



IV Ukrainian Programming Cup 2025

Contest 7, div 1

Official solutions



Problem A - Around the table

Statement: There is a hidden circular permutation p of size $N \leq 1e5$. You have to find it using at most 60 queries.

Query: you can query by giving a permutation q of size N and the answer will be the set of all elements that appear in q before their neighbours from p .

Problem A - Around the table



Note: we will call the elements node and say we have an edge between two nodes if they are neighbours in the hidden permutation.

Note: for a specific node we can easily binary search its neighbours, but we would like to be able to parallelize that

⇒ We will try to split the nodes into 3 independent sets A, B, C (a set is independent if it doesn't contain neighbours)

After we have the sets A, B, C we can do 2 binary searches to find the neighbours of nodes in A in B ∪ C (first the first neighbour and then the last neighbour). Then, because each node in B has at most one neighbour in C (why? see slide 5) we can find the edges between B and C with only one binary search.

No of queries for binary searches: $3 * \log(N) \approx 48$

Problem A - Around the table



Computing set A :

Note: In order to be able to split the rest of the nodes into B and C , set A has to be maximal (see why on the next slide)

To build set A we will first ask random queries until each node has been included in at least one of the answers. In practice this takes only a few tries, out of 1000 experiments, the biggest number of tries required was 6 so we will use this number for further analysis of the solution.

Now we have 6 sets of independent nodes ($A_1, A_2, A_3, A_4, A_5, A_6$) which are not necessarily maximal so we will try make the first set maximal by including elements from the other 5 sets.

If we have a candidate A and we want to add the nodes from A_i into it we just have to make a query with all the nodes in A followed by all the nodes in A_i followed by the rest of the nodes.

No of queries to compute set A : **at most $6 + 5 = 11$**

Problem A - Around the table

Computing sets B and C:

Getting back to why set A had to be maximal: the fact that A is maximal means that there are no three nodes that are consecutive in p and are not included in A (otherwise we could add the middle element to A which contradicts with the fact that it is maximal)

\Rightarrow if we look at the set of nodes that are not in $A = A^{-1}$, each node in set A^{-1} has at most one neighbour that is also in $A^{-1} \Rightarrow$ we can split A^{-1} into B and C using only **one query**

\Rightarrow Total number of queries (in worst case): $3 * \log(N) + 11 + 1 \approx 60$



Problem B - Divisible Trees

Statement: Let A be a tree (connected undirected graph) and k be a positive integer. We define $k * A$ as the set of trees that contain k trees that are isomorphic with A and $k - 1$ extra edges to connect them. Then we say that a tree A divides a tree B iff there is a positive integer k such that $B \in k * A$.

Given a tree T , count the number of distinct (not isomorphic) trees that divide it.

Problem B - Divisible Trees



Let n be the size of tree T

Note that a tree A of size p can be a divisor of T only if p divides n

Not let's look at a fixed p and try to see how many trees of size p can be divisors of T . Let us choose a root for T and order its nodes by height. If we then iterate over the nodes from bottom to top, when the subtree of a node has size p , then this subtree should be an instance of the divisor tree and we can disconnect it from the rest of the graph. This way we can uniquely split the tree into subtrees of size p
 \Rightarrow there can be at most one divisor of size p

Now we only thing left to do is to check if the resulting subtrees are isomorphic (and we can do this by hashing their linearizations <https://codeforces.com/blog/entry/101010>)



Problem C - The Quest for the Sacred Grove

Statement: We are given a tree of N nodes and a permutation P of length N . Find the number of subsequences $[l, r]$ such that $P[l], P[l + 1], \dots, P[r]$ induces a connected subgraph.



Problem C - The Quest for the Sacred Grove

Let $[l, r]$ be a subsequence of the permutation.

$[l, r]$ is connected iff $(r - l + 1) - \text{num_of_edges} = 1$.

num_of_edges is the number of edges which have both extremities inside $[l, r]$.

The value $(r - l + 1) - \text{num_of_edges} \geq 1$ for every $[l, r]$.

We iterate over r . We update the prefix $[1 \dots r]$ with $+1$ and for every edge $(P[i], P[r])$ where $i < r$ we update prefix $[1 \dots i]$ with -1 . Now we have to find the minimum value and its frequency using a lazy segment tree.

Total complexity is $O(N \log(N))$.



Problem D - The Romanian Sieve

Statment: Given the algorithm on the right and T , find biggest N such that iters will be $\leq T$.

```
int64_t iters = 0;
for (int64_t i = 1; i <= n; i++) {
    for (int64_t j = i; j <= n; j += i) {
        max_div[j] = i;
        iters++;
    }
}
```



Problem D - The Romanian Sieve

We binary search for N . (for given constraints $N \leq 11 * 10^{11}$).

We compute $F(N) = N / 1 + N / 2 + \dots + N / (N - 1) + N / N$ in $O(\sqrt{N})$.

The expression N / i takes $O(\sqrt{N})$ distinct values for $i = 1, N$.

We add up N / i , for each $i \leq \sqrt{N}$.

Now for each $c < \sqrt{N}$ we count how many i 's exist s.t. $N / i = c$.

This is equal to $(N / c) - (N / (c + 1))$.

Total complexity is $O(\sqrt{N} * \log(N))$.



Problem E - Goddess of Olympus

Statement: We are given an array T of N integers. For Q queries, calculate the number of subsequences $[l, r]$ such that $\min(T[l], \dots, T[r]) = X$ and $\max(T[l], \dots, T[r]) = Y$.



Problem E - Goddess of Olympos

Let $F(X, Y)$ = number of subsequences with $\min(l, r) \geq X$ and $\max(l, r) \leq Y$.

Then the answer for a query is:

$Q(X, Y) = F(X, Y) - F(X + 1, Y) - F(X, Y - 1) + F(X + 1, Y - 1)$ by principle of inclusion exclusion.

Now every query $Q(X, Y)$ will be split into four queries of the F function. Now we will only answer queries of the F function.



Problem E - Goddess of Olympos

We can now sort the values in the array as pairs $P[i] = (T[i], i)$ in lexicographical order.

For a query $F(X, Y)$ we will find the range $P[a..b]$ such that $P[a].first \geq X$ and $P[b].second \leq Y$. For query $F(X, Y)$ only positions in $P[a..b]$ will be “active”.

Now we use MO's algorithm to solve queries for the ranges in array P .

Total complexity is $O((N + Q) * \sqrt{N})$.



Problem F - The Cypriote Mermaid

Statement: You are given a string S with 0, 1, and ?. Count in how many ways you can replace ? with 0 or 1 such that you can reduce the string to the empty string by applying the following operation any number of times:

Choose i s.t. $S[i] = S[i + 1]$ and delete them from the string.



Problem F - The Cypriote Mermaid

For every odd index, we flip its value. A string can be reduced to empty iff number of 1's is equal to number of 0's. Let $\text{freq}[c]$ = frequency of character c .

The answer is

$$\binom{\text{freq}[?]}{\frac{n}{2} - \text{freq}[0]}$$



Problem G - FS's Critical Concert

Statement: Count the total number of critical edges in all the labelled (not necessarily connected) graphs of size N .

Problem G - FS's Critical Concert

To compute the requested sum we are going to select an edge and count the number of graphs in which that edge is critical.

An edge (a - b) is critical in a graph where a and b are in different connected components and then we add edge (a - b). In order to not overcount we are going to iterate over the size of a's connected component.

$$result = \binom{n}{2} \cdot \sum_{i=1}^{n-1} con[i] \cdot all[n-i] \cdot \binom{n-2}{i-1}$$

where $all[i] =$ the number of labelled graphs of size $i = 2^{i \cdot (i-1)/2}$

$con[i] =$ the number of labelled connected graphs of size i

Problem G - FS's Critical Concert

The hard part is to compute $con[i]$ and in order to do that we'll subtract the number of disconnected graphs from the total number of graphs and we can count the number of disconnected graph in a similar way to how we count the graphs for a critical edge.

$$con[i] = all[i] - not_con[i] = all[i] - \sum_{j=1}^{i-1} \binom{i-1}{j-1} \cdot con[j] \cdot all[i-j]$$

$$\Rightarrow \frac{con[i]}{(i-1)!} = \frac{all[i]}{(i-1)!} - \sum_{j=1}^{i-1} \frac{con[j]}{(j-1)!} \cdot \frac{all[i-j]}{(i-j)!}$$

Problem G - FS's Critical Concert

We are going to write everything in this equation as a formal series (infinite polynomial) in order to simplify it.

$$\frac{con[i]}{(i-1)!} = \frac{all[i]}{(i-1)!} - \sum_{j=1}^{i-1} \frac{con[j]}{(j-1)!} \cdot \frac{all[i-j]}{(i-j)!}$$

$$p_{con} = 0 + \sum_{i=1}^{\infty} \frac{con[i]}{(i-1)!} \cdot X^i$$

$$p_{all} = 0 + \sum_{i=1}^{\infty} \frac{all[i]}{i!} \cdot X^i$$

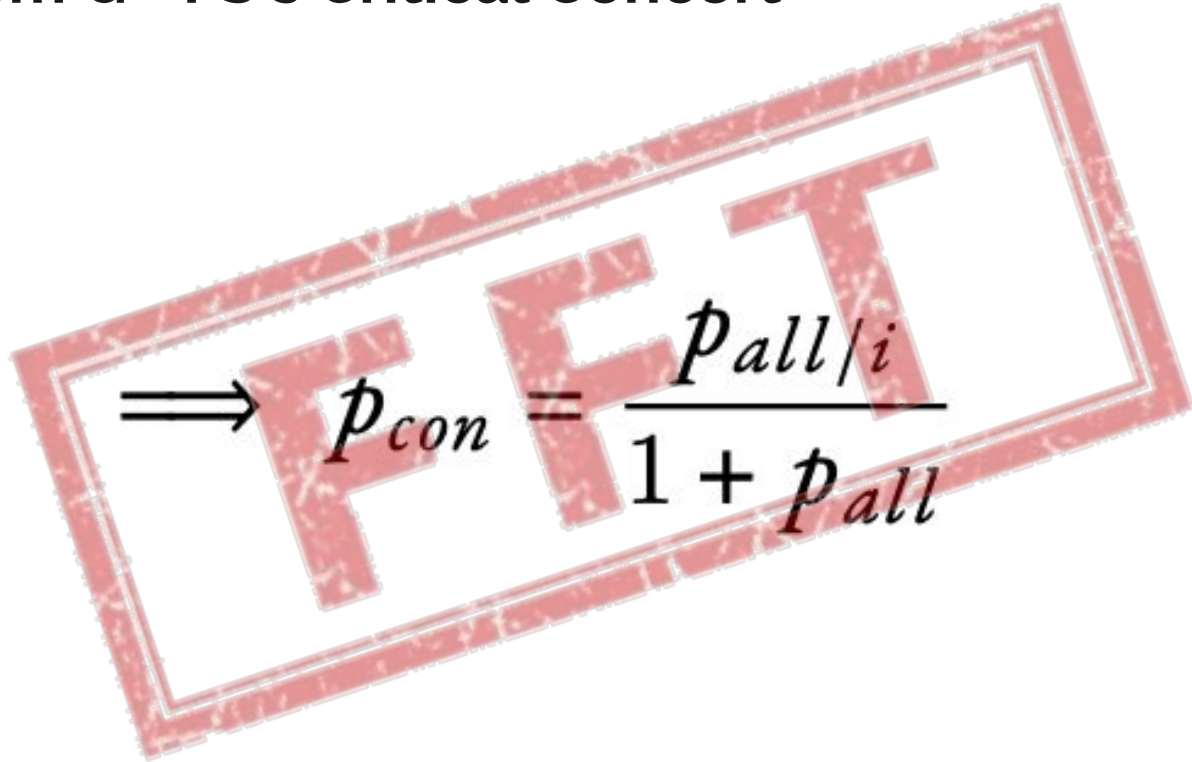
$$p_{all/i} = 0 + \sum_{i=1}^{\infty} \frac{all[i]}{(i-1)!} \cdot X^i$$



$$p_{con} = p_{all/i} - p_{con} \cdot p_{all}$$

$$\Rightarrow p_{con} = \frac{p_{all/i}}{1 + p_{all}}$$

Problem G - FS's Critical Concert


$$\Rightarrow p_{con} = \frac{p_{all/i}}{1 + p_{all}}$$



Problem G - FS's Critical Concert

Useful links:

1. source code: <https://pastebin.com/LtQLSukM>
2. CF blog post that was recommended during editorial:
<https://codeforces.com/blog/entry/103979>
3. CP Algorithms post: <https://cp-algorithms.com/algebra/polynomial.html>



Problem H - The Lottery WINNER

Statement: You are given a list of N strings $s[i]$. A string has digits, and at most 2 distinct letters. A character c covers string $s[i]$ if it appears in $s[i]$. Find the minimum number of characters which cover all the N strings.



Problem H - The Lottery WINNER

We brute force over all sets of digits and see what strings remain to be covered. For the remaining strings we will use the letters.

By having at most two letters we can build a graph where each letter is a vertex and every string is an edge.

On this graph we have to compute minimum vertex cover, which can be done in $O(2^{(n/2)})$, by computing the maximum clique of the complement graph and then taking n - its size.



Problem H - The Lottery WINNER

Let $G[\text{conf}]$ = the graph generated by strings whose digits appear in the subset conf . $G[\text{conf}]$ can be computed using SOS dp.

Now when we iterate over all subset of digits the graph we are interested computing minimum vertex cover on is $G[(1 \ll 10) - 1 - \text{conf}]$.

The solution does 2^{23} iterations.



Problem I - The Interview Problem

Statement: You are given a string containing brackets and digits. Each digit means that you have to delete as many brackets that appear (anywhere) before it in the string.

Check if you can delete brackets such that the result is a balanced bracketing.

Problem I - The Interview Problem

First we compute the number of closed and open brackets that we should delete

- n = total number of brackets
- $open$ = open brackets
- to_delete = the sum of all the digits in the string
- $\Rightarrow delete_open = open - (n - to_delete) / 2$
– we want to delete as many open brackets

If we n or to_delete don't have the same parity or the result is negative or bigger than to_delete , then there's no solution.

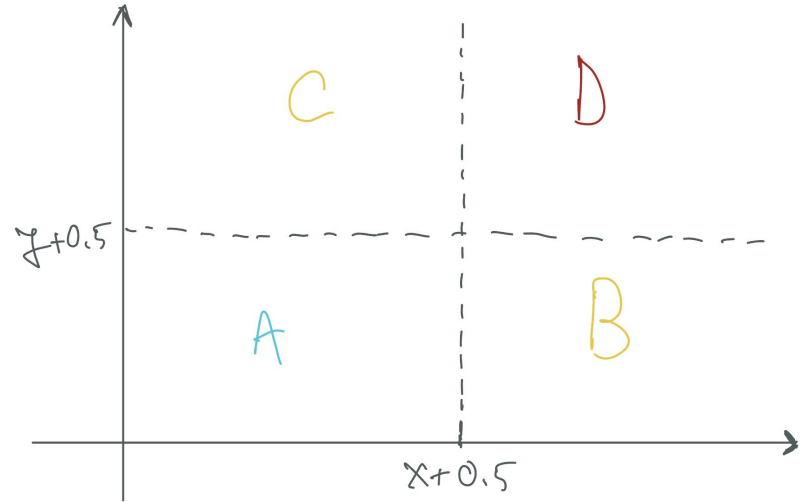
Now we just want to delete the closed brackets from the beginning of the string (and for that we can iterate over it from left to right) and the opened ones from the end, respecting the digits (and for that we can iterate over it from right to left).

Afterwards we just have to check if the result is balanced.

Problem J - Split the Picture

Statement: You are given N weighted points $x[i]$, $y[i]$, $s[i]$. For each $1 \leq x < N$ find y such that $\max(S(A), S(B), S(C), S(D)) - \min(S(A), S(B), S(C), S(D))$ is minimal.

$S(X)$ = sum of weights of points in quadrant X .





Problem J - Split the Picture

For fixed x , we can split y 's into 3 groups:

$Z1 = [1, y1 - 1] \rightarrow \min(A, B) = \min(A, B, C, D)$ and $\max(C, D) = \max(A, B, C, D)$

$Z2 = [y1, y2] \rightarrow \min(A, B) = \min(A, B, C, D)$ and $\max(A, B) = \max(A, B, C, D)$ OR $\min(C, D) = \min(A, B, C, D)$ and $\max(C, D) = \max(A, B, C, D)$

$Z3 = [y2 + 1, N - 1] \rightarrow \min(C, D) = \min(A, B, C, D)$ and $\max(A, B) = \max(A, B, C, D)$



Problem J - Split the Picture

We can binary search y_1 and y_2 , now the optimal y is either $y_1 - 1$, $y_2 + 1$, or the optimal y from $[y_1, y_2]$.

Let \min_{AB} be the minimum difference $A - B$ in $[y_1, y_2]$, \max_{AB} be the maximum difference $A - B$ in $[y_1, y_2]$. We define \min_{CD} and \max_{CD} similarly.

The optimal difference is calculated as $\max(\min_{AB}, -\max_{AB}, \min_{CD}, -\max_{CD})$. To simplify the implementation, one can see that \min_{CD} and \max_{CD} can be deduced from \min_{AB} and \max_{AB} .

Having this in mind, we can do a sweep line, moving points from the right side of x to the left side while maintaining a segment tree which stores in each interval the sum of weights of points on the left side, the right side, \min_{AB} and \max_{AB} .



Problem K - Adrian The Wonder Child

Statement: Given a tree of N nodes with 0-1 labels on the edges, we define a path to be K -alternating if the length longest sequence of equal labels is $\leq K$. Find the longest K -alternating path we can build by flipping at most M edges.



Problem K - Adrian The Wonder Child

Let's look at a fixed path. The edge labels will form a sequence similar to this:

00....00111...11100....00111...1111

Let $L[i]$ be the length of the i th maximal sequence of equal numbers. The minimum number of flip operations to make it k -alternating is $\sum \frac{L[i]}{(K + 1)}$



Problem K - Adrian The Wonder Child

Let's look at a fixed path. The edge labels will form a sequence similar to this:

00....00111...11100....00111...1111

Let $L[i]$ be the length of the i th maximal sequence of equal numbers. The minimum number of flip operations to make it k -alternating is $\sum \frac{L[i]}{(K + 1)}$



Problem K - Adrian The Wonder Child

We solve by centroid decomposition. Let X be a centroid. We will now find the longest K -alternating path passing through X . To do this we compute two arrays $C0$ and $C1$.

$Ci[j]$ = minimum cost required to obtain a path of length j which starts at X and goes downwards on an edge of color i .

We define the cost as a pair $(\text{rem}, \text{flips})$ where flips is the minimum number of flips needed to make it K alternating, and in case of equality we take the path that has the minimum length of the prefix of color i modulo $(K + 1) (\text{rem})$.

We can observe that both $C0$ and $C1$ are monotonic.

Problem K - Adrian The Wonder Child

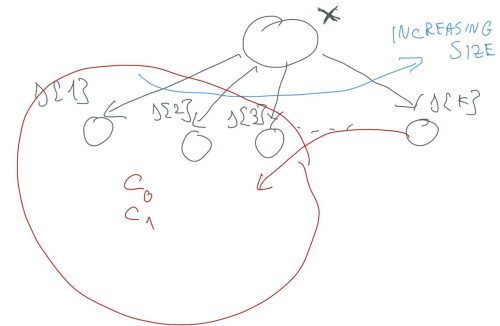
We will process X's descendants $s[1], s[2], \dots$ in increasing size.

For each descendant $s[k]$ we will compute such a vector C_p , where p is the color of the edge to $s[k]$.

We will now compute the longest K-alternating path which starts somewhere in subtrees $s[1], s[2], \dots, s[k-1]$ and ends somewhere in $s[k]$ by using the computed C_0 and C_1 so far of X.

This can be done using two-pointers technique.

Thus, the total complexity is $O(N \log(N))$.

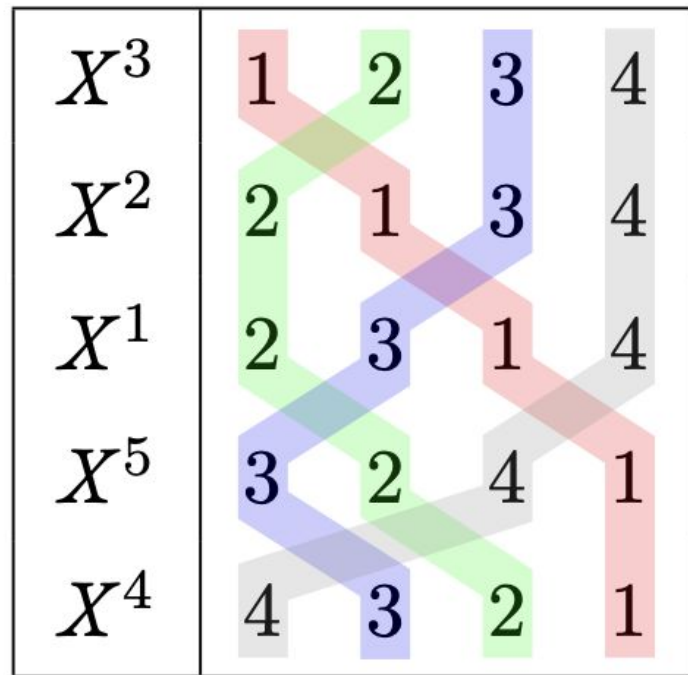


Problem L - Single Crossing

Statement: Given a set of N permutations of size M find out if there is a way to rearrange them so they form a single-crossing.

Single-crossing = any two elements a and b change their relative order at most once
(visually: they form at most one crossing)

<http://arxiv.org/pdf/2205.09092>



Problem L - Single Crossing

Part 1: Let's assume the permutations can form a single crossing and **find a way to arrange them**

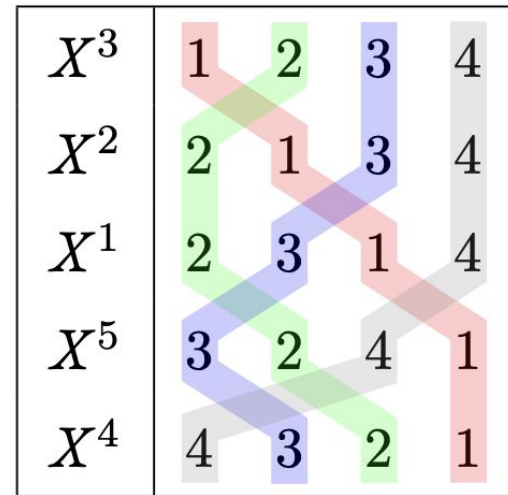
Furthermore let's assume we know which permutation will be first in the new order. If we relabel the elements of all the permutations according to the first one, we can then compare them by the number of inversions to obtain the final order.

⇒ there are only two correct orders (the one we find and its reversed)

How to find the first permutation?

We choose any permutation and assume it will be first and relabel all the permutations according to it. The one that has the most inversions (a.k.a. is farthest from the chosen permutation) will be either the first and the last permutation of the order.

Make note of the similarity with finding the diameter of a tree.



Problem L - Single Crossing

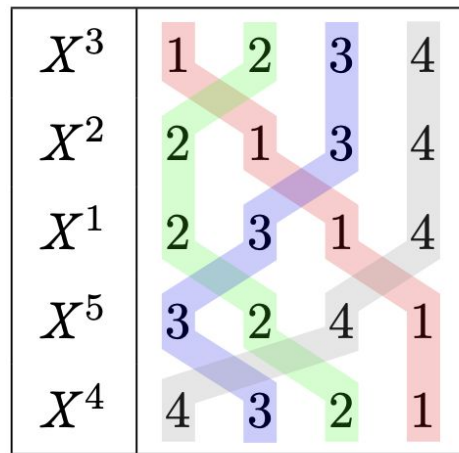
Part 2: Now that we have a way of sorting the permutations, we need to check if the order respects the single-crossing property

Let's define the following metrics over our set of permutations:

For any distinct $a, b \in \{1, 2, \dots, M\}$ we define:

$$d_{a,b}(i, j) = \begin{cases} 0 & \text{if } a \text{ and } b \text{ have the same relative order in both } X^i \text{ and } X^j \\ 1 & \text{otherwise} \end{cases}$$

This is a metric as it is positive (and 0 only for equal elements), it is symmetrical and it respects the triangle inequality.



Problem L - Single Crossing

Part 2: check if the order respects the single-crossing property

Because the previously defined functions are metrics, their sum is also a metric so we have the metric:

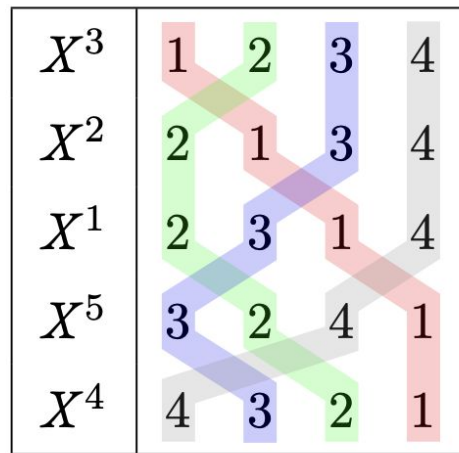
$$d(i, j) = \text{sum}(d_{a,b}(i, j)) \text{ for all } 1 \leq a < b \leq M$$

Note that this metric represents the relative number of inversions between two permutations so it can be computed in $O(M * \log(M))$

If (and only if) the permutations respect the single-crossing property, then we have:

$$d(i, j) = d(i, k) + d(k, j) \text{ for all } 1 \leq i < k < j \leq N$$

But it can be proved that it is enough to check the above condition for triplets of the form $(1, i, i + 1)$



Problem L - Single Crossing

Time complexity:

Part 1: $O(N * M * \log(N))$ (depending on the implementation)

Part 2: $O(N * M * \log(M))$

Total time complexity: $O(N * M * (\log(N) + \log(M)))$

