Problem A. Binary Strings

Build an AC automaton using the strings S and T, and then use the fail tree to determine the state of each node:

- Cannot be traversed (including some string in T)
- Contains a string from S (record its index)
- Contains multiple strings from S
- Has no special properties

After pruning the nodes that cannot be traversed, condense the AC automaton into strongly connected components, and then use DAG DP combined with the above states to determine if there exists a path containing multiple strings from S. If such a path exists, the answer is `Yes`; otherwise, it is `No`.

Problem B. Collinear Arrangements

For a query given a point, there are three cases:

1. If the point is outside the convex hull, then find the two tangent points of the point with the convex hull. Note that enumerating the line on which the given point and the point on the convex hull lie, and the other intersection point with the convex hull, is a reverse direction. It can be scanned linearly in $O(n)$ time for all possible lines.
2. If the point is on the convex hull edge, then the point will only affect that particular edge.
3. If the point is inside the convex hull, then rotate around the convex hull. Note that enumerating the line on which the given point and the point on the convex hull lie, and the other intersection point with the convex hull, is also in $O(n)$ time.

For a query given two points, it is possible to determine if there is an intersection with the convex hull by using ternary search on the convex hull, with a complexity of $O(\log n)$.

Therefore, the total complexity is $O(q_1 n + q_2 \log n)$.

Problem C. Comedy's Not Omnipotent

Consider dividing the sequence into groups of 8, and for each group, consider an interaction strategy. The interaction strategy can be viewed as a decision tree. Using 256-bit compressed DP, it is possible to bruteforce the optimal decision tree for groups of 8 with an average depth (approximately 10^8 valid states, single core 1h / 20G memory). This results in an algorithm with a score of 0.533. The 0.533 algorithm can use batch processing to provide more efficient single-point queries (i.e., temporarily store encountered single-bit queries until there are 8, then use the decision tree to solve). By adding the new operation to the previous bruteforce DP, a new optimal decision tree can be found. The optimal score found is 0.498, with an average of 0.489 in practical testing, and $p99 \leq 0.492$. The larger the group, the smaller the average depth of the optimal decision tree, but it also becomes more difficult to find. Groups of 16/18/20 are easier to find relatively good decision trees, but it is expected to be difficult to tabulate larger groups.

Problem D. Deep Abyss

x is a 128-bit unsigned integer, so each bit can be treated as a binary variable, resulting in 128 binary variables $w_0, w_1, \cdots, w_{127}$.

For each bit v_b in a 128-bit variable v, we can maintain a set of binary variables S_{v_b} and a coefficient a_{v_b}, representing that v_b equals the XOR sum of all elements in $S_{v_b} \cup \{a_{v_b}\}$. Initially, the set of the i-th bit of x, S_{x_i}, is $\{w_i\}$ and $a_{x_i} = 0$. All operations in the problem can be processed according to the above rules. Therefore, the problem can be solved by solving a system of equations $\forall i \in \{0, 1, \cdots, 127\}, a_{x_i} + \sum\limits_{w \in S_{x_i}} w \equiv w_i \pmod 2$. The problem requires outputting the minimum solution if it exists, which can be achieved by greedily setting the higher bits to 0 during Gaussian elimination.

Time complexity: $O(W^2L + W^3)$, where W is the number of bits in the variable and L is the number of input lines.

```
Newline      = \n
DecimalDigit = 0 ... 9
HexDigit     = 0 ... 9 | a ... f
Letter       = a ... z
Hexadecimal  = 0 x HexDigit{1,32}
Variable     = Letter{1,4}
Term         = ( ~ | ) ( Variable | Hexadecimal )
ConstantTerm = ( ~ | ) Hexadecimal
ShiftAmount  = DecimalDigit
             | ( 1 ... 9 ) DecimalDigit
             | 1 ( 0 | 1 ) DecimalDigit
             | 1 2 ( 0 ... 8 )
Expression   = Term
             | Term ␣ ^ ␣ Term
             | Term ␣ & ␣ ConstantTerm
             | ConstantTerm ␣ & ␣ Term
             | Term ␣ | ␣ ConstantTerm
             | ConstantTerm ␣ | ␣ Term
             | Term ␣ \<\< ␣ ShiftAmount
             | Term ␣ \>\> ␣ ShiftAmount
Assignment   = Variable ␣ = ␣ Expression
Procedure    = ( Assignment Newline )*
```

Problem E. Lines on a Phone Screen

Consider using a segment tree to maintain the number of lines in a sequence of sentences.

To facilitate merging sequences from both ends, we define $f(l, r, b)$ to represent the number of lines required to typeset the remaining sentences $s_l, s_{l+1}, \cdots, s_r$ in sequential order, given that the first line already has a space of length b occupied. We also define $t(l, r, b)$ to represent the length of the last line in the corresponding situation. The valid states for b are limited to 24.

With this definition, for $mid = \lfloor \frac{l+r}{2} \rfloor$, $f(l, r, b)$ can be derived from $f(l, mid, b)$ and $f\big(mid + 1, r, t(l, mid, b)\big)$, while $t(l, r, b)$ can be directly derived from $t\big(mid + 1, r, t(l, mid, b)\big)$.

Using a segment tree to maintain $f(l, r, b)$ and $t(l, r, b)$ for each interval, updates are performed during modification and merging during queries, resulting in a time complexity of $O(b \cdot n \log n)$ and a space complexity of $O(bn)$.

Problem F. Majority

A naive sorting network requires $O(\log^2 n)$ layers, and we need to compress the number of layers. Note that the circuit constraints of the sorting network are very strict, so we can construct the circuit without following the layers, and the operation can have any number of inputs.

Using the merge sort method to sort n inputs, the resulting values are stored in n nodes, and it is required that their values satisfy that all the 1's are before all the 0's. In the i-th layer, divide the inputs into $n/2^i$ groups, each group containing 2^i nodes. In the 0-th layer, each group has one node, which is the input node and is already sorted. A group in the i-th layer is formed by merging two groups from the layer below, and the two groups below are already sorted, for example, they are 1100, 1110. We need to merge the information of the two groups of nodes so that the values of the 8 nodes in the current layer are 11111000. Consider when the j-th node in the current layer is 1: this requires the first group to have at least j 1's, and no requirement for the second group; or the first group has at least $j-1$ 1's, and the second group has at least 1 1; $\ldots$; or the first group has no requirement, and the second group has at least j 1's. Because the groups below are already sorted, a group having at least k 1's is necessary and sufficient for the k-th node of this group to have a value of 1. So, "the first group has at least a 1's, and the second group has at least b 1's" is equivalent to the a-th node of the first group having a value of 1, and the b-th node of the second group having a value of 1, which can be represented by the AND of these two nodes. Performing an OR operation on the results of these $j+1$ AND operations gives the merged j-th node.

When outputting, it is only necessary to output the $\lceil n/2 \rceil$-th node that is sorted.

This method has $\log n$ layers, each with n nodes. During the calculation process, an auxiliary node is introduced in each layer, and each node introduces at most n auxiliary nodes. Therefore, at most $n^2 \log n \approx 3000$ nodes are introduced, and $2\lceil \log n \rceil + 1 \approx 13$ layers.

If a larger branching factor is used in the lower layers instead of binary branching, the number of layers can be further reduced without significantly increasing the number of nodes.

Problem G. Matrices and Determinants

First, we can transform A into an upper triangular matrix A' through several elementary row operations, where the transformation process can be done using the method of subtraction to avoid real number operations. Then, we record the transformation operations and apply them to I to obtain the transformation matrix M, such that $A' = MA$. Similarly, if we reverse all the operations, take the inverse, and apply them to I, we obtain the inverse matrix M^{-1}. Therefore, if we can decompose A' into $A' = BC$, and $A = M^{-1}MA = M^{-1}A'$, then $M^{-1}B$ and C can be a solution.

Now consider the decomposition of $A' = BC$. Obviously, $det(A')$ must be a positive perfect square, in this case, let $det(B) = det(C) = \sqrt{det(A')}$. Consider constructing B and C recursively. Let $dfs(k, b)$ be the function to construct two $k \times k$ matrices B_k and C_k such that $det(B_k) = b$ and $B_k C_k = A'_k$. The process is as follows:

1. Let $B_{k,k} = \gcd(b, A'_{k,k})$ and $C_{k,k} = \frac{A'_{k,k}}{B_{k,k}}$
2. If $k > 1$, call $dfs\left(k-1, \frac{b}{B_{k,k}}\right)$ to obtain B_{k-1} and C_{k-1}, otherwise, exit directly
3. At this point, the first $k-1$ rows and columns of B and C have been determined. Let the kth row and jth column of B and C be 0. Now, enumerate i from $k-1$ to 1, consider $B_{i,k}$ and $C_{i,k}$. It can be seen that each time, an indeterminate equation $B_{i,k}C_{k,k} + C_{i,k}B_{i,i} = A'_{i,k} - s$ can be obtained, where $C_{k,k}, B_{i,i}, A'_{i,k}$ are known, and s is a determined number. Here, $C_{k,k} = \frac{A'_{k,k}}{B_{k,k}} = \frac{A'_{k,k}}{\gcd(b, A'_{k,k})}$, and $B_{i,i} \mid \frac{b}{\gcd(b, A'_{k,k})}$, so $\gcd(C_{k,k}, B_{i,i}) \mid \gcd\left(\frac{A'_{k,k}}{\gcd(b, A'_{k,k})}, \frac{b}{\gcd(b, A'_{k,k})}\right) = 1$, thus a set of $B_{i,k}, C_{i,k}$ can be obtained through exgcd.

In summary, if $det(A')$ is not a positive perfect square, there is no solution. Otherwise, calling $dfs\left(n, \sqrt{det(A')}\right)$ will yield a set of $BC = A'$, $det(B) = det(C) \neq 0$ for B and C, and finally output $M^{-1}B$ and C.

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

Problem H. Matrices and Sums

Because the absolute value of the matrix elements does not exceed 1, the possible row and column sums are only $-n$ to n, a total of $2n+1$ possibilities. Since there are a total of $2n$ row and column sums, what we need to do is to select $2n$ numbers from these $2n+1$ possibilities as the row and column sums. It is known that the sum of the absolute values of these $2n$ numbers, denoted as $SumAbs$, is greater than or equal to $2 \times \frac{n(n+1)}{2} - n = n^2$, and the first n largest numbers are non-negative while the last n largest numbers are non-positive.

Suppose that among the first n largest numbers, there are k row sums, corresponding to the set of rows denoted as SR, and there are $n-k$ column sums, corresponding to the set of columns denoted as SC. For an element $M_{i,j}$ in the matrix:

- If $i \in SR$ and $j \in SC$ are both satisfied, then the contribution of this element to $SumAbs$ is $2M_{i,j}$.
- If only one of $i \in SR$ and $j \in SC$ is satisfied, then the contribution of this element to $SumAbs$ is 0.
- If neither $i \in SR$ nor $j \in SC$ is satisfied, then the contribution of this element to $SumAbs$ is $-2M_{i,j}$.

It can be derived that: $SumAbs = \sum_{i \in SR, j \in SC} 2M_{i,j} + \sum_{i \notin SR, j \notin SC} -2M_{i,j} \leq \sum_{i \in SR, j \in SC} 2 + \sum_{i \notin SR, j \notin SC} 2 = 4k(n-k)$. Also, $SumAbs \geq n^2$, thus $4k(n-k) \geq n^2$. Simplifying, we get $(n-2k)^2 \leq 0$, so we can discuss based on the parity of n:

- If n is odd, there is no solution.
- If n is even, we can take the $2n$ numbers from $-n+1$ to n as the row and column sums. Let $M_{i,j} = [2j \leq n] - [i+j > n+1]$, then:
  - $R_i = \frac{n}{2} - i + 1 \, (i = 1, 2, \cdots, n)$, taking the numbers from $-\frac{n}{2}+1$ to $\frac{n}{2}$
  - $C_1 = n, C_2 = n-1, \cdots, C_{\frac{n}{2}} = \frac{n}{2}+1$, taking the numbers from $\frac{n}{2}+1$ to n
  - $C_{\frac{n}{2}+1} = -\frac{n}{2}, \cdots, C_n = -n+1$, taking the numbers from $-n+1$ to $-\frac{n}{2}$

  Thus, we have taken the $2n$ numbers from $-n+1$ to n.

Problem I. Palindrome Strings

For a palindrome combination string $t_1 t_2 \cdots t_{|t|} S_l S_{l+1} \cdots S_r$, there are two cases:

- $|t| < r-l+1$: In this case, reverse S to obtain S', use Manacher's algorithm to find all positions' palindrome radius, then find R_i representing the number of palindromic substrings with S'_i as the first character. Next, build a suffix automaton, for each node u, calculate the sum of R_i values for all descendant nodes in the fail tree as Sum_u. For a query t, it corresponds to a node x in the suffix automaton, and Sum_x is the number of palindrome combination strings in this case.
- $|t| \geq r-l+1$: In this case, use Manacher's algorithm to find all palindrome suffixes of t, including the empty suffix, and the suffix cannot be t itself. Then, traverse t on the suffix automaton built with S', and if at some point there are no remaining palindromic suffixes of t, add the occurrence count of the current prefix in S' to the answer.

Time complexity: $O(|S| + \sum |t|)$

Problem J. Apple Family Reunion

The operation is: interchange the middle and the small, interchange the small and the large, both are reversible.

The operation maintains the number of inversions and does not change the relative order of 1 and n.

Let $[x^k]f(x)$ denote the number of permutations with k inversions and 1 before n.

Then $f(x) = (1)(1+x)\cdots(1+x+\cdots+x^{n-3})\sum_{i=0}^{n-2} x^i(1+x+\cdots+x^{n-2-i})$.

The number of inversions ranges from 0 to $\binom{n}{2} - (n-1)$.

Similarly, when n is before 1, the number of inversions ranges from $n-1$ to $\binom{n}{2}$.

Therefore, there are $n^2 - 3n + 4$ classes.

Next, we prove that any two permutations in each class can be transformed into each other.

If there is a valley in the permutation, the lexicographical order decreases after the operation.

Therefore, the operation can be performed until there are no valleys.

The resulting sequence is either monotonically increasing or first increasing and then decreasing, and such a sequence is called tidy.

Furthermore, only a segment of the sequence needs to be operated on to make it tidy.

For the initial permutation p, a permutation q with the smallest lexicographical order can be constructed from the number of inversions: if 1 is before n, put 1, 2, ..., k in order, then the remaining inversions are at most $\binom{n-k}{2}$, find k such that the number of inversions is in the interval $(\binom{n-(k+1)}{2}, \binom{n-k}{2}]$, i.e., putting k+1 after k does not increase the number of inversions. If k+1 is increased to k+h, the number of inversions is at most $\binom{n-(k+1)}{2} + h - 1$, and $\binom{n-(k+1)}{2} + n - k - 1 = \binom{n-k}{2}$. Choose the smallest h to obtain the smallest lexicographical order.

If n is before 1, to make the lexicographical order smallest, put 2, 3, ..., h first, then the reverse order at the back obtains the maximum number of inversions $\binom{n-h+1}{2} + h - 1$, and increasing h by 1 increases the maximum number of inversions by 1. Therefore, the method of putting obtains the smallest lexicographical order for the same class.

After making p tidy, if the lexicographical order is not the smallest in the same class, find the leftmost different position, then the number b above is larger than the number a at the same position in q, indicating that a is on the right slope. To move a to the front, operate on the peak to move the rightmost peak to the right until the right side of the peak is a, then operate to move a to the left. After making the segment from b to a tidy, repeat the previous process to move a to the front and make the sequence to the right of a tidy, obtaining a tidier sequence with a smaller lexicographical order, and thus p can be transformed into q.

Finally, to calculate $[x^k]f(x)$, the classic model is: find the number of permutations of n with exactly k inversions.

Hint: How **Elegia**'s mind works?

The link to the original article: `https://blog.csdn.net/EI_Captain/article/details/109064865`

$[x^k](1)(1+x)\cdots(1+x+\cdots+x^{n-1}) = \frac{\prod_{h=1}^{n}(1-x^h)}{(1-x)^n}$.

Let $f(z) = \frac{\prod_{h=1}^{n}(1-x^h z)}{(1-x)^n}$.

Then $f(zx) = \frac{\prod_{h=1}^{n}(1-x^{h+1} z)}{(1-x)^n} = \frac{1-x^{n+1}z}{1-xz} f(z)$.

$(1-zx)f(zx) = (1-x^{n+1}z)f(z)$.

$[z^t](1-zx)f(zx) = [z^t](1-x^{n+1}z)f(z)$.

Let $f_t = [z^t]f(z)$.

$[z^t]f(zx) - x[z^{t-1}]f(zx) = [z^t]f(z) - x^{n+1}[z^{t-1}]f(z)$.

$(f_t - f_{t-1})x^t = f_t - x^{n+1}f_{t-1}$.

$f_t = \frac{x^t - x^{n+1}}{x^t - 1} f_{t-1} = -x^t \frac{1-x^{n-t+1}}{1-x^t} f_{t-1}$.

$x^{(1+t)t/2} \mid f_t$, so only need to iterate $O(k^{1/2})$ times, total complexity $O(k^{3/2})$.

Problem K. Twinning Totem

Given a connected graph G and a vertex u, find two spanning trees such that for any vertex v, the two uv paths in the trees have no common internal vertices.

If the graph contains a cut vertex x (removing which disconnects the graph), then the path in the original graph between two different connected components after removing x must pass through x. If u is not equal to x, then there is no solution.

Therefore, if the number of cut vertices ≥ 2, there is no solution. If the number of cut vertices $= 1$, then there may be a solution only if $u = x$. We will now prove that a 2-connected graph has a solution. For a 1-cut vertex graph, the solution for each connected component along with x combined is a valid solution.

For a 2-connected graph with $n \geq 3$, any vertex has a cycle containing it. Take a cycle C containing u, and then at each step, take an existing vertex and walk along an unused edge to another existing vertex, dividing the edge set into a base cycle C and several chains (ears).

For C, start from u and assign values to the vertices in increasing order in one direction to construct two trees (referred to as the ascending tree and the descending tree). Do not take the last edge in the ascending tree and the first edge in the descending tree. Then, for any non-u vertex v, the values of the vertices in the u to v path in the ascending tree are increasing, and in the descending tree (considering u as infinity), they are decreasing (*).

For each ear added, since only 2 endpoints have been assigned values, assign values in increasing order from the smaller endpoint to the larger endpoint (and do not repeat previous values). Do not take the last edge in the ascending tree and the first edge in the descending tree, so the two trees still satisfy the condition (*).

Modifying Tarjan can yield an $O(m)$ ear decomposition algorithm. In particular, ears without new vertices can be discarded, leaving $O(n)$ edges. The assignment process can be implemented using balanced trees or similar methods, and each query can be completed in $O(n \log n)$.

Problem L. Count the Orders

The cost can be represented as the sum of n expressions of the form $a_x - a_y (a_x > a_y)$. Because each number is adjacent to 2 numbers, the upper limit of the cost is that all a_x are the top $\lceil \frac{n}{2} \rceil$ largest, and all a_y are the top $\lceil \frac{n}{2} \rceil$ smallest. As long as we try to arrange the larger and smaller numbers alternately, we can achieve this upper limit.

If it is not an arrangement that alternates as much as possible, when we write out the above formula, we will find that it is definitely lower than this upper limit, so the number of arrangements is equal to the number of alternating arrangements.

If n is even, then first fix the position of any number (such as the largest number), there are n possibilities for this, and then clockwise determine what to put in the next position. According to the multiplication principle, the final number of arrangements is $n \times \frac{n}{2}! \times (\frac{n}{2} - 1)!$.

If n is odd, then first fix the position of the $\lceil \frac{n}{2} \rceil$-th largest (also the $\lceil \frac{n}{2} \rceil$-th smallest) number, and then clockwise determine what to put in the next position. According to the multiplication principle, the final number of arrangements is $n \times \lfloor \frac{n}{2} \rfloor! \times \lfloor \frac{n}{2} \rfloor! \cdot 2$. It can also be explained by first fixing the position of another number, but it is simpler to think about fixing the position of the middle-sized number.

Problem M. Simple Math Problem

Let $f(m,n) = \sum\limits_{i=0}^{\lfloor m/2 \rfloor} \sum\limits_{j=0}^{\lfloor n/2 \rfloor} \binom{i+j}{j}^2 \binom{m+n-2i-2j}{n-2j}$, it is known that $f(a,b) = f(b,a)$ and $f(m,0) = \lfloor m/2 \rfloor + 1$.
Also, $\binom{m+n-2i-2j}{n-2j} = \binom{m+n-1-2i-2j}{n-2j} + \binom{m+n-1-2i-2j}{n-1-2j}$, therefore:

$$f(m,n) = \sum_{i=0}^{\lfloor m/2 \rfloor} \sum_{j=0}^{\lfloor n/2 \rfloor} \binom{i+j}{j}^2 \binom{m+n-1-2i-2j}{n-2j} + \sum_{i=0}^{\lfloor m/2 \rfloor} \sum_{j=0}^{\lfloor n/2 \rfloor} \binom{i+j}{j}^2 \binom{m+n-1-2i-2j}{n-1-2j}$$

$= \sum_{i=0}^{\lfloor (m-1)/2 \rfloor} \sum_{j=0}^{\lfloor n/2 \rfloor} \binom{i+j}{j}^2 \binom{m+n-1-2i-2j}{n-2j} + \sum_{i=0}^{\lfloor m/2 \rfloor} \sum_{j=0}^{\lfloor (n-1)/2 \rfloor} \binom{i+j}{j}^2 \binom{m+n-1-2i-2j}{n-1-2j}$

$+ [2 \mid m] \sum_{j=0}^{\lfloor n/2 \rfloor} \binom{m/2+j}{j}^2 \binom{n-1-2j}{n-2j} + [2 \mid n] \sum_{i=0}^{\lfloor m/2 \rfloor} \binom{i+n/2}{i}^2 \binom{m-1-2i}{-1}$

$= f(m-1,n) + f(m,n-1) + [2 \mid m][2 \mid n] \binom{m/2+n/2}{n/2}^2 \binom{-1}{0}$

$= f(m-1,n) + f(m,n-1) + [2 \mid m][2 \mid n] \binom{m/2+n/2}{n/2}^2$

Therefore, n, m can be transformed into one odd and one even form. Suppose $m = 2b, n = 2a+1$, then: $f(m=2b, n=2a+1) = \sum_{i=0}^{b} \sum_{j=0}^{a} \binom{i+j}{j}^2 \binom{2b+2a+1-2i-2j}{2b-2i}$.

Consider a string s containing $n = 2a+1$ zeros and $m = 2b$ ones. For a given pair (i, j), take two strings x, y of length $i+j$ containing i ones and j zeros, and one string z of length $2a+1+2b-2i-2j$ containing $2b-2i$ ones and $2a+1-2j$ zeros, the number of arrangements is $\binom{i+j}{i}^2 \binom{2b+2a+1-2i-2j}{2b-2i}$.

Furthermore, since s has $a+b+1$ substrings of length $a+b+1$, and any two have an intersection, where the number of ones in the first substring and the last substring sum to $2b$ or $2b+1$, one is less than or equal to b, and the other is greater than or equal to b. As the substring window shifts to the right, the number of ones in the substring remains unchanged or changes by ± 1. Therefore, there exists a substring containing exactly b ones.

For two substrings containing the same number of ones, removing their intersection results in two strings that still contain the same number of ones. For a substring containing exactly b ones, take the leftmost substring t containing exactly b ones, remove the intersection (denoted as u), and obtain a possible pair of x, y, with lengths $i+j$, both containing i ones. Then, connect the remaining parts to form z, with a length of $2b+2a+1-2i-2j$ and containing $2b-2i$ ones.

Consider all substrings of length $a+b-i-j+1$ in z, similarly, it is known that there exists a substring containing exactly $b-i$ ones, and the length of u is also $a+b+1-i-j$, containing $b-i$ ones. If the leftmost substring v containing exactly $b-i$ ones is not the intersection u, then removing u, v and the overlapping part, the beginning of v is the left adjacent part of t, and the end of u is the end of t, indicating that t is not the leftmost. Therefore, it is proved that the leftmost substring containing exactly $b-i$ ones must be u, and thus the position of x, y can be uniquely determined by z.

Therefore, for each pair (x, y, z), a unique original string s can be determined, and a position containing exactly b ones can be specified (i.e., the position of y, and assume its starting position is $k+1$). For $k \in [0, a+b]$, accept the string s from the $k+1$st to the $k+a+b+1$st position containing exactly b ones, then there are a total of $(a+b+1)\binom{a+b+1}{b}\binom{a+b}{b}$ pairs (k, s), thus $f(2b, 2a+1) = \binom{a+b}{b}\binom{a+b+1}{b}(a+b+1)$, further yielding:

$$f(m,n) = \frac{\lceil (n+m)/2 \rceil! \lceil (n+m+1)/2 \rceil!}{\lfloor n/2 \rfloor! \lfloor m/2 \rfloor! \lceil n/2 \rceil! \lceil n/2 \rceil!}$$