Problem A. Life game

Find a minimum cut in a reduced graph, using a two-dimensional sparse table structure network

Dividing elements into two subsets resembles of $\langle S, T\rangle$-cut.

The idea is to build a network such that there is a bijection between $\langle S, T\rangle$-cuts and different colorings, and the cut value should be equal to the corresponding amount of money.

Assume that we have already earned all the money and have to return the money for non-satisfied awards, so minimize amount of money we have to return. Let this amount be the value of the cut.

Consider one award, let's say $c = 0$. Create one vertex v for this award, add edges of $+\infty$ capacity from v to vertices in its submatrix, add edge of m capacity from source to v.

For $c = 1$ in a similar manner, add edges of $+\infty$ capacity from vertices in its submatrix to v, and add an edge of m capacity from v to the sink.

One can see that if the award criterion is not satisfied, then the m-valued edge is intersecting a cut. This will result in up to $50^2 \cdot 50\,000$ edges and $50 + 50\,000$ vertices. This is quite much for max-flow algorithms.

Let's optimize the size of the network. We will make a 2D sparse table structure:

- add a vertex for every r, c, k_r, k_c, corresponding to submatrix $[r, r + 2^{k_r}) \times [c, c + 2^{k_c})$
- avoid adding edges from every cell to an award. Instead add 4 edges from 4 submatrices of power-of-2 sizes to an award as in a 2D sparse table. Note that we do not care if they overlap.
- add $+\infty$ edges between power-of-2 size submatrices from the larger one to the smaller ones by a factor of two.

Make 2 such structures: one for outgoing and another for incoming edges. As it turns out, this optimization helps to meet the TL. Fast algorithms like Dinitz or Preflow-push algorithms help.

Here we have about $2 \cdot 50^2 \cdot \log^2 50$ edges for the sparse table, $5 \cdot 50\,000$ edges for the edges between awards and submatrices, and $2 \cdot 50^2 \cdot \log^2 50 + 50\,000$ vertices.

Problem B. String Queries

Precompute the answer for each subsegment using LCP and range sums

You are given a string s of length up to $5\,000$, and queries (L, R) — find the number of different substrings in the substring $s(L, R)$.

The solution described here is similar to finding the number of different colors in subtree of a rooted tree. We will add $+1$ and -1 for some substrings, such that in total for each query segment, every distinct substring will result in either 1 or 0, depending on whether it has an occurence there or not.

We will build an array of $f_{L,R}$ — the number of substrings in $s(L, R)$.

- Initialize $f_{i,j} = 0$ for all i and j.
- For each substring $s(i, j)$, it is inside all (x, y) for $x \le i$ and $j \le y$. For all $i \le j$, add 1 to all $f_{x,y}$ such that $x \le i$ and $j \le y$.
- Get all substrings equal to some t — $t = s(i_1, j_1) = s(i_2, j_2) = \ldots = s(i_k, j_k)$. Note that:
  - $i_1 < i_2 < \cdots < i_k$
  - $j_1 - i_1 = j_2 - i_2 = \cdots = j_k - i_k$

  Subtract 1 from all $f_{x,y}$ such that $x \le i_e$ and $j_{e+1} \le y$, for all $1 \le e < k$.

Then the answer for the query will be $f_{L,R}$.

Why does $f_{L,R}$ end up being the number of distinct substrings in $s(L,R)$?

Consider some $s(L,R)$, suppose there are w copies of t in $s(L,R)$. We added 1 for each of the copies, so it is $+w$. For each pair of neighbouring copies, we subtracted 1, so it is $-(w-1)$. Other additions and subtractions were made for substrings that are outside of (L,R), so their left or right border is outside (L,R), contributing 0. Thus, it is $w-(w-1)=+1$ for each substring that is inside.

How to implement it efficiently?

Calculate $LCP(i,j)$ — the longest common prefix of suffixes $s(i,|s|)$ and $s(j,|s|)$. Fix i and iterate over j in increasing order, maintaining d as the length of the longest substring $s(i,i+d)$ equal to some $s(k,k+d)$ for $i<k<j$. If $LCP(i,j)>d$, we have found substrings with lengths $d+1\dots LCP(i,j)$ that weren't found for previous k. For each substring of length $d+1\le g\le LCP(i,j)$m subtract 1 from $f_{x,y}$ such that $x\le i$ and $j+g-1\le y$. For additions, we can use 2D prefix sums. The solution's time and memory complexity is $O(|s|^2+Q)$.

Suffix tree or automaton solution.

For every suffix, build suffix data structure in linear time. You can append a single letter and calculate the number of substrings. For instance, for an automaton it is `+= len[last] - len[sufLink[last]]`. This is still an $O(|s|^2)$ time and $O(|s|^2)$ memory solution.

Problem C. Coprimes

Optimize dynamic programming state by finding similarity in numbers

Given n up to 28, find the number of different permutations of length n where every two consecutive elements are coprime.

For each x, we are interested in the subset of $[1\dots n]$, so that x is coprime to them. Group them based on these resulting subsets. Based on these equivalence classes, count the number of different multisets of classes — there are 1 728 000 of those. Then apply dynamic programming to solve the problem.

Problem D. Colored Balls

Inclusion-exclusion principle and dynamic programming

You are given n white balls. In a single turn, you choose $(l,\,r)$ such that $1\le l\le r\le n$ equiprobably. Color each ball x, such that $l\le x\le r$, to black. What is the expected number of turns all balls to become colored?

The answer is $\sum\limits_{i=0}^{+\infty} p(i)$, where $p(x)$ is the probability that in x moves there still exists a white ball after x moves.

How to calculate $p(x)$?

Let d_A be the probability to leave a set A white in x moves. Choosing these intervals will color some subset of $\overline{A}$. Use inclusion-exclusion formula — probability to color all balls in x moves is $\sum\limits_{A} d_A\cdot(-1)^{|A|}$.

Let c_A be the number of intervals that cover only balls from $\overline{A}$. Then $d_A=\left(\frac{c_A}{\binom{n+1}{2}}\right)^x$.

So the answer is $\sum\limits_{i=0}^{+\infty}\left(1-\sum\limits_{A}(-1)^{|A|}\left(\frac{c_A}{\binom{n+1}{2}}\right)^i\right)$.

We know that $d_\varnothing=1$, so the answer is: $\sum\limits_{i=0}^{+\infty}\sum\limits_{A\neq\varnothing}(-1)^{|A|}\left(\frac{c_A}{\binom{n+1}{2}}\right)^i$.

Change the order to $\sum\limits_{A\neq\varnothing}(-1)^{|A|}\sum\limits_{i=0}^{+\infty}\left(\frac{c_A}{\binom{n+1}{2}}\right)^i$, we need to calculate 2^n geometric series sums. For every

c_A and $(|A| \bmod 2)$, calculate the number of such A. As an exercise, come up with a polynomial-time dynamic programming algorithm to do that.

Problem E. Another Tree Problem

Use Stirling numbers to calculate a^b

You are given a tree with at most 50 000 vertices. For each vertex v, calculate $\sum\limits_u d_{v,u}^k$, where $d_{x,y}$ is the distance from v to u. The value of k is up to 500.

Use Stirling numbers of the second kind $S(k, i)$ — the number of ways to partition a k element set into i subsets. Therefore $d^k = \sum\limits_{i=0}^{k} S(k, i) \cdot d \cdot (d-1) \cdot \cdots \cdot (d-i+1)$, simplifying to $d^k = \sum\limits_{i=0}^{k} S(k, i) \cdot \binom{d}{i} \cdot i!$.

For every vertex v, calculate the array $a_i = \sum\limits_u \binom{d_{v,u}}{i}$. To add one more edge to a path, every distance will increase by one:

- $\binom{d+1}{i} = \binom{d}{i} + \binom{d}{i-1}$
- $a_i^{\mathtt{new}} = a_i + a_{i-1}$

To calculate a:

- Make the tree rooted.
- Sum up all $\binom{d}{i}$ over all descendants.
- Sum up all $\binom{d}{i}$ for non-descendants by performing a second DFS.
- Calculate a for all vertices in $O(nk)$ time.
- Calculate $S(i, j)$ and $i!$.
- Use the formula to get the answer for every vertex.

Problem F. String and Queries — 2

Calculate the parities of each letter for each prefix, solve using the inclusion-exclusion principle and multi-dimensional partial sums

You are given a string s of length up to 10^5, consisting of the first 20 letters of the alphabet. Answer the queries:

- Given $c_1, c_2, \ldots c_k$ — letters, where $k \le 5$.
- Find the number of pairs (i, j) such that $s(i, j)$ contains an even number of each of these k letters.

For each prefix $0 \le i \le |s|$ find the subset p_i of letters that occur an odd number of times. For the substring $s(i+1, j)$, we have to calculate $p_i \oplus p_j$.

Let f_A be the number of i such that $p_i = A$.

Calculate $g_A = \sum\limits_{A \subset B} f_B$:

- It's just partial sums on a $2 \times 2 \times \ldots \times 2$ array.
- Calculated in $O(2^{|\Sigma|}|\Sigma|)$.

To answer the queries, p_i and p_j have to have equal parity for $c_1, c_2, \ldots, c_k$. For each X out of 2^k parities of the given k letters get g_A, where X is a subset of 2^k mapped to A, a subset of 2^n. Initialize $d_X = g_A$.

Then use the inclusion-exclusion principle:

```
for $X = (2^k - 1) \ldots 0$:
    for $Y \supset X$:
        $d_X := d_X - d_Y$
```

After that, d_X is the number of p_i such that the given letters' parity is the one from X and the other letters' parity is either odd or even.

The answer is $\sum\limits_X \frac{d_X(d_X-1)}{2}$.

Problem G. LCM

Try every number from $\frac{n}{2}$ to the next prime number

Given an integer n up to 10^9, find positive integers a and b such that:

1. $a + b = n$;
2. $\operatorname{lcm}(a, b)$ is the maximum possible.

Solution:

- If $x > y$ and $d > 0$, then $xy \geq (x + d)(y - d)$.
- Let p be the smallest prime greater than $\frac{n}{2}$.
  - It is coprime to $n - p$, since $n - p < p$.
  - So, the answer is not less than $(n - p)p$.
  - You do not have to look at values of $x > p$
  - The gap between prime numbers is small enough, so try every $\frac{n}{2} < x \leq p$.

Problem H. Erase the String

Dynamic programming over all subsequences

Given a string of length not greater than 16. In a single move, you can erase any subsequence that is a palindrome. Find the minimum number of moves to erase the entire string.

To solve the problem, for each of the $2^n - 1$ subsequences, determine if it is a palindrome. Let P be the set of all palindrome subsequences, and let $f[A]$ be the minimum number of moves to erase subset A. To calculate f, use $f[A] = \min\limits_{B \in P \wedge B \subset A} f[B] + 1$. This is calculated in $O(3^n)$ time.

Problem I. Thickness

Divide the plane by intersection points and vertices; perform a sweep line at each vertical segment

You are given several triangles. For every integer k, find the area of the plane covered by exactly k triangles.

Intersect all pairs of sides of all triangles to get all x-coordinates of all intersections and vertices. Let $x_1 < x_2 < \cdots < x_m$ be those coordinates. Consider the part of the plane with points (x, y) such that $x_i < x < x_{i+1}$. The intersection of this part of the plane with each triangle is either an empty set or a trapezoid since no two non-vertical trapezoid sides intersect in this area.

For every segment of the plane between x_i and x_{i+1}:

- Get every side of all triangles that intersect this segment.
- Compute the y-coordinate at $\frac{x_i+x_{i+1}}{2}$ for these sides.
- Sort all the sides by this y-coordinate.
- Iterate over these line segments as a sweep line moving from bottom to top.
- Each side of a triangle either opens the triangle or closes it.
- Keep track of the number of opened triangles k. Increase the answer for k by the trapezoid area between the last two triangle sides.

Problem J. GCD

Offline queries: for every divisor, set its value to the second-to-last element divisible by it, and RMQ

You are given n integers ($n \le 50\,000$), each number not exceeding $50\,000$. You are also given queries:

- Each consists of L and R.
- Find a pair (i, j) such that $i \ne j$ and $L \le i, j \le R$:
  - And $\gcd(a_i, a_j)$ is the maximum possible.

Let's answer all the queries in order of increasing R. Consider gcd is equal to v, and denote the rightmost two positions $i < j \le R$ such that $v \mid a_i$ and $v \mid a_j$. The notation $a \mid b$ means a is a divisor of b. Then for every $L \le i$, answer is at least v.

For every $1 \le i \le R$ store the maximum possible divisor v of a_i such that there is $i < j \le R$, and $v \mid a_j$. Maintain segment tree or binary indexed tree, denoted as t, to calculate the maximum value on the range. Additionally, for each v, maintain the last index that is divisible by v. To increase R by one, iterate over all $v \mid a_R$, and:

- Update `t[last[v]] := max(t[last[v]], v)`.
- Update `last[v] := R`.

The answer for the query (L, R) is the maximum value in `t[L..R]`.

Problem K. Points on a Plane

Look at points that are closer than the current answer so far, just by x-coordinate

You are given a sequence of n points in 2D, with n being at most $5 \cdot 10^5$. The points are generated pseudorandomly with coordinates up to n. After each point is given, determine the distance between the two closest points.

Maintain an array of already added points sorted by x-coordinate. When a new point (x_0, y_0) is added:

- Let d be the current answer.
- Check all points (x, y) such that $|x - x_0| < d$, noting that other points are not closer than d.
- Update the answer by calculating the distance to these points.

Intuitively, the time complexity can be explained like this. Consider we have added p points to our set. Let's create an $r \times r$ grid over the $[1 \dots n] \times [1 \dots n]$ area. Choose r such that the probability of two points being located in the same cell is at least $\frac{1}{2}$. The birthday paradox suggests that the number of cells can be quadratic in p, so $r \approx p$. So the expected number of points in the range $x_0 - d < x < x_0 + d$ is $\frac{2pd}{n}$, and $d \approx \frac{n}{r} \approx \frac{n}{p}$, the expression $\frac{2pd}{n} \approx 2\frac{\frac{n}{p}p}{n} = 2$.

So summing up over all p, we achieve $O(n)$ runtime.