repeatedly choosing, among all currently available quadruples (i, j, k, l), the rightmost one.

Therefore, the optimal approach is to scan S from right to left and greedily match characters that satisfy the required pattern.

Proof of the correctness of the greedy method:

Fix an arbitrary sequence of operations.

Suppose that in two consecutive operations in this sequence, we choose (i, j, k, l) and then (i', j', k', l') in this order. Then, even if we replace these two operations with the following two quadruples formed by pairing the coordinatewise maxima and minima, the total gained value does not decrease:

$$(\max(i, i'), \max(j, j'), \max(k, k'), \max(l, l'))$$

$$(\min(i, i'), \min(j, j'), \min(k, k'), \min(l, l'))$$

For example, suppose the ordering relations are

$$i < i', \quad j' > j, \quad k < k', \quad l < l'.$$

Then the two quadruples after replacement become (i', j, k', l) and (i, j', k, l').

The difference in the total gained value caused by this replacement is

$$(i'jk'l + ij'kl') - (ijkl + i'j'k'l') = (i'k' - ik)(jl - j'l').$$

Since $ik < i'k'$ and $jl > j'l'$, this value is nonnegative.

A similar inequality can be shown to hold in all other possible relative orderings as well.

Thus, we may assume without loss of generality that the chosen indices i, j, k, l in the operation sequence are each monotonically decreasing.

If the first operation does not choose the rightmost quadruple (i, j, k, l), then replacing it with the rightmost one can only increase the obtained value. The same applies to the second and later operations. Hence, choosing the rightmost (i, j, k, l) at every step is optimal.

From the above discussion, we obtain an algorithm that scans the string S from right to left while managing partially formed `C`, `PC`, and `UPC` using queues.

More concretely, we process S in decreasing order of index i, and do the following:

- If $S_i =$ `C`, add i to the queue managing `C`.
- If $S_i =$ `P` and the queue managing `C` is nonempty, extract one element from it, multiply it by i, and add the result to the queue managing `PC`.
- If $S_i =$ `U` and the queue managing `PC` is nonempty, extract one element from it, multiply it by i, and add the result to the queue managing `UPC`.
- If $S_i =$ `C` and the queue managing `UPC` is nonempty, extract one element from it, multiply it by i, and add the result to the answer.

Problem M. Linked VERSE

First, compress the sequence into a form where (positive numbers) and (-1) appear alternately.

If c is so large that 0 is always chosen in $x := \max(0, x - c)$, then it suffices to simply output the maximum value in the entire sequence. On the other hand, if c is so small that $x - c$ is always chosen, then the problem becomes one of finding the maximum of several linear functions, which can be solved using the Convex Hull Trick. The question is how to connect these two extremes.

Consider going from large c to small c while using the merging technique (small-to-large). Each subproblem maintains the following information:

- A set of points whose coordinates are given by (the number of occurrences of -1, the sum of positive integers up to that point)
  - In fact, it suffices to know only the upper convex hull, so we do that.
- For the current c, the point in that subproblem attaining the maximum value

Within each subproblem, we maintain the state in which we may regard $x := \max(0, x - c)$ as equivalent to $x := x - c$.

Therefore, we can manage when adjacent subproblems should be merged in $O(N \log N)$ time.

Also, when merging two upper convex hulls, a problem arises of merging upper convex hulls A and B together with a translation of B relative to A. However, there is a constraint that the lower-left point of B comes further up-right than the upper-right point of A. Because of this, the merge can be completed in $O(1)$ time per deleted point by repeatedly removing the middle point among any three consecutive points that form a concave turn at the merge boundary.

The update of "the point in the subproblem attaining the maximum value for the current c"can also be handled as one of the events. Moreover, when merging, this point needs to be recomputed, but since the number of points that can reappear is at most 1 per merge, the number of such updates does not increase enough to affect the overall complexity.

By handling everything appropriately, the process up to this point can be done in $O(N \log N)$ time.

Finally, by aggregating with a CHT supporting line insertion, taking c as the horizontal axis, we can solve the problem in $O(N \log^2 N)$ time.

Bonus! There is also a solution running in $O(N \log N)$ time.

Problem N. Cellular Component Constellation

Partial Hint 1

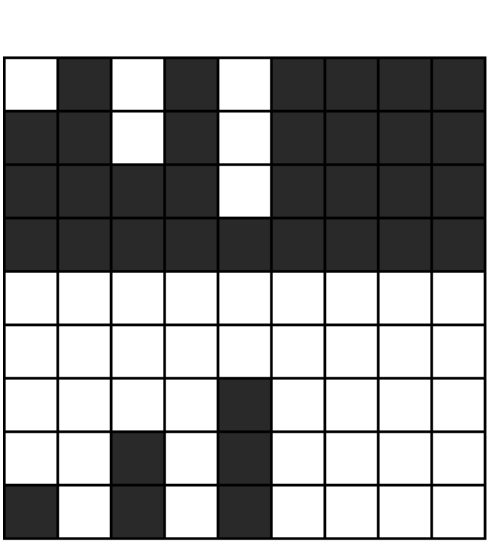
(constraint: $2M \leq N \leq 100$)

`https://atcoder.jp/contests/abc092/tasks/arc093_b` is a helpful reference.

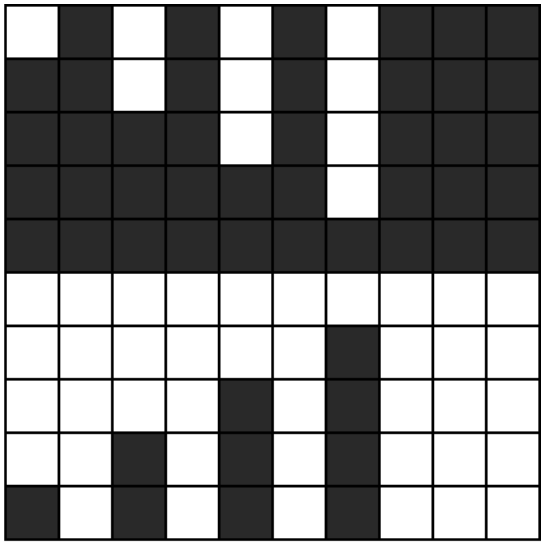
Partial Solution

In the partial dataset, since M is small relative to N, one can use the following method: first color one half of the board white and the other half black, then create small connected components of black cells inside the white region and small connected components of white cells inside the black region.

There is some freedom in how to divide the regions and place the connected components, but for example, by arranging the connected components as in the figure below, unnecessary connections and separations can be avoided.



N: 奇数



N: 偶数

Full Hint 1

As M becomes large relative to N, the construction seems to become difficult. Up to how large a value of M can we still construct such a grid?

Full Hint 2

Assuming that construction is possible when $M = N - 1$, we can see that just to create connected components of M different sizes, we need at least $N(N-1)$ cells. Think about a tiling that wastes as few cells as possible.

Full Hint 3

Naturally, let us consider constructing a rectangular grid by using, for both white and black, connected components of sizes $1, 2, \cdots, M$. Be careful that the construction fails if there is any undecided cell adjacent to connected components of both colors; this naturally limits the possible arrangements.

Full Solution

To create connected components of M different sizes, we need

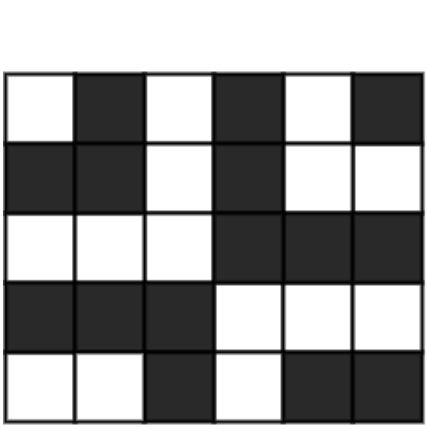
$$1 + 2 + \cdots + M = \frac{M(M+1)}{2}$$

cells. Therefore, counting both white and black, we need a total of $M(M+1)$ cells.

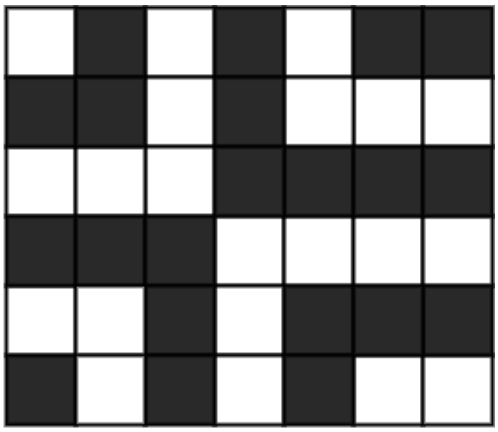
Hence, when $N \leq M$, since $N^2 < M(M+1)$, no grid satisfying the condition exists.

Now consider the case $N > M$. In fact, in this case we can always construct a grid satisfying the condition.

For both white and black, use connected components of sizes $1, 2, \cdots, M$, and combine them as shown below to construct an $M \times (M+1)$ grid.

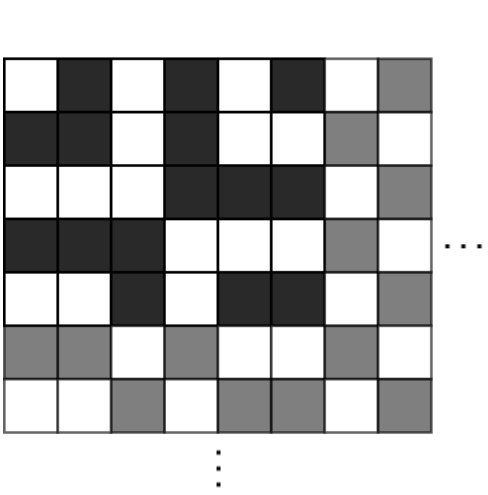


M: 奇数

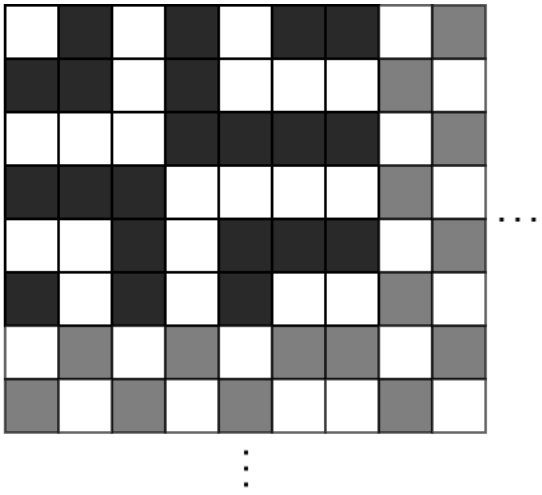


M: 偶数

After that, as shown in the figure below, by coloring the outside of the $M \times (M+1)$ grid in a checkerboard pattern, we can extend it to an $N \times N$ grid without changing the number of distinct connected-component sizes, and thus construct a grid satisfying the problem statement.



M: 奇数



M: 偶数

It is relatively easy to implement if you think of coloring cells according to the parity of the Chebyshev distance or Manhattan distance from a certain cell.

Problem O. Xor Triangle

Let the bitwise AND of a, b be denoted by $a\&b$. We use the following identities:

$$a + b = (a \oplus b) + 2(a\&b),\ a = b \oplus a \oplus b$$

Let $c = a \oplus b$, and consider the conditions for forming a triangle. Then the condition in the problem can be rewritten as follows:

$$a + b > c,\ b + c > a,\ c + a > b$$

$$\iff \quad a\&b > 0,\ b\&(a \oplus b) > 0,\ (a \oplus b)\&a > 0$$

In other words, the conditions are: there exists a bit position where both a and b have a 1 bit, and there exists a bit position where a has a 1 bit and b has a 0 bit, and there exists a bit position where b has a 1 bit and a has a 0 bit.

The number of pairs of integers (a, b) with $1 \leq a, b < 2^N$ satisfying these conditions can be computed by the principle of inclusion-exclusion as follows:

$$4^N - 3 \cdot 3^N + 3 \cdot 2^N - 1$$