Polytechnic University of Catalonia Contest Analysis

40th Petrozavodsk Programming Camp

UPC teams

February 2021

Problem J - Joyful Numbers

Statement

We say that an integer $n \geq 1$ is *joyful* if appending the digits 25 to the right of n we get a perfect square.

Find the number of distinct prime factors of the k-th joyful number (with $1 \leq k \leq 10^9$).

Author: Xavier Povill

- Number of solves: 97
- First team to solve: NNSU: Almost Retired (Danilyuk, Kalinin, Ryabchikova)

Problem J - Joyful Numbers

Statement

We say that an integer $n \geq 1$ is *joyful* if appending the digits 25 to the right of n we get a perfect square.

Find the number of distinct prime factors of the k-th joyful number (with $1 \leq k \leq 10^9$).

Key observation

The k-th joyful number is $k(k+1)$.

Problem J - Joyful Numbers

Key observation

The k-th joyful number is $k(k+1)$.

Proof

By definition, n is joyful iff $100n + 25$ is a perfect square.

$$100n + 25 = 5^2(4n+1) \implies \text{it is a square iff } 4n+1 \text{ is a square}$$

$4n+1$ is odd, so it must be the square of an odd number. Let this odd number be $2\ell + 1$, with $\ell \geq 0$. Then,

$$4n + 1 = (2\ell+1)^2 = 4\ell^2 + 4\ell + 1 \implies n = \ell(\ell+1)$$

As we are only interested in $n \geq 1$, the k-th joyful number will be $k(k+1)$.

Problem J - Joyful Numbers

Statement

We say that an integer $n \geq 1$ is *joyful* if appending the digits 25 to the right of n we get a perfect square.

Find the number of distinct prime factors of the k-th joyful number (with $1 \leq k \leq 10^9$).

Key observation

The k-th joyful number is $k(k+1)$.

Solution

As k and $k+1$ are coprime, they don't share any prime factor, so we can compute the number of prime factors of k and $k+1$ separately, and the answer will be the sum. This can be done in $\mathcal{O}(\sqrt{k})$.

Problem I: Interesting Scoring Systems

Statement

There is a tournament of chess with n players. Points can be assigned in two different ways: either $2 * wins + draws$ (Abel's criteria) or $3 * wins + draws$ (Bolzano's criteria). We have to determine if, given the points with each criteria, it is possible to create a tournament with said punctuation such that no one won player 0 and there is a path of wins from player 0 to any other player.

Author: Edgar Moreno

- Number of solves: 94
- First team to solve: SPb ITMO University: Standard deviation (Budin, Kirillov, Sayutin)

Problem I: Interesting Scoring Systems

Key observation

Then number of wins of player i is the difference between the points he has according to Bolzano and Abel.

Solution

if $n \geq 3$, 0 can be the winner iff he won at least one match and the sum of the wins of all players is at least $n - 1$. We have to be careful with cases $n = 1$ and $n = 2$.

Observation

The solution has cost $\mathcal{O}(n)$.

Problem K: Königsberg Bridges

Statement

Given an undirected graph G, add some edges to G such that there exists a simple path that goes through all its bridges and the number of bridges is maximized.

Author: Félix Moreno Peñarrubia

- Number of solves: 90
- First team to solve: NNSU: Almost Retired (Danilyuk, Kalinin, Ryabchikova)

Problem K: Königsberg Bridges

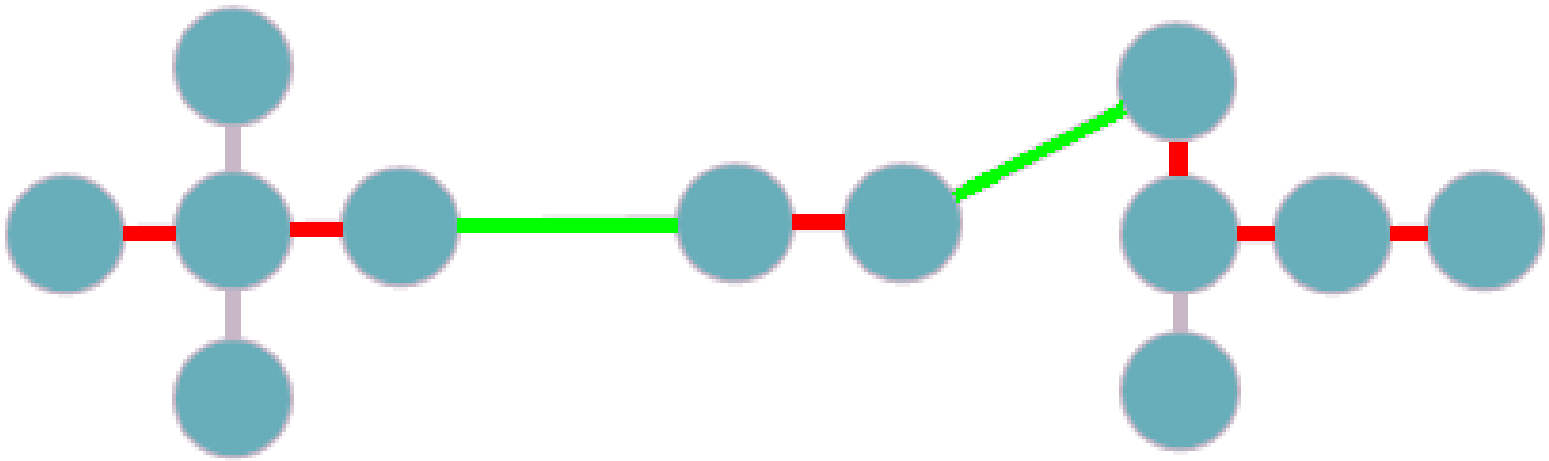
Key observation

If G is connected, we can't increase the number of bridges by adding edges.

We split G into connected components $C_1, \ldots, C_k$ and treat each connected component separately.
We find the bridges of each connected component C_i using DFS traversal and split it into 2-edge-connected components. Create a new graph C_i' where each node is a 2-edge-connected component of C_i and each edge is a bridge that connects two of these 2-edge-connected components. C_i' will be a tree.
Use BFS to calculate the diameter of C_i'. Let's call this diameter d_i. This tells us the maximum number of bridges we can cross in a simple path.
We can eliminate the remaining bridges by duplicating them.

Problem K: Königsberg Bridges

Once we do this calculation for all connected components, we can add edges to join all the connected components in a straight line. Each edge we add will be a bridge, and we'll be able to cross all of them in a simple path.



This means that the answer is $k - 1 + \sum_{i=1}^{k} d_i$.

Problem C: Cartesian MST

Statement

Given two graphs of n and m nodes, compute the MST of their cartesian product.

Author: Félix Moreno

- Number of solves: 89
- First team to solve: Harbor Space U: Spacers (Oboznyi, Velasco Sotomayor, Yousefi)

Problem C: Cartesian MST

Solution

We will use Kruskal's algorithm: we will be adding the edges by order of weight, avoiding forming cycles. For each group of edges in the cartesian product corresponding to a single edge in an original graph, we will add them at once. The structure of the cartesian product can be seen as placing one copy of one of the graphs at each vertex of the second graph: when we add a group of edges corresponding to the second graph, all the copies of the first graph will have the same edges added and in particular the same connected components, so one edge between the same connected component of the copies of the first graph will create a cycle in the cartesian graph only if such edge would create a cycle in the second graph. Therefore, the number of edges that can be added without creating a cycle is the number of current connected components in the first graph. So we can just compute the MST of the two graphs at the same time, and each time we take an edge, we multiply its cost by the current number of connected components in the other graph's MST.

Problem L: Long grid covering

Statement

Count the number of ways to fill a $3 \times n$ grid using triominos.

Author: Jordi Rodriguez

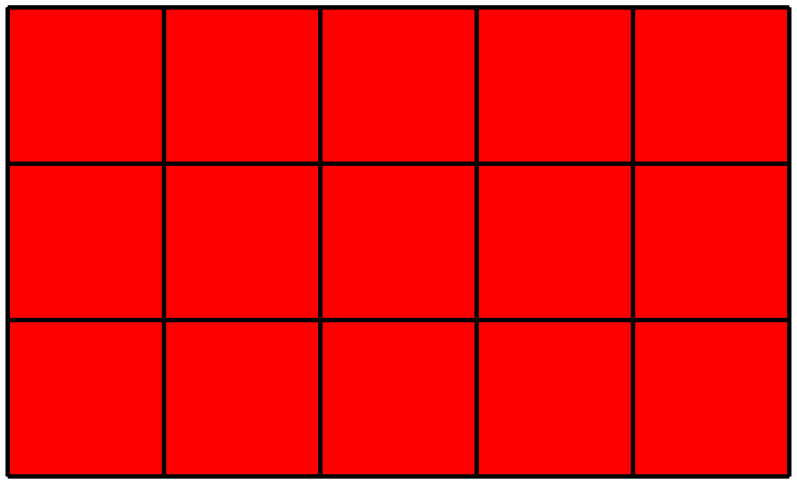
- Number of solves: 75
- First team to solve: Radewoosh Team (Radecki)

Solution

The solution is to find a recurrence relation in matrix form and compute the answer using binary exponentiation. One possible way to find the recurrence is the following:

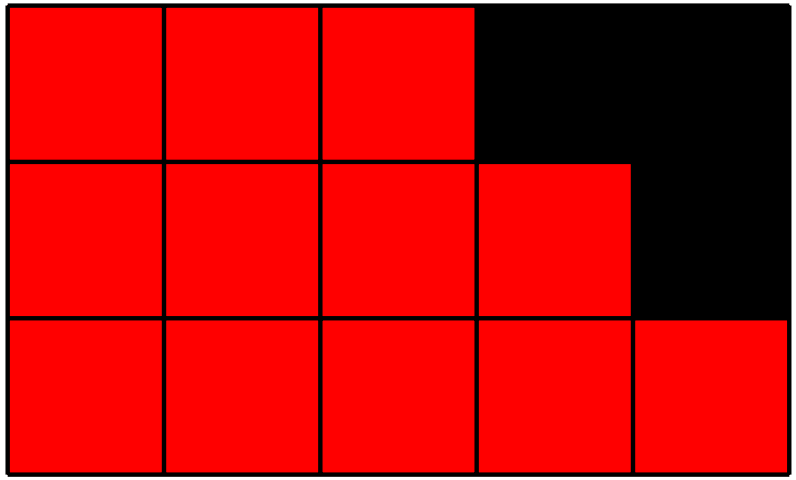
Problem L: Long grid covering

Let's consider 7 sequences $(a_k), (b_k), (c_k), (d_k), (e_k), (f_k), (g_k)$. Each sequence counts the number of ways to fill a $3 \times n$ grid using triominos in which some cells may have been removed.
a_n is the number of ways to fill a $3 \times n$ grid (the answer to the problem):

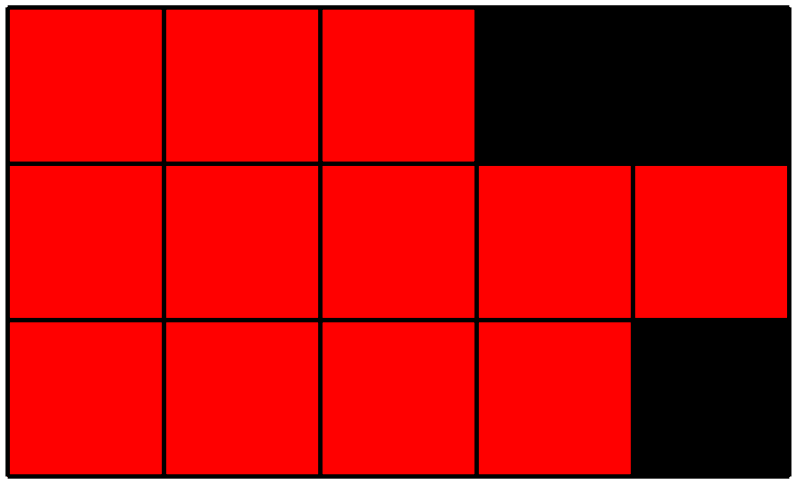


Problem L: Long grid covering

b_n is the number of ways to fill the following $3 \times n$ grid:

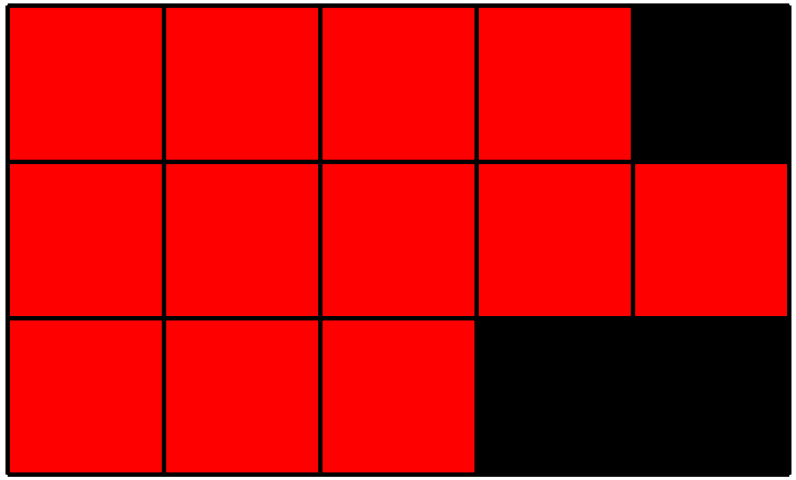


c_n is the number of ways to fill the following $3 \times n$ grid:

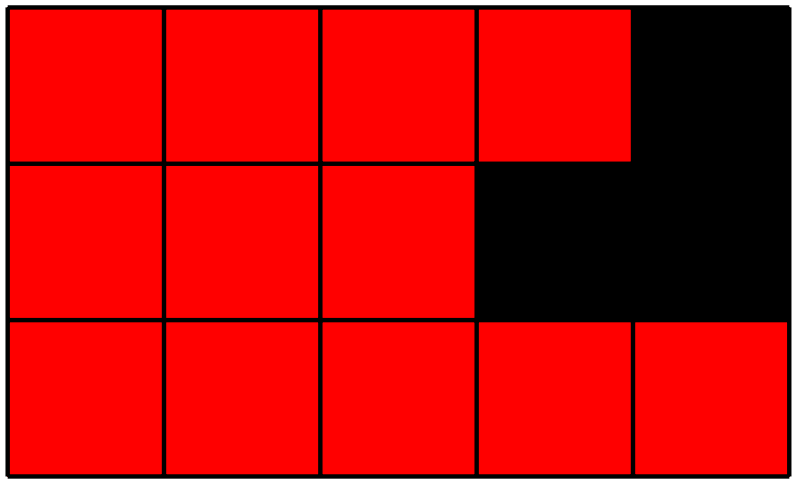


Problem L: Long grid covering

d_n is the number of ways to fill the following $3 \times n$ grid:

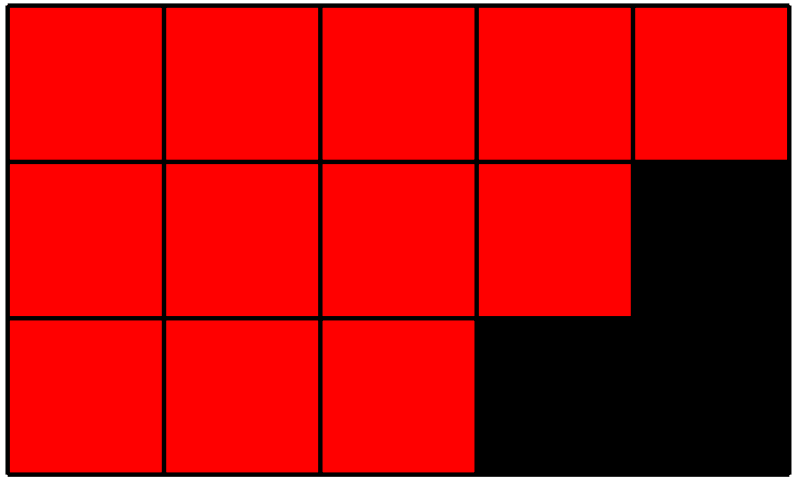


e_n is the number of ways to fill the following $3 \times n$ grid:

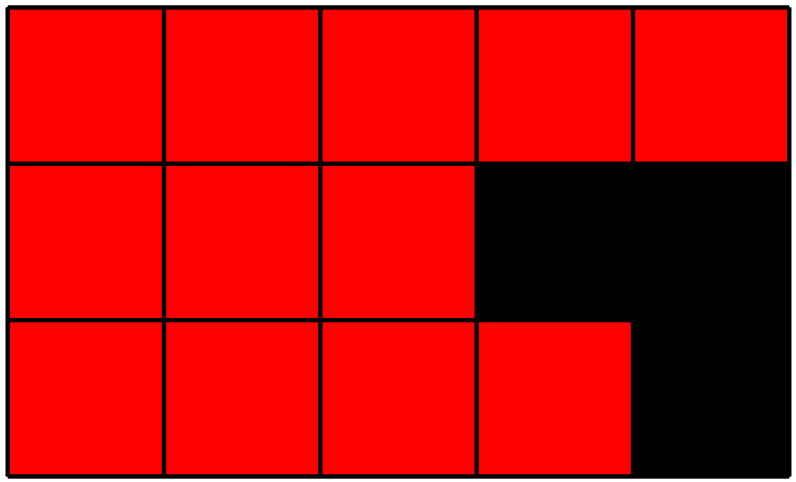


Problem L: Long grid covering

f_n is the number of ways to fill the following $3 \times n$ grid:



g_n is the number of ways to fill the following $3 \times n$ grid:



Problem L: Long grid covering

We have the following relations between the sequences:

$$a_{n+1} = a_n + a_{n-2} + b_{n+1} + e_{n+1} + f_{n+1} + g_{n+1}$$

$$b_{n+1} = a_{n-1} + d_n$$

$$c_{n+1} = a_{n-1} + e_n$$

$$d_{n+1} = a_{n-1} + g_n$$

$$e_{n+1} = f_n$$

$$f_{n+1} = a_{n-1} + c_n$$

$$g_{n+1} = b_n$$

Problem L: Long grid covering

And after some substitutions, those relations can be transformed into:

$$a_{n+1} = a_n + 2a_{n-1} + a_{n-2} + d_n + f_n + c_n + b_n$$

$$b_{n+1} = a_{n-1} + d_n$$

$$c_{n+1} = a_{n-1} + e_n$$

$$d_{n+1} = a_{n-1} + g_n$$

$$e_{n+1} = f_n$$

$$f_{n+1} = a_{n-1} + c_n$$

$$g_{n+1} = b_n$$

Problem L: Long grid covering

Now we can express those relations in matricial form:

$$\begin{pmatrix} a_{n+1} \\ a_n \\ a_{n-1} \\ b_{n+1} \\ c_{n+1} \\ d_{n+1} \\ e_{n+1} \\ f_{n+1} \\ g_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & 2 & 1 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_n \\ a_{n-1} \\ a_{n-2} \\ b_n \\ c_n \\ d_n \\ e_n \\ f_n \\ g_n \end{pmatrix}$$

And using binary exponentiation, we can find the answer with cost $\mathcal{O}(M^3 \log(n))$ where M is the size of the matrix

Problem E: Even intervals

Statement

Given an array of size n, answer q queries. Each query consists of two integers l, r and you should compute the sum of the numbers at the even positions of the array containing elements $a_l, \ldots, a_r$ once sorted.

Author: Max Balsells

- Number of solves: 73
- First team to solve: Peking U: Inverted Cross (Kong, Pan, Zhou)

Problem E: Even intervals

Solution

The idea of the problem is to process queries offline using Mo's Algorithm, keeping track of the sum of the numbers in even positions using a balanced BST.

Mo's Algorithm

We sort queries in the following way:

$$(\ell_i, r_i) < (\ell_j, r_j) \text{ if } \begin{cases} \left\lfloor \frac{\ell_i}{\sqrt{N}} \right\rfloor < \left\lfloor \frac{\ell_j}{\sqrt{N}} \right\rfloor \\ \text{or} \\ \left\lfloor \frac{\ell_i}{\sqrt{N}} \right\rfloor = \left\lfloor \frac{\ell_j}{\sqrt{N}} \right\rfloor \text{ and } r_i < r_j \quad (*) \end{cases}$$

Then we can update intervals by adding subtracting points. At the end, we will perform at most $\mathcal{O}(\sqrt{n}(n+q))$ insertions/deletions

Problem E: Even intervals

Addition/Subtraction

Now all we have to do is add a point at one of the ends of an interval/take out one of its extremes. In order to do it efficiently, we store all the values of the interval we are considering in a balanced BST. In each node we store the number of childs it has and the sum of the numbers at the even position of its subtree (sum_{even}), as well as the sum of the numbers at the odd positions (sum_{odd}).
Then the answer for an interval is just the value of sum_{even} for the root of the BST. When inserting/deleting nodes we can easily update the modified nodes. In total, the cost of adding/deleting a node will be $\mathcal{O}(log(n))$

Observation

Therefore, the expected complexity is $\mathcal{O}((n+q)\sqrt{n}\log(n))$

Problem A: Adjacent Rooks

Statement

Count the number of ways of placing n rooks in a $n \times n$ chessboard such that no two rooks are in the same column or file, and such that exactly k pairs of rooks are diagonally adjacent.

Constraints: $1 \leq n \leq 1000$; $0 \leq k \leq n-1$; $1 \leq t \leq 4000$.

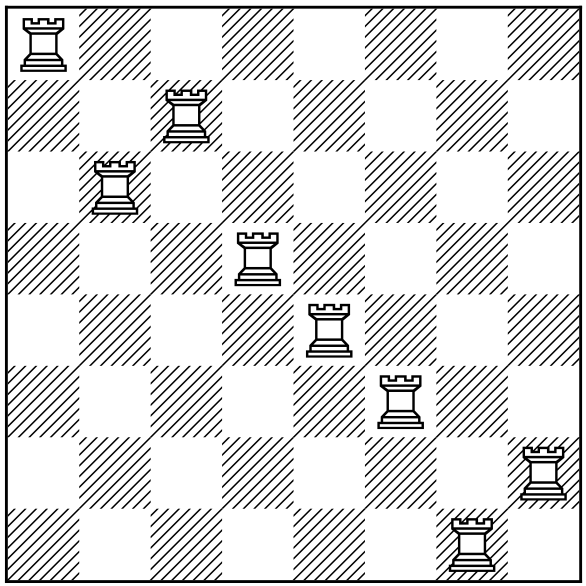
Author: Enric Rodríguez

- Number of solves: 56
- First team to solve: Peking U: Inverted Cross (Kong, Pan, Zhou)

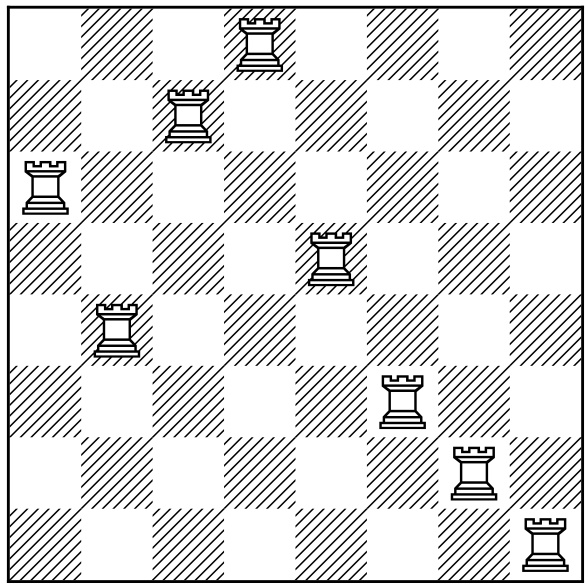
Problem A - Adjacent Rooks

Observation

Each disposition of the n-rooks in the chessboard corresponds to a permutation of $\{1, \ldots, n\}$, so the problem is equivalent to counting the number of permutations with k pairs of adjacent consecutive numbers.



$$(1, \underbrace{3, 2}, \overbrace{4, \underbrace{5, 6}}, \underbrace{8, 7}) \rightarrow k = 4$$



$$(\underbrace{4, 3}, 1, 5, 2, \overbrace{6, \underbrace{7, 8}}) \rightarrow k = 3$$

Problem A - Adjacent Rooks

It can be solved using a recurrence for the generating function of the sequence, found in https://oeis.org/A001100. The official solution is longer but easier to understand.

Problem A - Adjacent Rooks

It can be solved using a recurrence for the generating function of the sequence, found in https://oeis.org/A001100. The official solution is longer but easier to understand.

Definition (Cool pair)

To simplify the notation, let's call a pair of adjacent consecutive numbers in the permutation a *cool pair*.

We need to find the number of permutations of size n with k cool pairs.

Problem A - Adjacent Rooks

We define the following sequences:

Problem A - Adjacent Rooks

We define the following sequences:

- $A(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that start with n.

Problem A - Adjacent Rooks

We define the following sequences:

- $A(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that start with n.
- $B(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, none of which are $(n, n-1)$ or $(n-1, n)$, and that start with n.

Problem A - Adjacent Rooks

We define the following sequences:

- $A(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that start with n.
- $B(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, none of which are $(n, n-1)$ or $(n-1, n)$, and that start with n.
- $C(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that end with n.

Problem A - Adjacent Rooks

We define the following sequences:

- $A(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that start with n.
- $B(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, none of which are $(n, n-1)$ or $(n-1, n)$, and that start with n.
- $C(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that end with n.
- $D(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, none of which are $(n, n-1)$ or $(n-1, n)$, and that end with n.

Problem A - Adjacent Rooks

We define the following sequences:

- $A(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that start with n.
- $B(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, none of which are $(n, n-1)$ or $(n-1, n)$, and that start with n.
- $C(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that end with n.
- $D(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, none of which are $(n, n-1)$ or $(n-1, n)$, and that end with n.
- $E(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that neither start nor end with n.

Problem A - Adjacent Rooks

We define the following sequences:

- $A(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that start with n.
- $B(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, none of which are $(n, n-1)$ or $(n-1, n)$, and that start with n.
- $C(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that end with n.
- $D(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, none of which are $(n, n-1)$ or $(n-1, n)$, and that end with n.
- $E(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, one of which is $(n, n-1)$ or $(n-1, n)$, and that neither start nor end with n.
- $F(n, k) :=$ number of permutations of $\{1, \ldots, n\}$ with k cool pairs, none of which are $(n, n-1)$ or $(n-1, n)$, and that neither start nor end with n.

Problem A - Adjacent Rooks

From the previous definitions, we see that the answer of the problem is

$$A(n,k) + B(n,k) + C(n,k) + D(n,k) + E(n,k) + F(n,k) =$$
$$= 2A(n,k) + 2B(n,k) + E(n,k) + F(n,k)$$

so we just need a way to calculate these efficiently. We are going to show that each of these sequences follows a recurrence in terms of the others.

Problem A - Adjacent Rooks

- $A(n, k) = A(n-1, k-1) + B(n-1, k-1)$

  If the permutation starts with n and has the cool pair $(n, n-1)$, then the remaining permutation after deleting the first element has length $n-1$, has $k-1$ cool pairs and starts with $n-1$.

Problem A - Adjacent Rooks

- $A(n,k) = A(n-1,k-1) + B(n-1,k-1)$
- $B(n,k) = C(n-1,k) + D(n-1,k) + E(n-1,k) + F(n-1,k)$

  If the permutation starts with n and doesn't have the cool pair $(n, n-1)$, then the remaining permutation after deleting the first element has length $n-1$, has k cool pairs and doesn't start with $n-1$.

Problem A - Adjacent Rooks

- $A(n,k) = A(n-1,k-1) + B(n-1,k-1)$
- $B(n,k) = C(n-1,k) + D(n-1,k) + E(n-1,k) + F(n-1,k)$
- $C(n,k) = C(n-1,k-1) + D(n-1,k-1)$

  Same reasoning as A.

Problem A - Adjacent Rooks

- $A(n,k) = A(n-1,k-1) + B(n-1,k-1)$
- $B(n,k) = C(n-1,k) + D(n-1,k) + E(n-1,k) + F(n-1,k)$
- $C(n,k) = C(n-1,k-1) + D(n-1,k-1)$
- $D(n,k) = A(n-1,k) + B(n-1,k) + E(n-1,k) + F(n-1,k)$

  Same reasoning as B.

Problem A - Adjacent Rooks

- $A(n,k) = A(n-1,k-1) + B(n-1,k-1)$
- $B(n,k) = C(n-1,k) + D(n-1,k) + E(n-1,k) + F(n-1,k)$
- $C(n,k) = C(n-1,k-1) + D(n-1,k-1)$
- $D(n,k) = A(n-1,k) + B(n-1,k) + E(n-1,k) + F(n-1,k)$
- $E(n,k) = A(n-1,k) + C(n-1,k) + E(n-1,k) + B(n-1,k-1) + D(n-1,k-1) + E(n-1,k-1) + 2F(n-1,k-1)$

  We know that n is in the middle of the permutation, and that $n-1$ is next to it. The first 3 terms correspond to the case where the element on the other side is $n-2$, and the last 4 terms correspond to the case where it is another element.

Problem A - Adjacent Rooks

The last one is a bit longer...

$$
\begin{aligned}
F(n,k) = {} & (k-1)\,(B(n-1,k+1) + D(n-1,k+1) + F(n-1,k+1)) + \\
& + k\,(A(n-1,k+1) + C(n-1,k+1) + E(n-1,k+1)) + \\
& + (n-2-k)\,(A(n-1,k) + C(n-1,k)) \\
& + (n-3-k)\,(A(n-1,k) + C(n-1,k) + E(n-1,k)) + \\
& + (n-4-k)F(n-1,k)
\end{aligned}
$$

The basic idea is looking at a permutation of $\{1,\ldots,n-1\}$ and counting all the possible ways to insert n in the middle of the permutation (and not next to $n-1$). For example, the terms of the first line correspond to the number of ways of inserting n in the middle of a cool pair of a permutation that doesn't contain $(n-1,n-2)$ or $(n-2,n-1)$.

Problem A - Adjacent Rooks

As $A(n,k) = C(n,k)$ and $B(n,k) = D(n,k)$, in practice we just need to compute $A(n,k)$, $B(n,k)$, $E(n,k)$ and $F(n,k)$. These values can be either precomputed or dynamically stored. The solution runs in $\mathcal{O}(nk)$.

Problem F: Friendship Circles

Statement

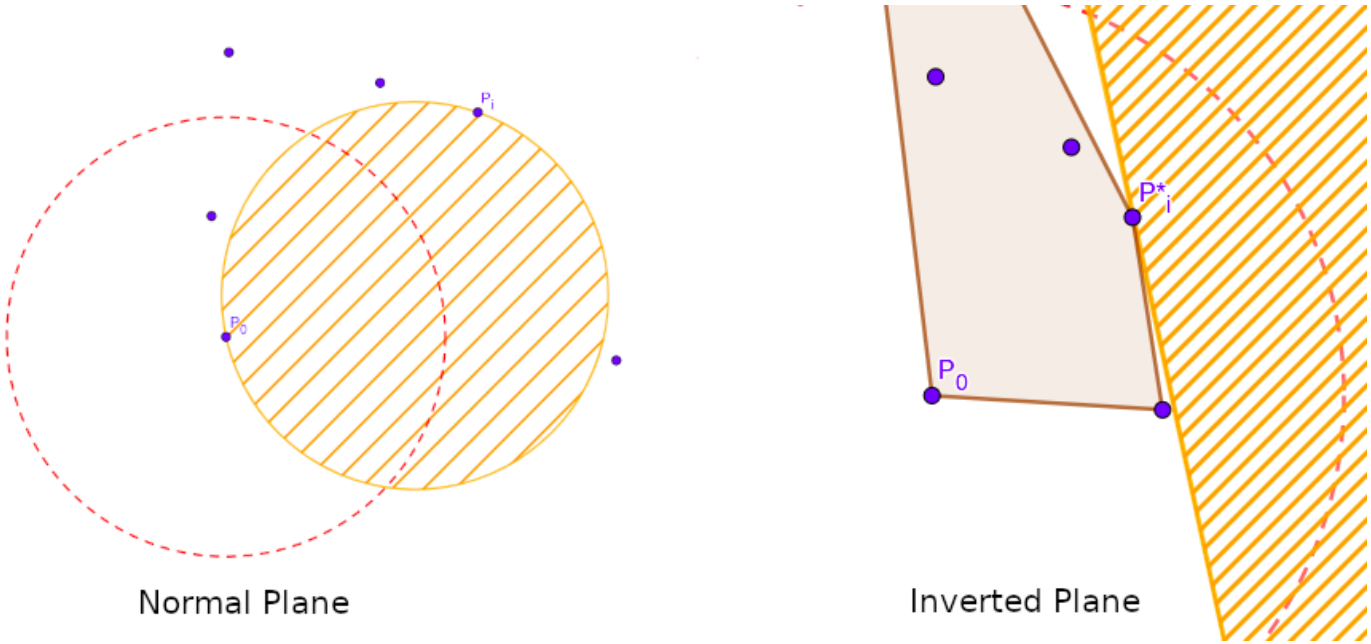
Let $p_0, p_1, \ldots, p_{n-1}$ be n points in the plane. We say that two points are *friends* if one can draw a circle that contains both points in its interior and all the other $n-2$ points in its exterior. Print the indices of the points that are friends with p_0.

Author: Félix Moreno Peñarrubia

- Number of solves: 30
- First team to solve: Peking U: Inverted Cross (Kong, Pan, Zhou)

Problem F: Friendship Circles

We compute a geometric inversion centered at p_0.



Any circle through p_0 and p_i gets transformed into a line passing through p_i^*, and the interior of the circle into the half-plane of the line not containing $p_\infty^* \equiv p_0$. Such a half-plane that does not contain any other point exists if and only if p_i^* lies on the convex hull of $p_1^*, p_2^*, \ldots, p_n^*, p_0$, so we just have to compute that convex hull.

With this solution, you have to choose the inversion radius carefully or you will have precision issues.

$R = \sqrt{d_{\min} \cdot d_{\max}}$ works fine, where $d_{\min}$ and $d_{\max}$ are the minimum and maximum distances to p_0.

Problem F: Friendship Circles

There is an alternative solution.

Theorem

The friends of p_0 are its neighbors in the Voronoi diagram.

Proof

We can see that the center of the circle passing through p_0 and p_i must be in the edge between the two respective cells in the Voronoi diagram, as otherwise there would be another point inside. Conversely, any circle centered on the edge works.

We could compute the entire Voronoi diagram (and it would allow us to compute all the friendship pairs), but it is hard to implement.
Fortunately, we only need p_0's cell, and it can be computed by doing a halfplane intersection on the perpendicular bisectors between p_0 and all the other points.
It is interesting to note that the halfplane intersection can be computed via its projective dual, which is convex hull. So we end up with the same problem if we use inversive duality or if we use projective duality.

Problem D: Display of Springs

Statement

You are given n linear functions $f_1, f_2, \ldots, f_n$ in the domain $\{1, 2, \ldots, 10^5\}$. You can compare the values of two functions in a point. You are asked to answer queries for the which function has maximum value at a point using few comparisons, but you can do many comparisons before you start answering queries.

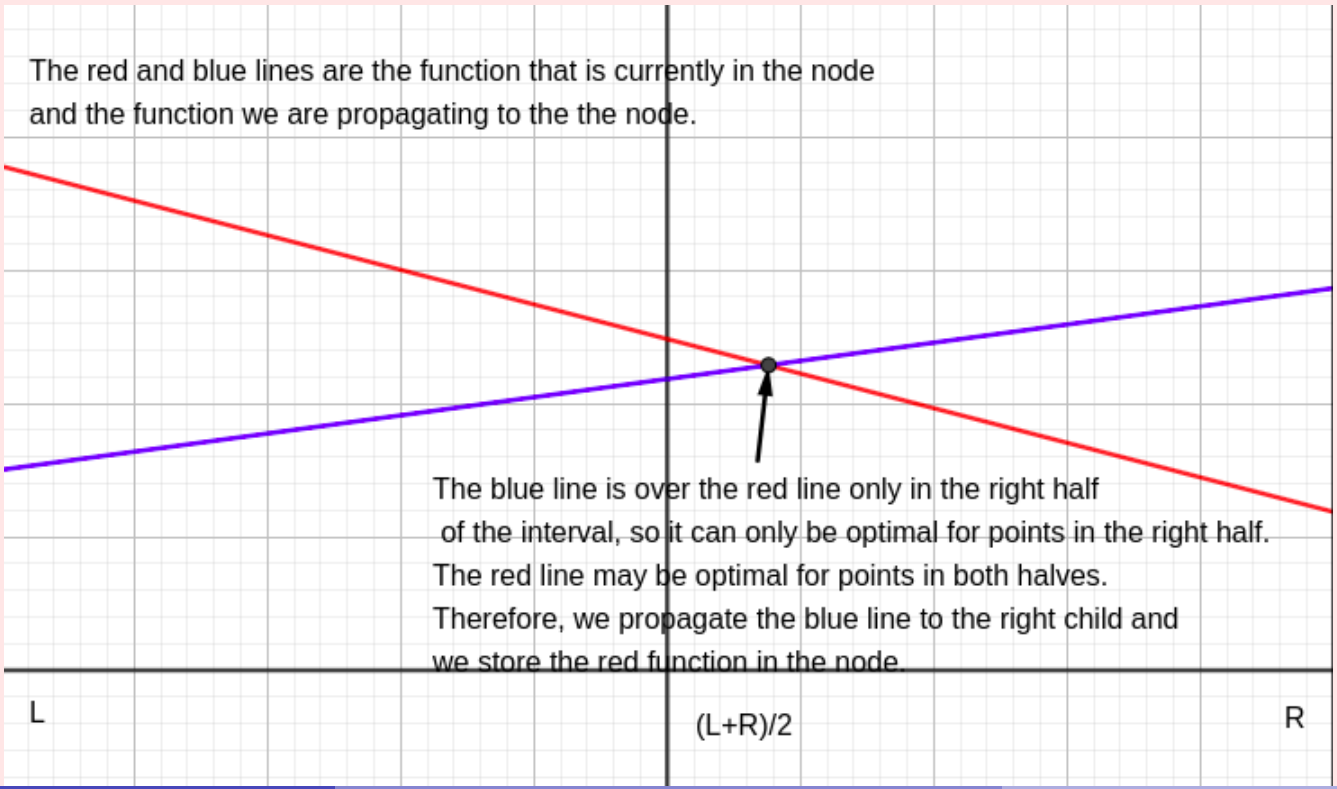
Author: Félix Moreno Peñarrubia

- Number of solves: 26
- First team to solve: NNSU: Almost Retired (Danilyuk, Kalinin, Ryabchikova)

Problem D: Display of Springs

Li Chao Tree

We construct a segment tree over the points in the domain. We will store one function in each node, in a way so that the maximum function for a point x will always be in the path from the root to the leaf corresponding to x. The nodes are initialized with $-\infty$ function, and we add the functions one by one with the following propagation rule:



Problem D: Display of Springs

The Li Chao Tree data structure basically does what is asked in the statement.
For this problem, we have to notice that we don't need to compute the explicit intersection point of the two lines, we can just compare the values at the one endpoint of the interval and the midpoint to see in which half the intersection point lies.

Problem H: Hackerman

Statement

Given three recurrences $(x_n), (y_n), (z_n)$, with unknown seeds x_0, y_0, z_0 and prime numbers lists $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$. Public key of k-th user is $n = pqr$, with $p = \mathcal{X}[x_k], q = \mathcal{Y}[y_k], r = \mathcal{Z}[z_k]$.
Compute the prime factors of the u-th user and the v-th user.

Author: Izan Beltran

- Number of solves: 15
- First team to solve: Past Glory (Korotkevich)

Problem H: Hackerman

Observation

The primes are really big ($> 10^{30}$), so we can't just try factoring.
Let's analyze the recurrences,

$$x_n = 11x_{n-1} + 7 \mod 611953 \qquad \forall n > 0$$
$$y_n = 13y_{n-1} + 5 \mod 746773 \qquad \forall n > 0$$
$$z_n = 53z_{n-1} + 3 \mod 882389 \qquad \forall n > 0$$

At some point those recurrences will cycle, this means some primes will be shared among some users, so we can try finding two users with common factors and compute $\gcd(n_k, n_t)$.
Notice that all numbers involved are primes.

Problem H: Hackerman

Observation

With the generic recurrence $w_n \equiv aw_{n-1} + b \pmod{m}$:

$$\begin{aligned} w_n &\equiv aw_{n-1} + b \equiv a(aw_{n-2} + b) + b \equiv a^2 w_{n-2} + b + ab \equiv \\ &\equiv a^2(aw_{n-3} + b) + b + ab \equiv a^3 w_{n-3} + b + ab + a^2 b \equiv \\ &\equiv \cdots \equiv a^n w_0 + b\sum_{i=0}^{n-1} a^i \equiv a^n w_0 + b(a^n - 1)(a-1)^{-1} \equiv \\ &\equiv a^n(w_0 + b(a-1)^{-1}) - b(a-1)^{-1} \equiv \\ &\equiv a^n\alpha + \beta, \quad \text{with } \alpha = w_0 + b(a-1)^{-1},\ \beta = -b(a-1)^{-1} \text{ constants} \end{aligned}$$

In particular, $w_0 \equiv \alpha + \beta$. Now that we know that w_n is a linear function of a^n, we just need to find out how a^n cycles.

Problem H: Hackerman

Observation

We know m is prime, so by Fermat's little theorem, $a^{m-1} \equiv 1$.
This implies $w_{m-1} \equiv \alpha + \beta \equiv w_0$, and the cycle size must be either $m-1$ or one of its divisors.
We also found out that the recurrence cycles start from the first element, and seeds doesn't matter for the cycle size, so we can use an arbitrary seed and compute cycle sizes for $(x_n), (y_n)$ and (z_n).

Problem H: Hackerman

Observation

We'll denote the lengths of the three cycles as C_x, C_y, C_z.
Note these cycle lengths are pairwise coprime, for the chosen modulus in the recurrences, so we can solve systems of congruences.

Key observation

$i \equiv j \mod C_x \implies x_i = x_j \implies$ i-th and j-th users share the smallest prime factor p.
Same applies with the other recurrences,
$i \equiv j \mod C_y \implies$ i-th and j-th users share the medium prime factor q.
$i \equiv j \mod C_z \implies$ i-th and j-th users share the biggest prime factor r.

Problem H: Hackerman

Observation

If we want to get the p of u-th user, query for n_u.
Then query for n_k for some k s.t.

$$\begin{aligned} k &\equiv u \mod C_x \\ k &\not\equiv u \mod C_y \\ k &\not\equiv u \mod C_z \end{aligned}$$

We use $k = u + C_x$.
Now we can compute $p = \gcd(n_u, n_k)$.
We can follow this strategy, but we will need 6 queries to get the factors of both u-th and v-th users.

Problem H: Hackerman

Key observation

We can do the same with only 5 queries (even with 4):
After queries for n_u and n_v, we can get at the same time p_u and q_v with only one more query.
Query for n_k for some k s.t.

$$\begin{aligned} k &\equiv u \mod C_x \\ k &\equiv v \mod C_y \\ k &\not\equiv u \mod C_z \\ k &\not\equiv v \mod C_z \end{aligned}$$

You can find this solution by using Chinese remainder theorem. If the inequalities are not satisfied, just keep adding $C_x C_y$ to k until it's satisfied.

Problem H: Hackerman

Key observation - continued

Then query for n_k, after this (assuming u is not congruent to v mod C_x, C_y and C_z), we have $p_u = \gcd(n_u, n_k)$ and $q_v = \gcd(n_u, n_k)$

Solution

The previous method works for most inputs, except when u and v are congruent respect to some of the modulus.
In this cases, we can just choose another factors and do a similar strategy to the one shown above.
If they share 2 or 3 factors, the task is even easier, but we need to deal with these cases with some if-else logic.

Problem G: Game on a tree

Statement

m friends friends are playing a card game. There are cards of c different color, each color has an associated number of points p_i. We define the score of a deck as the gcd of the points of its cards. The game has m turns (queries).

We have a graph of n nodes. Each node contains a card of one of the colours. In the i-th turn, player i chooses two vertexes v_i, u_i and for each card on the path from v_i to u_i, each player draws a card of that same colour. If he already had a card of that colour he discards both. Then that player gains as many points as the sum of the score of each player's deck. Finally, he gets to change the color of the card of a node.

Compute the number of points of each player at the end of the game.

Author: Max Balsells

- Number of solves: 14
- First team to solve: NNSU: Almost Retired (Danilyuk, Kalinin, Ryabchikova)

Problem G: Game on a tree

Key observation 1

We can represent each players deck as a number from 0 to $2^c - 1$, where in the binary representation of that number: $a = a_0, a_1, \ldots, a_{c-1}$, a_i takes value 1 if the player has a card of color i and 0 otherwise. With this observation, drawing a card of color i is equivalent to perform $2^i \oplus a$.

Notation

Let *color(v)* be the color of the card on node v. Let $v_{i,j} \in V_i$ be the nodes in the path between the two given nodes v_i, u_i in the i-th query. We define $q_i = \bigoplus_{v_{i,j} \in V_i} 2^{color(v_{i,j})}$.

Key observation 2

With this notation and the previous observation, drawing each card in the part from v_i to u_i is equivalent to perform the xor with q_i.

Problem G: Game on a tree

observation 3

In each turn, we should update each person's deck by doing the xor with q_i and compute the sum of the points of each player's deck. However, since $\oplus$ is associative, we can be working with the original deck of each player all the time; instead of updating each player's deck in each round, in the i-th query we can just compute the sum of each player's original deck after doing xor with $\bigoplus_{j=0}^{i} q_j$. Lets define $\widehat{q_i} = \bigoplus_{j=0}^{i} q_j$

Key observation 4

When doing queries we will be interested in knowing which would be the sum of the points of each deck after doing xor of each deck with $\widehat{q_i}$. In order to compute this efficiently, we can use Fast Walsh-Hadamard Transform ($\mathcal{FWHT}$).

Key observation 4 - continuation

We will precompute for each number the sum of the points of the deck of each player after doing xor with that number. That is, let d_i be the original deck for player i, and let points$[i]$ be the points for deck i, then, for a number *mask* we want to compute:

$$answer_{mask} = \sum_{i=0}^{m} \text{points}[d_i \oplus mask]$$

Let times$[i]$ be the number of times that the deck i appears among all friend's original deck, then:

$$answer_{mask} = \sum_{i=0}^{2^c-1} \text{times}[i] \cdot \text{points}[i \oplus mask] = \sum_{i \oplus j = mask} \text{times}[i] \cdot \text{points}[j]$$

$$= \{\text{times} \circledast \text{points}\}_{mask}$$

That is, we want to compute an xor convolution

Key observation 4 - continuation

We know that:

$$\mathcal{FWHT}\{\text{times}\}_{mask}\mathcal{FWHT}\{\text{points}\}_{mask} = \mathcal{FWHT}\{\{\text{times}\circledast\text{points}\}\}_{mask}$$

So we can compute $answer_{mask}$ as:

$$anwer_{mask} = \mathcal{FWHT}^{-1}\{\mathcal{FWHT}\{\text{times}\}_{mask}\mathcal{FWHT}\{\text{points}\}_{mask}\}_{mask}$$

Using this, we can compute $answer_{mask}$ for each $mask \in \{0, \ldots, 2^c - 1\}$ in $\mathcal{O}(2^c \cdot c)$, since the transform takes $\mathcal{O}(nlogn)$ with respect to its size.

Oobservation 5

All that is left at this points is to compute efficiently q_i and update the nodes value after each turn. In order to compute efficiently the xor sum in the path from v to u, we can split the query into two computing two xor sums, one from v to $lca(v, u)$ and another one from u to $lca(u, v)$, here $lca(u, v)$ denotes the lowest common ancestor of u and v.

Observation 6

In order to compute the xor sum from a node v to one of its ancestor, we can consider the heavy-light decomposition of the tree, and consider an xor Segment tree independent for each heavy-light component. Then, the xor sum of the path reduces to find the xor sum of intervals of a segment tree. Since there are at most $\mathcal{O}(log(n))$ heavy-light components from any node to the root and, finding the xor sum of a segment in a segment tree can be done in $\mathcal{O}(log(n))$. Hence, for each query we compute q_i in $\mathcal{O}(log^2(n))$

Solution

Now, putting all observations together we can find the solution of the problem. First, we precompute for each number from 0 to $2^c - 1$ the sum of the points of each deck after doing xor to each one with said number. We can do so by using $\mathcal{FWHT}$.
Then we consider the heavy-light decomposition of the tree and build an xor segment tree, with disjoint segments for each heavy-light component. For each query, we compute the xor sum of the path from v_i to u_i by computing the xor sum of the path from each node to their LCA. Then we update $\widehat{q_i} = \widehat{q_{i-1}} \oplus q_i$ and the answer to the query will be the value we precomputed before for $\widehat{q_i}$.
Finally, we have to update node w, we just have to update the corresponding leaf of the segment tree, updating the path to its root.

Observation 7

The complexity will thus be $\mathcal{O}(2^c \cdot (c + log(max\{p_i\})) + mlog^2(n))$

Problem B: Beautiful permutation

Statement

A permutation $a_0, a_1, \ldots, a_{n-1}$ of $0, 1, \ldots, n-1$ is said to be *beautiful* if the sequence $b_0, \ldots, b_{n-1}$ defined as $b_i = |a_i - i|$ is also a permutation of $0, \ldots, n-1$.

Given n, construct a beautiful permutation of n elements or say that it does not exist.

Author: Félix Moreno Peñarrubia

- Number of solves: 5
- First team to solve: Peking U: Inverted Cross (Kong, Pan, Zhou)

Problem B: Beautiful Permutation

Claim

If $n \equiv 2, 3 \pmod 4$, then the answer is NO.

Proof

Assume we have a beautiful permutation of a of n elements and consider the parity of the sum of the elements of b:

$$\frac{n(n-1)}{2} \equiv \sum_{i=0}^{n-1} b_i \equiv \sum_{i=0}^{n-1} (a_i + i) \equiv n(n-1) \equiv 0 \pmod 2$$

$$\implies n \equiv 0, 1 \pmod 4$$

We will see that if $n \equiv 0, 1 \pmod 4$ then the answer is YES.

Problem B: Beautiful Permutation

Is there a way to construct a beautiful permutation of larger n from beautiful permutations of smaller n? Yes!

Fact

If we have a beautiful permutation of n elements, we also have a beautiful permutation of $3n + 1$ elements.

One possible construction is: we first write the numbers $3n, 3n - 1, 3n - 2, \ldots, 2n + 1, 0$, then we write the permutation of n elements with $n + 1$ added to all its elements, and then we write the numbers $n, n - 1, \ldots, 1$.

Examples:

$$n = 1 : 0 \rightarrow 3\ 0\ 2\ 1$$

$$n = 4 : 3\ 0\ 2\ 1 \rightarrow 12\ 11\ 10\ 9\ 0\ 8\ 5\ 7\ 6\ 4\ 3\ 2\ 1$$

$$n = 5 : 4\ 2\ 0\ 3\ 1 \rightarrow 15\ 14\ 13\ 12\ 11\ 0\ 10\ 8\ 6\ 9\ 7\ 5\ 4\ 3\ 2\ 1$$

Problem B: Beautiful Permutation

The idea for the construction is that we can embed the permutation of n elements inside the permutation of $3n+1$ elements, shifting the values so that we still get the numbers $0, 1, \ldots, n-1$ when doing the absolute value difference, and then trying to get the remaining numbers $n, \ldots, 3n$ with the rest of the permutation.
With this table we can see that the construction works in general:

i	0	1	2	$\ldots$	$n-1$	n	$n+1, \ldots, 2n$	$2n+1$	$\ldots$	3n
a_i	$3n$	$3n-1$	$3n-2$	$\ldots$	$2n-1$	0	$\ldots$	n	$\ldots$	1
b_i	$3n$	$3n-2$	$3n-4$	$\ldots$	$n+2$	n	$0, \ldots, n-1$	$n+1$	$\ldots$	$3n-1$

But this doesn't work for constructing all $n \equiv 0, 1 \pmod 4$. It can only possibly work for $n \equiv 1 \pmod 3$.
Maybe we can find other constructions of the type $n \to 3n + c$ for $c \equiv 0, 2 \pmod 3$? But in order to be consistent with the $n \equiv 0, 1 \pmod 4$ condition we should have $c \equiv 1 \pmod 4$. Maybe $c = 5, 9$ work?

Problem B: Beautiful Permutation

Main idea

If we can construct beautiful permutations of $3n+1$, $3n+5$, $3n+9$ elements from a beautiful permutation of n elements, then we will be able to recursively construct a beautiful permutation for all $n \equiv 0, 1 \pmod 4$ from the base cases $n = 1, 5, 9$.

But it turns out the construction is not so straightforward for the $3n+5, 3n+9$ cases.

For the $3n+5$ case we can find a construction with a similar descending part at the right (but with some twist at the end). We have to separate cases between even and odd n.

n even case:

i	0	1	2	3	4	5	...	n	$n+1$
a_i	$3n+4$	$3n+1$	$3n+3$	$3n-1$	$3n+2$	$3n-3$	...	$2n+6$	1
b_i	$3n+4$	$3n$	$3n+1$	$3n-4$	$3n-2$	$3n-8$	...	$n+6$	n

i	$2n+3$	$2n+4$	$2n+5$	...	$3n$	$3n+1$	$3n+2$	$3n+3$	$3n+4$
a_i	$n+2$	$n+1$	n	...	5	4	3	0	2
b_i	$n+1$	$n+3$	$n+5$	...	$3n-5$	$3n-3$	$3n-1$	$3n+3$	$3n+2$

Problem B: Beautiful Permutation

(The odd n case is similar).
Finally we hace the $3n + 9$ case, which is the trickiest of them. For finding it, it helps to program a bruteforce solution and carefully filter its output to look for patterns that can be generalized. The pattern we found starts with a descending left segment (with a 2 thrown in between and some swaps) and the right segment repeats a pattern with period 4 (again with some tweaks that break the clean pattern a bit)

$n = 4 : 3\ 0\ 2\ 1 \rightarrow 20\ 19\ 18\ 17\ 16\ 14\ 2\ 15\ 13\ 12\ 9\ 11\ 10\ 7\ 4\ 8\ 5\ 0\ 3\ 6\ 1$

(some contestants found simpler solutions)