2024 Stage 3: Division 1 Solutions

E·Space, gyh, Itst, JohnVictor, liuzhangfeiabc, SpiritualKhorosho[†]

February 11, 2024

Tsinghua University Algorithm Association

Apart from problem setters, special thanks to caeious, ix35, Mys.C.K., rsy, and others who participated in preparing the problems.

† Arranged in alphabetical order.

Part I (Speaker: *liuzhangfeiabc*)

K: Kids and Integers

F: Fast Algorithm

I: Interesting Words

Part II (Speaker: *Mys.C.K.*)

A: A Tree Game

B: Binary String

H: Hamiltonian Circuit

Part III (Speaker: *E.Space*)

E: Easily Broadcasable Tensors

J: Junctions

C: Colored Slime Balls

Part IV (Speaker: *JohnVictor*)

G: Generating The Sequence

D: Daisies on a Grid

Part I (Speaker: *liuzhangfeiabc*)

Kids and Integers by *liuzhangfeiabc*

Let $f(n)$ be the sum of the decimal digits of n. Define $f_k(n) = f(f(\cdots f(n)))$ (composed k times). Given positive integers N, k, m, find how many $n \in [1, N]$ satisfy $f_k(n) = m$.

$N \le 10^{1000}, k, m \le 10^9$.

Solution

For numbers not exceeding 10^{1000}, $f(n)$ is at most $f(999\ldots9)$ (1000 times 9) $= 9000$.

For numbers not exceeding 9000, $f(n)$ is at most $f(8999) = 35$.

For numbers not exceeding 35, $f(n)$ is at most $f(29) = 11$.

For numbers not exceeding 11, $f(n)$ is at most 9.

Therefore, any number not exceeding 10^{1000} will become a single-digit number after at most 4 applications of f, and will not change thereafter.

Thus, the data range of m and k is misleading.

Solution

For the original problem, we can split the process of applying f k times into two parts:

- First, apply f once to get a number not exceeding 9000.
- Then, apply f $k-1$ times to get m.

We directly preprocess all numbers from 1 to 9000 to see what they become after 0 to 4 applications of f.

Then, for each 1 to 9000, calculate the number of numbers in 1 to N whose digit sum is x.

- Perform straightforward digit DP: $dp(i, x, 0/1)$ represents the number of numbers with digit sum x and length at most i, whether exceeding the upper limit or not.

Then, for each 1 to 9000, check whether x becomes m after applying f $k-1$ times, and if so, add the contribution of $dp(x)$.

Fast Algorithm by *ltst*

Find the shortest cycle in a weakly connected graph with n vertices, m edges, and positive integer weights on the edges.

$n \le 3 \times 10^5$, $m - n \le 1500$.

Solution

Note that when the in-degree or out-degree of a node is 0, this node will not appear in the shortest cycle. We can remove such nodes.

Additionally, if a node v has both in-degree and out-degree exactly equal to 1, then we can directly contract the edges $u \to v$ and $v \to w$ into a single edge $u \to w$ with a weight equal to the sum of the weights of the original two edges. Since passing through v will always follow $u \to v \to w$, this does not change the existence and length of the shortest cycle.

Note that after performing these operations, the sum of in-degrees and out-degrees of each node is at least three, and each edge contributes only one in-degree and one out-degree. Therefore, let n' be the final number of vertices and m' be the number of edges. We have $2m' \geq 3n'$.

Since the graph is weakly connected, each time we remove a node using the above operations, we delete at least one edge. Thus, $m' - n' \leq 1500$, which implies $n' \leq 3000$ and $m' \leq 4500$.

Therefore, we enumerate each node in the final graph, run the shortest path algorithm, and update the answer for the shortest cycle. The

Interesting Words by *SpiritualKhorosho*

Given $s_1, \cdots, s_N$, find the number of index sequences $i_1, i_2, \cdots, i_k$ (k as any positive integer) satisfying the following conditions, where + indicates string concatenation:

- $s_{i_1} + s_{i_2} + \cdots + s_{i_k}$ is a palindrome of length L.

Ensure $1 \le N \le 333$, $1 \le L \le 1000$, and $\sum_{i=1}^{N} |s_i| \le 600$.

What does the title mean?
Using the words given in Example 3, you can reconstruct a famous English palindrome sentence, and the title is the Chinese translation of this sentence. In addition, the Pinyin (excluding tones) of the first and fourth characters, and the second and third characters of the title are the same.

Analysis

When solving string matching-related problems, there is a common strategy to keep track of the current position and which node it corresponds to in the automaton.

This problem requires the entire string to be a palindrome, so it's not convenient to maintain matching information in one direction. To solve this problem, let's match simultaneously from both sides: let $f(i,\cdots)$ denote the scenario when processing the i-th and $(L-i+1)$-th positions, with matching conditions represented by $\cdots$.

Analysis

When solving string matching-related problems, there is a common strategy to keep track of the current position and which node it corresponds to in the automaton.

This problem requires the entire string to be a palindrome, so it's not convenient to maintain matching information in one direction. To solve this problem, let's match simultaneously from both sides: let $f(i,\cdots)$ denote the scenario when processing the i-th and $(L-i+1)$-th positions, with matching conditions represented by $\cdots$.

When transitioning from $f(i,\cdots)$ to $f(i+1,\cdots)$, we need to enumerate the next letter α. The pointer corresponding to the i-th position, pointing to the node q_l in the automaton, should find the forward transition edge corresponding to α. The pointer corresponding to the $(L-i+1)$-th position, pointing to the node q_r in the automaton, should find the reverse transition edge corresponding to α. Forward transitions can be easily maintained using a Trie tree, but reverse transitions are more complicated.

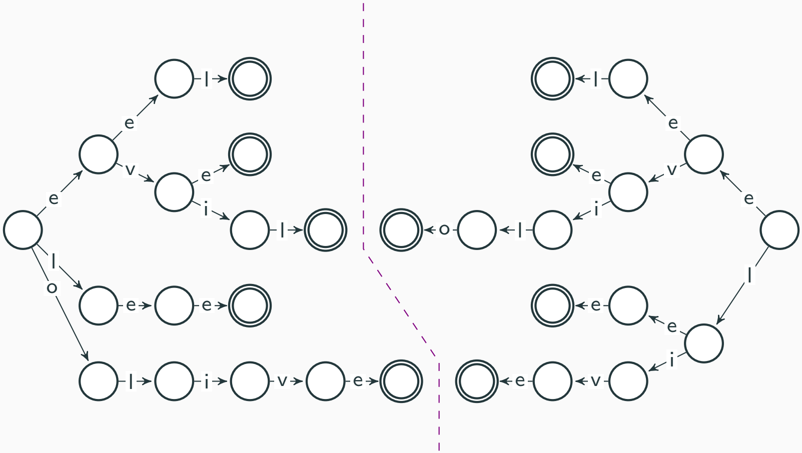
Analysis

When solving string matching-related problems, there is a common strategy to keep track of the current position and which node it corresponds to in the automaton.

This problem requires the entire string to be a palindrome, so it's not convenient to maintain matching information in one direction. To solve this problem, let's match simultaneously from both sides: let $f(i, \cdots)$ denote the scenario when processing the i-th and $(L-i+1)$-th positions, with matching conditions represented by $\cdots$.

When transitioning from $f(i, \cdots)$ to $f(i+1, \cdots)$, we need to enumerate the next letter α. The pointer corresponding to the i-th position, pointing to the node q_l in the automaton, should find the forward transition edge corresponding to α. The pointer corresponding to the $(L-i+1)$-th position, pointing to the node q_r in the automaton, should find the reverse transition edge corresponding to α. Forward transitions can be easily maintained using a Trie tree, but reverse transitions are more complicated.

Forward Trie and Backward Trie



Approach

Let $f\left(i, q_l, q_r\right)$ represent the scenario when processing the i-th and $(L-i+1)$-th positions, where the i-th position corresponds to the node q_l on the forward Trie tree, and the $(L-i+1)$-th position corresponds to the node q_r on the backward Trie tree. When transitioning, enumerate the letters α at the $(i+1)$-th and $(L-i)$-th positions. If both q_l and q_r have transition edges for α, then transition accordingly to $f\left(i+1, q_l', q_r'\right)$.

We need to handle the merging of q_l and q_r at $i = L/2$: if it's odd, the two nodes need to be able to reach the same position of the same word with the same letter; if it's even, both sides might reach the end of the word, or they might be adjacent positions of the same word.

- stac**k**/cats
- step/o**n**/**n**o/pets
- n**oo**n

Approach

The official solution represents $f\left(i, q_l, q_r\right)$ as the number of valid scenarios for the i-th to $(L-i+1)$-th positions when both sides are matched to q_l and q_r. First, calculate $f(\lceil L/2 \rceil, \cdot, \cdot) = 0/1$ based on the word list, and then expand $f(i, \cdot, \cdot)$ according to $f(i+1, \cdot, \cdot)$. Directly use loops to calculate DP with rolling arrays to avoid space issues.

- Of course, the space limit for this problem was expanded to 1024MiB. As long as you don't create two int arrays of size 500×600^2, space won't be an issue. If you want to use memoization, you can use char to record whether each state has been visited, which should also pass for this problem.

The complexity seems to be $O\left(L\left(\sum_{i=1}^{N}|s_i|\right)^2 |\Sigma|\right)$, but in reality, not all states will be filled. A more efficient approach is to enumerate the transition edges of q_l and q_r, achieving a strict $O\left(L\left(\sum_{i=1}^{N}|s_i|\right)^2\right)$, which can easily pass this problem.

N Range?

$$1 \times 26 + 2 \times (333 - 26) = 640 > 600.$$

$$\frac{600 - 1 \times 26}{2} = 287.$$

Part II (Speaker: *Mys.C.K.*)

A Tree Game by *Itst*

Given a tree with n nodes, the two players play a game on the tree.

Initially, node 1 has a chess piece. In each move, the first player prohibits an edge, and the second player chooses an edge that has not been prohibited to move the chess piece along it. The second player wins when reaching a node with exactly one degree, and the first player wins when unable to move.

Ask who wins when both players are extremely smart.

$1 \leq n \leq 10^5$

Solution

Firstly, it needs to be noted that the first player wins when $n = 1$ because there is no node with degree 1.

Solution

Firstly, it needs to be noted that the first player wins when $n = 1$ because there is no node with degree 1.

We first gain some more intuitive ideas:

- For the second player, going back is not optimal, as it gives the first player more chances to prohibit edges. So, we can consider the tree as an outward tree with 1 as the root, and the second player will only move the piece away from 1.

Solution

Firstly, it needs to be noted that the first player wins when $n = 1$ because there is no node with degree 1.

We first gain some more intuitive ideas:

- For the second player, going back is not optimal, as it gives the first player more chances to prohibit edges. So, we can consider the tree as an outward tree with 1 as the root, and the second player will only move the piece away from 1.
- For the first player, it is always more optimal to prohibit an edge connected to the piece in each move.

Solution

Firstly, it needs to be noted that the first player wins when $n = 1$ because there is no node with degree 1.

We first gain some more intuitive ideas:

- For the second player, going back is not optimal, as it gives the first player more chances to prohibit edges. So, we can consider the tree as an outward tree with 1 as the root, and the second player will only move the piece away from 1.
- For the first player, it is always more optimal to prohibit an edge connected to the piece in each move.
- If following the second player's strategy, which always moves away from 1, the first player will always prohibit the edge connected to the current node and one of its children in each move.

Solution

After considering the tree as an outward tree, a recursive structure naturally arises:

Solution

After considering the tree as an outward tree, a recursive structure naturally arises:

- If there are ≥ 2 winning subtrees, the second player can win by going into any of them;

Solution

After considering the tree as an outward tree, a recursive structure naturally arises:

- If there are ≥ 2 winning subtrees, the second player can win by going into any of them;
- If there are ≤ 1 winning subtrees, the first player prohibits the subtree, and the second player has no chance.

Solution

After considering the tree as an outward tree, a recursive structure naturally arises:

- If there are ≥ 2 winning subtrees, the second player can win by going into any of them;
- If there are ≤ 1 winning subtrees, the first player prohibits the subtree, and the second player has no chance.

Changing the outward tree to an undirected tree does not change the outcome:

Solution

After considering the tree as an outward tree, a recursive structure naturally arises:

- If there are ≥ 2 winning subtrees, the second player can win by going into any of them;
- If there are ≤ 1 winning subtrees, the first player prohibits the subtree, and the second player has no chance.

Changing the outward tree to an undirected tree does not change the outcome:

- We only need to consider whether the winning situation for the first player changes.

Solution

After considering the tree as an outward tree, a recursive structure naturally arises:

- If there are ≥ 2 winning subtrees, the second player can win by going into any of them;
- If there are ≤ 1 winning subtrees, the first player prohibits the subtree, and the second player has no chance.

Changing the outward tree to an undirected tree does not change the outcome:

- We only need to consider whether the winning situation for the first player changes.
- Since the second player loses in each subtree, even if they can go back into another subtree, they cannot win.

Solution

After considering the tree as an outward tree, a recursive structure naturally arises:

- If there are ≥ 2 winning subtrees, the second player can win by going into any of them;
- If there are ≤ 1 winning subtrees, the first player prohibits the subtree, and the second player has no chance.

Changing the outward tree to an undirected tree does not change the outcome:

- We only need to consider whether the winning situation for the first player changes.
- Since the second player loses in each subtree, even if they can go back into another subtree, they cannot win.

Therefore, DFS calculates the winning situation for each subtree when 1 is the root.

Complexity is $O(n)$, and it can be easily passed.

Binary String by *ltst*

Given a string s with the character set `01?`. For each $1 \le k \le |s|$, define the string t_k as the following `01` string:

- If s_i is not a ?, then $t_{k,i} = s_i$.
- Otherwise, $t_{k,i} = t_{k,i-k}$, and you need to recursively calculate $t_{k,i-k}$; if $i-k$ is out of bounds, it is `0`.

You need to output the number of `1` in each t_k.

$1 \le |s| \le 10^5$.

Solution

To obtain each character of the string t_k in order of indices results in an $O(n^2)$ solution.

Solution

To obtain each character of the string t_k in order of indices results in an $O(n^2)$ solution.

Notice that the character set is `01`, so consider bitwise compression, obtaining the $\omega = 64$ characters of t_k each time.

Solution

To obtain each character of the string t_k in order of indices results in an $O(n^2)$ solution.

Notice that the character set is `01`, so consider bitwise compression, obtaining the $\omega = 64$ characters of t_k each time.

For $k \le \omega$, the solution is straightforward. For $k \ge \omega$, the characters $t_{k,i} \ldots t_{k,i+\omega}$ depend on the characters $t_{k,i-k} \ldots t_{k,i-k+\omega}$, which are calculated in advance. Therefore, we need to perform the following operations to obtain this segment of characters:

Solution

To obtain each character of the string t_k in order of indices results in an $O(n^2)$ solution.

Notice that the character set is `01`, so consider bitwise compression, obtaining the $\omega = 64$ characters of t_k each time.

For $k \le \omega$, the solution is straightforward. For $k \ge \omega$, the characters $t_{k,i} \dots t_{k,i+\omega}$ depend on the characters $t_{k,i-k} \dots t_{k,i-k+\omega}$, which are calculated in advance. Therefore, we need to perform the following operations to obtain this segment of characters:

- Obtain the binary representation of $t_{k,i-k} \dots t_{k,i-k+\omega}$;

Solution

To obtain each character of the string t_k in order of indices results in an $O(n^2)$ solution.

Notice that the character set is `01`, so consider bitwise compression, obtaining the $\omega = 64$ characters of t_k each time.

For $k \le \omega$, the solution is straightforward. For $k \ge \omega$, the characters $t_{k,i} \dots t_{k,i+\omega}$ depend on the characters $t_{k,i-k} \dots t_{k,i-k+\omega}$, which are calculated in advance. Therefore, we need to perform the following operations to obtain this segment of characters:

- Obtain the binary representation of $t_{k,i-k} \dots t_{k,i-k+\omega}$;
- Obtain the binary representation of the positions of `?` and `1` in $s_{k,i} \dots s_{k,i+\omega}$.

Solution

To obtain each character of the string t_k in order of indices results in an $O(n^2)$ solution.

Notice that the character set is `01`, so consider bitwise compression, obtaining the $\omega = 64$ characters of t_k each time.

For $k \leq \omega$, the solution is straightforward. For $k \geq \omega$, the characters $t_{k,i} \dots t_{k,i+\omega}$ depend on the characters $t_{k,i-k} \dots t_{k,i-k+\omega}$, which are calculated in advance. Therefore, we need to perform the following operations to obtain this segment of characters:

- Obtain the binary representation of $t_{k,i-k} \dots t_{k,i-k+\omega}$;
- Obtain the binary representation of the positions of `?` and `1` in $s_{k,i} \dots s_{k,i+\omega}$.
- For the positions of `?`, use $t_{k,i-k} \dots t_{k,i-k+\omega}$, and for the non-`?` positions, use $s_{k,i} \dots s_{k,i+\omega}$, which can be easily implemented using bitwise operations after obtaining the binary representation.

Solution

If we use a long long to store a segment of characters like $t_{k,j\omega} \dots t_{k,(j+1)\omega-1}$, the first part is only related to two long longs. If we choose i as a multiple of ω each time, the second part can be preprocessed directly.

Solution

If we use a long long to store a segment of characters like $t_{k,j\omega} \ldots t_{k,(j+1)\omega-1}$, the first part is only related to two long longs. If we choose i as a multiple of ω each time, the second part can be preprocessed directly.

This results in a solution with complexity $O\left(\frac{n^2}{\omega}\right)$, which should be acceptable.

Hamiltonian Cycle by *Itst*

Given n pairs (a_i, b_i).

Consider a weighted directed complete graph G with n nodes, where the weight of edge $i \to j$ is $|a_i - b_j|$.

Find a Hamiltonian cycle in G that maximizes the sum of edge weights, and provide the maximum value.

$n \leq 10^5, a_i, b_i \leq 10^9$.

Solution

We need to maximize the sum of absolute values.

Solution

We need to maximize the sum of absolute values.

Since $|a_i - b_j| = \max(a_i - b_j, b_j - a_i)$, we can find a Hamiltonian cycle and assign positive and negative signs to each a_i and b_j so that each edge $i \to j$ has one positive and one negative number. The maximum value we seek is the maximum sum obtained from this assignment.

Solution

We need to maximize the sum of absolute values.

Since $|a_i - b_j| = \max(a_i - b_j, b_j - a_i)$, we can find a Hamiltonian cycle and assign positive and negative signs to each a_i and b_j so that each edge $i \to j$ has one positive and one negative number. The maximum value we seek is the maximum sum obtained from this assignment.

Due to the fact that each node in the Hamiltonian path has an indegree and outdegree of 1, the final assignment should have n positive and n negative numbers. Therefore, we need to find the assignment of n positive and n negative numbers that maximizes the sum, ensuring that there exists a cycle that satisfies the condition (a_i and b_j with opposite signs).

Solution

We need to maximize the sum of absolute values.

Since $|a_i - b_j| = \max(a_i - b_j, b_j - a_i)$, we can find a Hamiltonian cycle and assign positive and negative signs to each a_i and b_j so that each edge $i \to j$ has one positive and one negative number. The maximum value we seek is the maximum sum obtained from this assignment.

Due to the fact that each node in the Hamiltonian path has an indegree and outdegree of 1, the final assignment should have n positive and n negative numbers. Therefore, we need to find the assignment of n positive and n negative numbers that maximizes the sum, ensuring that there exists a cycle that satisfies the condition (a_i and b_j with opposite signs).

If we ignore the constraint of the cycle, the maximum sum assignment is simple: sort the numbers and take the larger n as positive and the smaller n as negative. However, this may not satisfy the Hamiltonian cycle condition, so we need to carefully consider this constraint.

Solution

Consider which sign assignments will violate the condition of a Hamiltonian cycle.

Solution

Consider which sign assignments will violate the condition of a Hamiltonian cycle.

Note that for the edges in the cycle, we require a_i marked as negative (positive) must connect to b_j marked as positive (negative). If all b_i marked as negative correspond to a_i marked as positive, then since b_i is negative and a_i is positive, i cannot be connected to j where b_j is positive and a_j is negative. Thus, there is no way to form a large cycle.

Solution

Consider which sign assignments will violate the condition of a Hamiltonian cycle.

Note that for the edges in the cycle, we require a_i marked as negative (positive) must connect to b_j marked as positive (negative). If all b_i marked as negative correspond to a_i marked as positive, then since b_i is negative and a_i is positive, i cannot be connected to j where b_j is positive and a_j is negative. Thus, there is no way to form a large cycle.

If only one of these situations exists or there exists a_k and b_k both marked as positive, we can connect the two parts through k, thus constructing a Hamiltonian cycle.

Solution

Consider which sign assignments will violate the condition of a Hamiltonian cycle.

Note that for the edges in the cycle, we require a_i marked as negative (positive) must connect to b_j marked as positive (negative). If all b_i marked as negative correspond to a_i marked as positive, then since b_i is negative and a_i is positive, i cannot be connected to j where b_j is positive and a_j is negative. Thus, there is no way to form a large cycle.

If only one of these situations exists or there exists a_k and b_k both marked as positive, we can connect the two parts through k, thus constructing a Hamiltonian cycle.

Therefore, the conditions for the nonexistence of a Hamiltonian cycle are:

Solution

Consider which sign assignments will violate the condition of a Hamiltonian cycle.

Note that for the edges in the cycle, we require a_i marked as negative (positive) must connect to b_j marked as positive (negative). If all b_i marked as negative correspond to a_i marked as positive, then since b_i is negative and a_i is positive, i cannot be connected to j where b_j is positive and a_j is negative. Thus, there is no way to form a large cycle.

If only one of these situations exists or there exists a_k and b_k both marked as positive, we can connect the two parts through k, thus constructing a Hamiltonian cycle.

Therefore, the conditions for the nonexistence of a Hamiltonian cycle are:

- There exists b_i marked as negative and a_j marked as positive, as well as b_j marked as positive and a_i marked as negative;

Solution

Consider which sign assignments will violate the condition of a Hamiltonian cycle.

Note that for the edges in the cycle, we require a_i marked as negative (positive) must connect to b_j marked as positive (negative). If all b_i marked as negative correspond to a_i marked as positive, then since b_i is negative and a_i is positive, i cannot be connected to j where b_j is positive and a_j is negative. Thus, there is no way to form a large cycle.

If only one of these situations exists or there exists a_k and b_k both marked as positive, we can connect the two parts through k, thus constructing a Hamiltonian cycle.

Therefore, the conditions for the nonexistence of a Hamiltonian cycle are:

- There exists b_i marked as negative and a_j marked as positive, as well as b_j marked as positive and a_i marked as negative;
- There does not exist a_k and b_k both marked as positive.

Solution

Since the initial greedy solution may not satisfy the constraints, we consider making the smallest adjustments possible to obtain a solution that satisfies the constraints.

Solution

Since the initial greedy solution may not satisfy the constraints, we consider making the smallest adjustments possible to obtain a solution that satisfies the constraints.

The first greedy adjustment is to swap the signs of the elements with ranks n and $n+1$. However, this may still not satisfy the constraints. If not, then the elements with ranks n and $n+1$ correspond to some pair (a_i, b_i). Therefore, further adjustments—swapping the signs of the elements with ranks n and $n+2$ or ranks $n-1$ and $n+1$—will obtain a valid solution.

Solution

Since the initial greedy solution may not satisfy the constraints, we consider making the smallest adjustments possible to obtain a solution that satisfies the constraints.

The first greedy adjustment is to swap the signs of the elements with ranks n and $n+1$. However, this may still not satisfy the constraints. If not, then the elements with ranks n and $n+1$ correspond to some pair (a_i, b_i). Therefore, further adjustments—swapping the signs of the elements with ranks n and $n+2$ or ranks $n-1$ and $n+1$—will obtain a valid solution.

Compare the legality of all four schemes and choose the optimal solution among the valid ones.

Solution

Since the initial greedy solution may not satisfy the constraints, we consider making the smallest adjustments possible to obtain a solution that satisfies the constraints.

The first greedy adjustment is to swap the signs of the elements with ranks n and $n+1$. However, this may still not satisfy the constraints. If not, then the elements with ranks n and $n+1$ correspond to some pair (a_i, b_i). Therefore, further adjustments—swapping the signs of the elements with ranks n and $n+2$ or ranks $n-1$ and $n+1$—will obtain a valid solution.

Compare the legality of all four schemes and choose the optimal solution among the valid ones.

The complexity is $O(n \log n)$.

Part III (Speaker: *E.Space*)

Easily Broadcastable Tensors by *ltst*

Two sequences $a_1, \ldots, a_m$ and $b_1, \ldots, b_n$ are called broadcastable if and only if the following conditions are met:

- For all $0 \le i \le \min(n, m) - 1$, either $a_{m-i} = b_{n-i}$ or $\min(a_{m-i}, b_{n-i}) = 1$.

Given two sequences of positive integers, you need to insert some 1s into both sequences, minimizing the number of insertions, while ensuring that after insertion, the two sequences are broadcastable.

Sequence length, value range ≤ 2000.

Solution

Since broadcasting requires matching the suffixes of the sequences, consider making decisions from the end. Also, this insertion and matching process is similar to edit distance, so consider dynamic programming.

Let $f_{i,j}$ be the minimum number of additional 1s needed when the first sequence has matched $i, \ldots, m$, and the second sequence has matched $j, \ldots, n$.

Solution

Three transitions:

- a_i is matched with a newly inserted 1, transition from $f_{i+1,j}+1$.
- b_j is matched with a newly inserted 1, transition from $f_{i,j+1}+1$.
- a_i and b_j can match directly $\left(a_i = b_j\right.$ or $\left.\min(a_i, b_j) = 1\right)$, transition from $f_{i+1,j+1}$.

The initial value is $f_{m+1,n+1} = 0$, and the final answer is $\min_{\min(i,j)=1} f_{i,j}$, as according to the definition of broadcasting, once one of the sequences is completely matched, the remaining prefixes of the other sequence do not need to be matched.

Complexity: $O(nm)$.

Junctions by *gyh*

There is an undirected weighted graph with n vertices. For each edge, determine whether it is a necessary edge in the shortest path between two points.

$n \le 500$

Solution

Property: For an edge (x, y), if it is a necessary edge in the shortest path between two points, then it must be a necessary edge in the shortest path between x and y.

Proof: If (x, y) is not a necessary edge between x and y, then each occurrence of x, y can be replaced by another path.

Solution

Determine whether (x, y) is the unique shortest path between x and y. Simply check if there exists z such that $d_{x,z} + d_{z,y} \leq w_{x,y}$. You can use the Floyd-Warshall algorithm to find all-pairs shortest paths and directly enumerate z for checking. The overall complexity is $O(n^3)$.

Colored Slime Balls by *E.Space*

Zuma game.

Each time a ball is launched, it costs w (can only increase the length of a certain segment).

At least k consecutive balls are required to eliminate.

Eliminating a segment of length i can get a score of p_i, ensuring $p_i - p_{i-1} < w$.

Initially, there are n segments, find the highest score after elimination.

$n \le 150, k \le 10$.

Solution

Interval DP, consider which segments to remove in the end.

This subsequence must satisfy being able to be divided into two parts with a length $< k$, and the total sum of the entire subsequence must be $\geq k$.

Then this subsequence divides the entire sequence into several segments.

In each segment, the last one removed cannot be the same color as this subsequence.

So, remembering these conditions in the state can perform the transition.

Solution

Let $f(l,r)$ be the answer for the interval $[l,r]$.

$g_1(l,r,p_1)$ is the answer for the case where the left endpoint of the interval is l, considering the part of the subsequence in $[l,r]$, and taking the r part, when the sum of the first part is p_1, the answer for all intervals not in the subsequence within the $[l,r]$ interval.

$g_2(l,r,p_2)$ is ... and the answer for the case where the sum of the second part is p_2, ... the answer.

Then enumerate the next/last segment for the transition.

Color conflict is when calculating f, consider $l-1$ and $r+1$.

Time complexity $O(n^3k^2)$.

Part IV (Speaker: *JohnVictor*)

Generating The Sequence by *JohnVictor*

Given a sequence $a_1, a_2, \cdots, a_n$ of length n, where each a_i is an odd number.

There are two types of operations:

1. Given l, r, x, add the even number x to $a_l, a_{l+1}, \cdots, a_r$;
2. Given l, r, calculate the product of $a_l, a_{l+1}, \cdots, a_r$, and output the result modulo 2^{20}.

$n, q \leq 2 \times 10^5$.

Solution

Consider a segment tree, where each node maintains the polynomial $\prod(x+a_i) \bmod x^{20}$.

For the translation operation, simply replace x with $x+\Delta_x$ and substitute. The complexity is $O(20^2)$. The correctness lies in the fact that, under the condition that x is guaranteed to be even and the answer is modulo 2^{20}, the polynomial modulo x^{20} does not affect the result.

For the product calculation operation, return the constant term.

The overall complexity is $O(20^2 n \log n)$.

Daisies on a Grid by *JohnVictor*

Recall the problem statement: There is a grid of size $n \times m$ colored with $\mathbb{Z}_3$.

A single evolution is defined as follows: For a cell with color c, if there is an adjacent cell with color $c-1$, then in the next round, the color of this cell becomes $c-1$, otherwise, it remains unchanged.

A coloring scheme is considered **good** if, and only if, after a finite number of evolutions, all cells have the same color.

For a good coloring scheme, its weight is defined as the smallest positive integer t such that after evolving the grid for t times, the color of the top-left cell remains unchanged.

Some cells on the grid have predetermined colors, while others do not. The task is to find the number of good coloring schemes and their total weight.

$2 \le n \le 5, 2 \le m \le 50$.

Solution

Observe that

$$\begin{bmatrix} 0 & 1 \\ 2 & 2 \end{bmatrix} \mapsto \begin{bmatrix} 2 & 0 \\ 1/2 & 1 \end{bmatrix},$$

which means that matrices of this type (or their rotations/flips, plus an added constant) will definitely map to themselves.

Solution

Observe that

$$\begin{bmatrix} 0 & 1 \\ 2 & 2 \end{bmatrix} \mapsto \begin{bmatrix} 2 & 0 \\ 1/2 & 1 \end{bmatrix},$$

which means that matrices of this type (or their rotations/flips, plus an added constant) will definitely map to themselves.

We call this structure **blocking** a grid from being good.

Solution

Observe that

$$\begin{bmatrix} 0 & 1 \\ 2 & 2 \end{bmatrix} \mapsto \begin{bmatrix} 2 & 0 \\ 1/2 & 1 \end{bmatrix},$$

which means that matrices of this type (or their rotations/flips, plus an added constant) will definitely map to themselves.

We call this structure **blocking** a grid from being good.

Claim: Grids without blocking structures are always good.

Solution

Now, let $C \in \mathbb{Z}_3^{n\times m}$ be a grid, and all 2×2 subgrids of C are not blocking structures.

We can always find an assignment of integers to each cell, denoted as $F \in \mathbb{Z}^{n\times m}$, such that:

- $C_{ij} = F_{ij} \bmod 3$.
- The difference between adjacent entries in F is at most 1.

Solution

$$C = \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, F = \begin{bmatrix} 0 & & \\ & & \\ & & \end{bmatrix}$$

Solution

$$C = \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, F = \begin{bmatrix} 0 & 0 & & \\ & & & \\ & & & \end{bmatrix}$$

Solution

$$C = \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, F = \begin{bmatrix} 0 & 0 & -1 \\ & & \\ & & \end{bmatrix}$$

$F_{ij} \in F_{i(j-1)} + \{-1, 0, 1\}$ ensures that F is uniquely determined.

Solution

$$C = \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, F = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ & & \end{bmatrix}$$

$F_{ij} \in F_{i(j-1)} + \{-1, 0, 1\}$ ensures that F is uniquely determined.

Solution

$$C = \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, F = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

$F_{ij} \in F_{i(j-1)} + \{-1, 0, 1\}$ ensures that F is uniquely determined.

Without blocking structures, it ensures that F is legal (ensuring that the constraints not used to generate F are satisfied).

Solution

$$C = \begin{bmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, F = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}$$

Solution

$$C = \begin{bmatrix} 0 & 2 & 2 \\ 0 & 0 & 2 \\ 1 & 0 & 0 \end{bmatrix}, F = \begin{bmatrix} 0 & -1 & -1 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}$$

Solution

$$C = \begin{bmatrix} 2 & 2 & 2 \\ 0 & 2 & 2 \\ 0 & 0 & 2 \end{bmatrix}, F = \begin{bmatrix} -1 & -1 & -1 \\ 0 & -1 & -1 \\ 0 & 0 & -1 \end{bmatrix}$$

Solution

$$C = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 0 & 2 & 2 \end{bmatrix}, F = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ 0 & -1 & -1 \end{bmatrix}$$

Solution

$$C = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}, F = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- Evolutions on C correspond to evolutions on F.
- Evolution on F: Change each cell to the maximum of itself and its neighbors.

Solution

$$C = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}, F = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- Evolutions on C correspond to evolutions on F.
- Evolution on F: Change each cell to the maximum of itself and its neighbors.
- Stable state: All numbers stabilize to the **minimum** value of F.

Solution

$$C = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}, F = \begin{bmatrix} -1 & -1 & -1 \\ -1 & -1 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- Evolutions on C correspond to evolutions on F.
- Evolution on F: Change each cell to the maximum of itself and its neighbors.
- Stable state: All numbers stabilize to the **minimum** value of F.
- Weight: Shortest path from the top-left cell to the **minimum value** in F.

Solution

Let's assume $n \leq m$, and fill in colors from the smallest to the largest column by column. Maintain the values of F on the contour line.

Solution

Let's assume $n \le m$, and fill in colors from the smallest to the largest column by column. Maintain the values of F on the contour line.

The state of F on the contour itself only needs to know the difference between adjacent numbers, and the number of states is $O(3^n)$.

Solution

Let's assume $n \le m$, and fill in colors from the smallest to the largest column by column. Maintain the values of F on the contour line.

The state of F on the contour itself only needs to know the difference between adjacent numbers, and the number of states is $O(3^n)$.

However, we also need to know the shortest distance from the top-left corner to the minimum value seen so far, which has a size of $O(m)$.

We also need to know how much larger the minimum value on the contour is than the minimum value, which has a size of $O(m)$.

The time complexity is $O(nm \cdot 3^n m^2)$.

Solution

Let's assume $n \leq m$, and fill in colors from the smallest to the largest column by column. Maintain the values of F on the contour line.

The state of F on the contour itself only needs to know the difference between adjacent numbers, and the number of states is $O(3^n)$.

~~However, we also need to know the shortest distance from the top-left corner to the minimum value seen so far, which has a size of $O(m)$.~~

We also need to know how much larger the minimum value on the contour is than the minimum value, which has a size of $O(m)$.

~~The time complexity is $O(nm \cdot 3^n m^2)$.~~

When updating the minimum value, decide if this is the last minimum value. If so, the shortest distance can be at most n more, which can be used to optimize the state, achieving a time complexity of $O(nm \cdot 3^n nm)$.

Thank you for listening!