3A Contest

March 11, 2019

Filipp Rukhovich, Mikhail Tikhomirov

Moscow PreFinals Workshop 2019

A. Amidakuji

This problem can be reformulated in the following way. Given a permutation $p : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$ as a sequence of m transpositions (a_i, b_i), $i = 1, 2, \ldots, m$. We are to perform two types of operations:

- type A i: remove i-th swap from definition of p;
- type B i: find and print $p(i)$.

In fact, each transposition in our problem is of type $(v, v + 1)$, $1 \leqslant v \leqslant m - 1$, but our solution will work for any possible transpositions.

A. Amidakuji

The solution will be off-line.

p can be built in the following way. Let store p in an array $[1..n]$. Initialize p as $[1, 2, \ldots, n]$. Then, go through all transpositions in backward(!!!) order; for each transposition (a, b), swap elements $p[a]$ and $p[b]$.

Suppose that we should remove transpositions in the same order as they are given in the input. To remove first transposition (a_1, b_1), it's enough to swap $p[a_1]$ and $p[b_1]$ to maintain a correct p; then, the answer for query B_1 will be always $p[i]$.

So, the idea is to move all the transpositions which we has a query to remove at the beginning, possibly with changing of numbers of transpositions.

A. Amidakuji

Suppose that we want to move i-th (in given order) transposition to beginning of sequence. Note that the same permutation p can be calculated in the following way:

- $p = q = [1, 2, \ldots, n]$; q will be maintained as p^{-1};
- for $j = n, n-1, \ldots, i+1$, swap $p[a_j]$ and $p[b_j]$ (and maintain q);
- assign $u = p[a_i]$, $v = p[b_i]$;
- for $j = i-1, i-2, \ldots, 1$, swap $p[a_j]$ and $p[b_j]$ (and maintain q);
- swap $p[q[u]]$ and $p[q[v]]$ (and maintain q).

It means that we can move transposition (a_i, b_i) to the beginning and change it to $p[q[u]]$ and $p[q[v]]$, for p as a permutation generated by the same sequence of transpositions except i-th one and some u, v depending on the swaps from $i+1$ to n.

A. Amidakuji

We need to move up to m transpositions to the beginning, one-by-one; in fact, we need to sort them by the time of removing. To do that, we will use Merge Sort.

Suppose that we have two subarrays, L (left) and R (right) to merge. Then, suppose that p and q are as if we would emulate the process of calculaing p from n-th (current) transposition to the beginning of L, inclusively, and $q = p^{-1}$. It will be our invariant, and we will maintain it.

First of all, "cancel" all transpositions of L for p, q. Then, we will merge L and R by moving from left to right. When we write a new transposition to the answer, we perform a step of calculating p. When we add a new transposition from L into answer, we don't do anything; otherwise, we cancel this transposition for p, q, then find u, v for this transposition and add these (u, v) into answer with some flag.

A. Amidakuji

After merging, we should replace all u, v-s in transpositions with flag by restoring p, q from right to left in the united array; after that, just cancel this knowledge. Such merge works in linear time; so, the total complexity of our solution is $O(n\ log\ n + m)$.

B. Bees

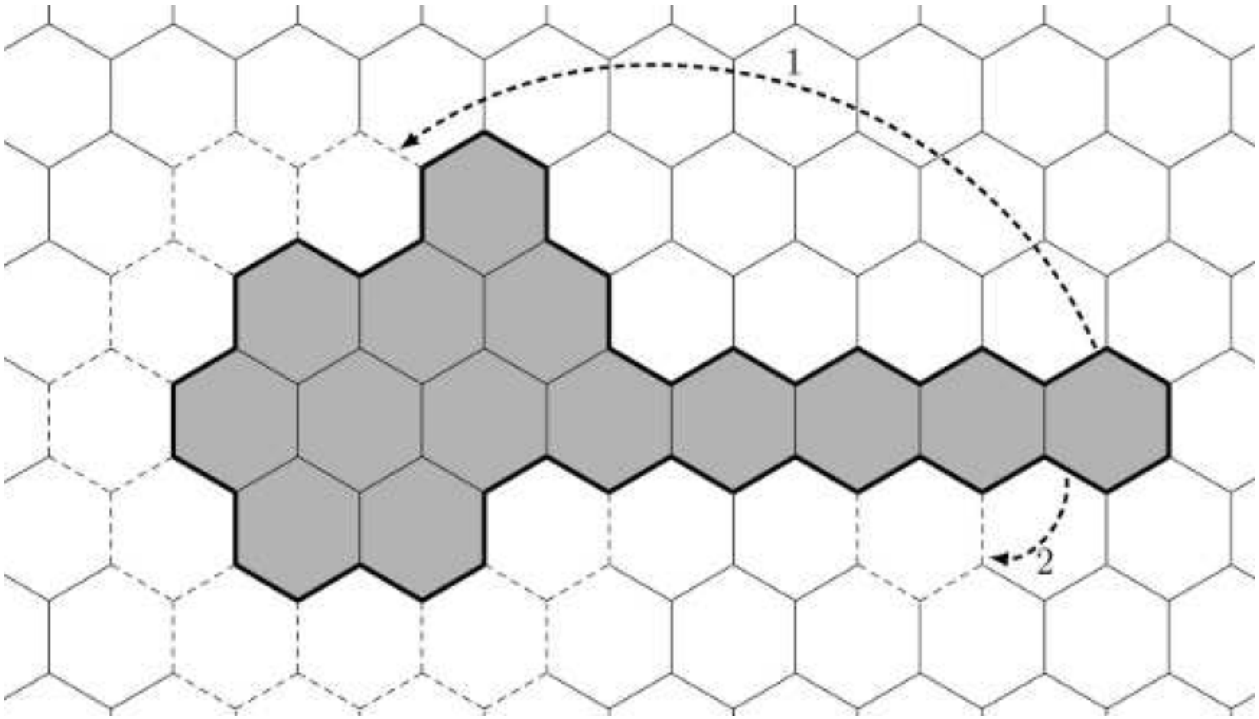
Our task is to construct a polygon on a hexagonal grid, which consists of n hexagonal cells and p edges, or state that such a polygon does not exist.

Let's fix the number of cells n and think what are the possible values of p such that a polygon exists. It is obvious that p must be even. It is also easy to see that p must be no bigger than $4n + 2$, that is the perimeter of the polygon consisting of n cells placed in a row. On the other hand, intuitively the polygon of minimal perimeter should look like a big hexagon. Formally, its perimeter is given by the formula $2\lceil\sqrt{12n - 3}\rceil$.

B. Bees

It turns out that these conditions are not only necessary but also sufficient. The algorithm could be as follows: we start from the polygon of n cells and the maximum perimeter. Then, as long as the perimeter of our polygon is greater than p, we delete one cell from one end of the polygon, and add one cell to the other end, building there a big hexagon layer by layer (see the arrow labeled by 1 on the following figure).

B. Bees



Such a move decreases the perimeter by 2, 4, or 6, so at the end we possibly would have to adjust the perimeter by 2 or 4, again by moving some cells (for instance rotating cells at the right end of the polygon decreases the perimeter by exactly 2; see the arrow labeled by 2 on the figure).

B. Bees

Moreover, depending on implementation details, one probably needs to treat specially polygons for small values of n and for perimeters close to the minimal value.

C. Cutting

Given string S and set of strings $M = \{M_1, M_2, \ldots, M_n\}$. On each step, we can erase from S some substring which is equal to some string in set M. How many steps do we need to make S empty?

Let for string T and some indices i, j:
$T[i..j] = T[i]T[i+1]\ldots T[j]$. We will use dynamic programming on subsegments. Let $S = S[0..l-1]$, $l = |S|$, and $dp[i][j]$, $0 \leqslant i < l$ is a minimal possible number of operations we need to perform $S[i..j]$ to empty string; if it's impossible that $dp[i][j]$ should be ∞.

C. Cutting

Note that $S[i..j]$ can be removed with last move as erasing some string $T \in M$ iff at least one of the following conditions is satisfied:

- $S[i..j] = T$; in this case, $dp[i][j] = 1$;
- $|T| < |S[i..j]| = j - i + 1$, and there exist such $k \in [0, |T|]$ that $S[i..i + k - 1] = T[0..k - 1]$, $S[j - (|T| - k) + 1][j] = T[k..|T| - 1]$, and $dp[i + k][j - (|T| - k)] \neq \infty$. By this way, $S[i][j]$ can be erased in $1 + dp[i + k][j - (|T| - k)]$.

For any i, j, T, all these variants can be checked in $O(|T|)$ time (in this time, one can easily find all common prefixes and suffixes of T and $S[i..j]$). Going through all $T \in M$ gives us a solution with total complexity $O(|S|^2 nm)$, for m as maximal possible length of string $T \in M$, with very-very small hidden constant.

Could you find a mistake?

C. Cutting

The problem is that it is possible way for erasing $S[i..j]$ to erase, for example, two (or more) non-intersecting substrings in such a way that concatenation of the remaining three or more pieces is equal to some $T \in M$.

Possible correct solution is to calculate 4-dimensional function $d[i][j][k][l]$ - what is a minimal number of operations we need to do to transfrom $S[i..j]$ to $M_k[1..l]$ (in 1-indexation), and also $dp[i][j]$ which was defined before.

If $d[i][j][k][l]$, then l-th letter of M_k should be equal to some symbol of $S[i][j]$, and this symbol should not be erased; if it will be j' then $d[i][j][k][l] = d[i][j' - 1][k][l - 1] + dp[j' + 1][l]$, and we should take a minimum among such values for all $j' \in [i, j]$ such that $S[j'] = M_k[l]$.

C. Cutting

Having calculated $d[i][j][k][l]$ for some i, j and all possible k, l, we can find $dp[i][j]$ as $1 + min\{d[i][j][k][|M_k|] \mid 1 \leqslant k \leqslant n\}$.

The solution works in $O(|S|^2 * nm^2)$ time, but with more very small hidden constant which can be made smaller using, for example, the fact that we can ignore inachievable states like states in which l is more that length of $S[i..j]$ etc.

D. Duel

Two players play a game with cards. Each of them has some cards. During each step, players independently choose one of their cards and show it to each other. Weaker card goes to retreat, the stronger goes back to the player who opened it. Strength relation is non-transitive and defined with matrix A. If some player has at least one card at some moment of time while other has not any cards then the player wins 1. If both players become out of cards at some moment of time that the result of both plays is 0.5. We are to define the price of such a game.

D. Duel

Let S and T be initial sets of cards for first and second players, respectively. Let $dp[S'][T']$, $S' \subseteq S$, $T' \subseteq T$ be a price of a similar game but in which first player starts a game with cards of subset S', and second — with subset T'. We will calculate dp in order of size of increasing of S', for fixed subset S' — in order of increasing of size of T'.

So, suppose that we are calculating $dp[S'][T']$. For each pair of cards (c, d), $c \in S'$, $d \in T'$, we see that if players choose these cards then the game will be reduced to the game with smaller subsets, and we can find a price of this game. Moreover, we can also find the $|S'| \times |T'|$-matrix $B(S', T')$, and in fact $dp[S'][T']$ is a price of a matrix game with matrix B. This game can be solved using simplex-method.

E. Elimination

There are N points on a plane, each of them is black or white. On each iteration, any point which sees at least one point of another color is eliminated. You are to find the number of iterations which will be performed and also see points of what color will be alive after these iterations.

If all the points lie on one line then it can be done in $O(n \log n)$ by sorting of all points and then by emulating the process by using a double linked list and maintaining the list of neighbouring pairs of points with different colours in $O(n)$ in total.

E. Elimination

But what if there are three points which don't lie on one line?

First of all, note that the number of iterations is always one. To prove it, suppose that after first iteration, there are at some black point A and some white point B which are still alive. Then, there are some points which don't lie on line AB; let C be the closest point to the line AB among them. It's easy to see that C sees both A and B; then, at least one of points A, B should be eliminated, and it gives as a contradiction. But how to understand points of what color will alive?

If number of points of some color, for example white, is less that $\sqrt{(n)}$ then we cah easily check in $O(n\ log\ n)$ for each white point, will it be alive. More concrete, let p is some white point. Sort all points y polar angle with respect to p, and in case of equal polar angle — by distance from p. It gives us the information which points see p. Having done such operation for each of white points, we have enough information to list all the points which will be alive after the elimination.

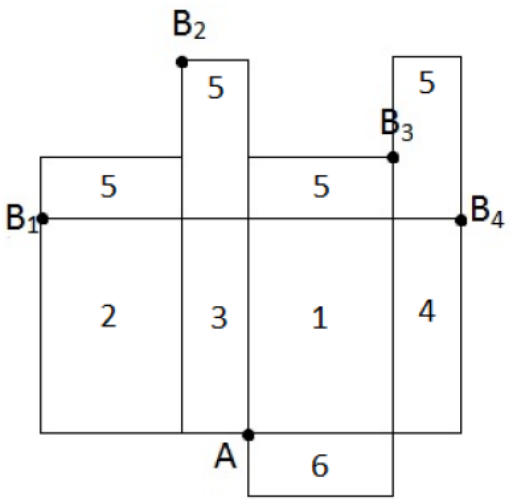
E. Elimination

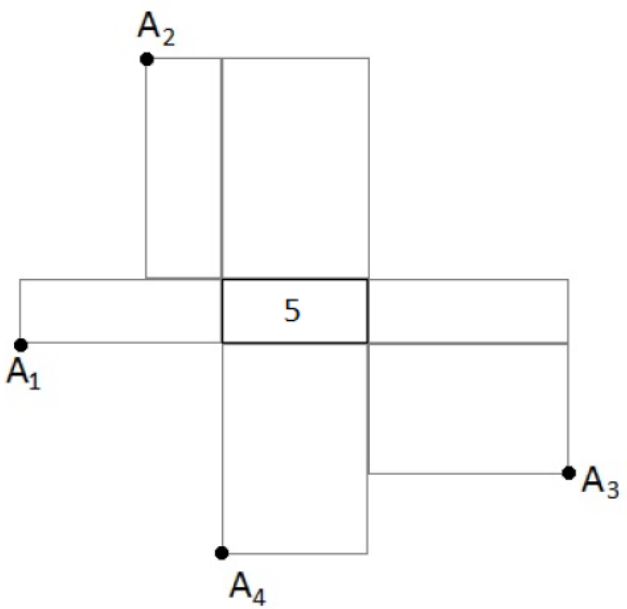
If numbers of points of some colour are too big then use the following heuristics. Consider a random point p; sort all other points by polar angle with respect to it. Then, we can see that points lying on neighbouring rays which angle less that π can eliminate each other; mark the points which we reveal to be eliminated using such a fact. After that, choose another point p' of different colour with minimal possible number of points on line connecting p' and p and so on. If all points will be marked then the answer is "Draw"; otherwise, if the marked points is not added by some number of iterations then just print color of any alive point.

F. Far Far Away Point

Given parallelepiped with length A, width B and height C, we are to find surface distance to the most distant (by surface) point from vertex of parallelepiped.

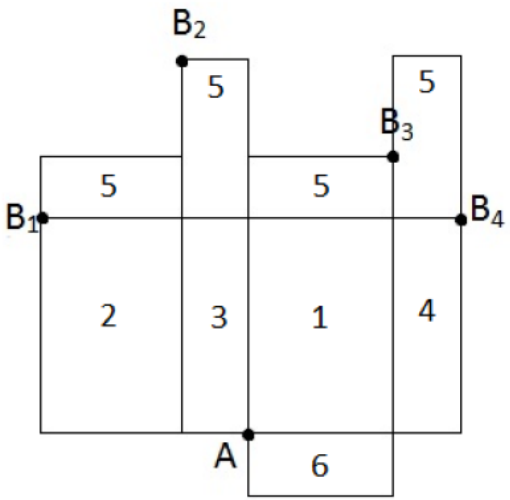
First of all, the answer can be not the opposite vertex. But we'll use it for finding the answer. Consider the unfolding of the parallelepiped - left part of the following picture.

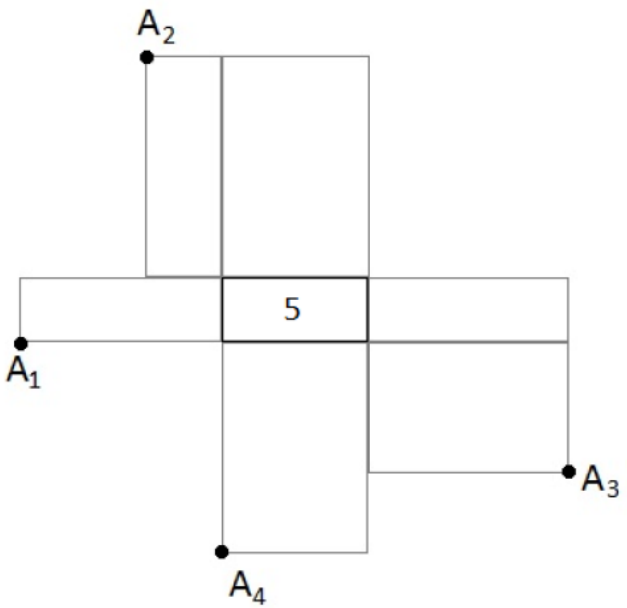




F. Far Far Away Point

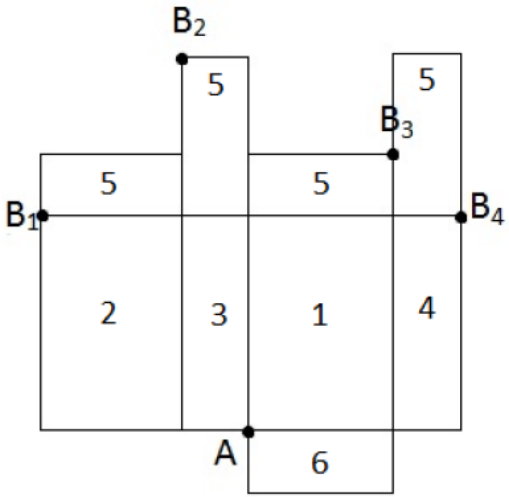
Here, A is an initial vertex, and B_1, B_2, B_3, B_4 are those points of unfolding in which the opposite vertex B occurs. Obviously, B_4 is the closest to A. It means the all points on faces $1, 2, 3, 4, 6$ are not farther than B_4, and the answer is in the face 5.

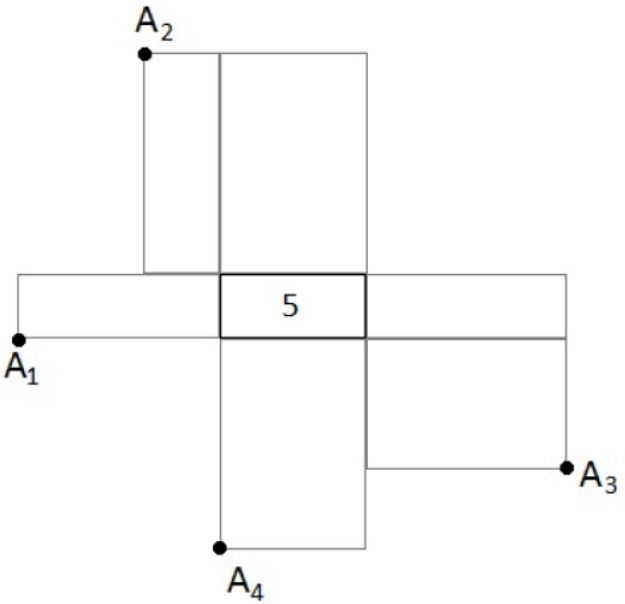




F. Far Far Away Point

Consider face 5 on more details (right picture). Let A_1, A_2, A_3, A_4 be possible positions of A on unfolding with respect to face 5. Then we are to find the point p on face 5 such that the minimal distance from p to one of point A_1, A_2, A_3, A_4 is as large as possible. It can be done using, for example, two nested ternary searches.





G. Grouping

For a given array of numbers $a_1, \ldots, a_n$ find the minimum of $\sum_{i=1}^{n} \min_{j=1}^{k} |a_i - b_j|$ over all possible sequences $b_1, \ldots, b_k$.

G. Grouping

Sort $a_1, \ldots, a_n$. Observe that for any b_j the indices of a_i such that $|a_i - b_j|$ is minimal for that i is a segment. Further, if a segment $a_l, \ldots, a_r$ is matched with the same number b_j, the optimal value of b_j is the median of that segment.

G. Grouping

Sort $a_1, \ldots, a_n$. Observe that for any b_j the indices of a_i such that $|a_i - b_j|$ is minimal for that i is a segment. Further, if a segment $a_l, \ldots, a_r$ is matched with the same number b_j, the optimal value of b_j is the median of that segment.

Let $dp_{i,j}$ be the answer for the prefix $a_1, \ldots, a_i$ for sequences $b_{\ldots}$ of j elements. We have

$$dp_{i,j} = \min_{i' \leqslant i} dp_{i',j} + cost(i', i),$$

where $cost(l, r)$ is $\min_x \sum_{i=l}^{r-1} |a_i - x|$. Note that $cost(l, r)$ can be computed in $O(1)$ with prefix sums.

G. Grouping

Sort $a_1, \ldots, a_n$. Observe that for any b_j the indices of a_i such that $|a_i - b_j|$ is minimal for that i is a segment. Further, if a segment $a_l, \ldots, a_r$ is matched with the same number b_j, the optimal value of b_j is the median of that segment.

Let $dp_{i,j}$ be the answer for the prefix $a_1, \ldots, a_i$ for sequences $b_{\ldots}$ of j elements. We have

$$dp_{i,j} = \min_{i' \leqslant i} dp_{i',j} + cost(i', i),$$

where $cost(l, r)$ is $\min_x \sum_{i=l}^{r-1} |a_i - x|$. Note that $cost(l, r)$ can be computed in $O(1)$ with prefix sums.

With this DP, the answer can be computed in $O(n^3)$.

G. Grouping

Let $opt(i, j)$ be the (leftmost) optimal value of i' in the relation for $dp_{i,j}$. One can show that $opt(i-1, j) \leqslant opt(i, j) \leqslant opt(i, j+1)$.

G. Grouping

Let $opt(i, j)$ be the (leftmost) optimal value of i' in the relation for $dp_{i,j}$. One can show that $opt(i-1, j) \leqslant opt(i, j) \leqslant opt(i, j+1)$.

Let us find $opt(i, j)$ (and $dp_{i,j}$) by increasing of $i - j$. The case $i = j$ is trivial. Otherwise, explicitly try all candidates of $opt(i, j)$ between $opt(i-1, j)$ and $opt(i, j+1)$. Since these intervals do not intersect for different values of i and j, the total time needed to process all states with fixed $i - j$ is $O(n)$, hence the total complexity is $O(n^2)$.

H. Hidden Triangles

We put n triangles on the plane in order. Determine which triangles are not completely covered in the end.

H. Hidden Triangles

Let us construct a planar graph G with vertices in all triangle vertices and side intersections. It contains $O(n^2)$ vertices and edges, and can be constructed in $O(n^2 \log n)$ by explicitly intersecting all sides, and sorting intersection points lying on each side.

H. Hidden Triangles

Let us construct a planar graph G with vertices in all triangle vertices and side intersections. It contains $O(n^2)$ vertices and edges, and can be constructed in $O(n^2 \log n)$ by explicitly intersecting all sides, and sorting intersection points lying on each side.

Obtain the dual H of this graph: vertices of H are faces of G, and edges connect neighbouring faces. Note that there could be several connected components in G; disconnected components are either disjoint, or one of them is completely contained inside a face of another.

H. Hidden Triangles

Let us construct a planar graph G with vertices in all triangle vertices and side intersections. It contains $O(n^2)$ vertices and edges, and can be constructed in $O(n^2 \log n)$ by explicitly intersecting all sides, and sorting intersection points lying on each side.

Obtain the dual H of this graph: vertices of H are faces of G, and edges connect neighbouring faces. Note that there could be several connected components in G; disconnected components are either disjoint, or one of them is completely contained inside a face of another.

Now, for each face we need to know the largest number of a triangle covering it; straightforward $O(n^3)$ approach is too slow.

H. Hidden Triangles

Instead, perform a DFS of the dual graph H starting from any face. During the DFS, maintain the set of triangle indices covering the current face.

H. Hidden Triangles

Instead, perform a DFS of the dual graph H starting from any face. During the DFS, maintain the set of triangle indices covering the current face.

Observe that moving to an adjacent edge (= crossing the border of a triangle) results in adding/removing a single triangle in the set.

H. Hidden Triangles

Instead, perform a DFS of the dual graph H starting from any face. During the DFS, maintain the set of triangle indices covering the current face.

Observe that moving to an adjacent edge (= crossing the border of a triangle) results in adding/removing a single triangle in the set.

The resulting information is enough to obtain the answer. Complexity is $O(n^2 \log n)$. Take care when processing different connected components.

I. Irreducible Polynomials

Count (modulo m) the number of monic irreducible polynomials of degree n over $\mathbb{F}_p$.

I. Irreducible Polynomials

For a fixed p, we have $x^{p^n} - x = \prod Q(x)$, where $Q(x)$ are all irreducible monic polynomials in $\mathbb{F}_p$ with degrees dividing n.

I. Irreducible Polynomials

For a fixed p, we have $x^{p^n} - x = \prod Q(x)$, where $Q(x)$ are all irreducible monic polynomials in $\mathbb{F}_p$ with degrees dividing n.

Sketch of a proof: consider any such $Q(x)$ with degree $d \mid n$. Due to Lagrange theorem, the multiplicative group of polynomials modulo $Q(x)$ has size $p^d - 1$, hence $x^{p^d} \equiv x \pmod{Q(x)}$, thus $Q(x)$ divides $x^{p^n} - x$ (it's divisible by $x^{p^d} - x$).

I. Irreducible Polynomials

For a fixed p, we have $x^{p^n} - x = \prod Q(x)$, where $Q(x)$ are all irreducible monic polynomials in $\mathbb{F}_p$ with degrees dividing n.

Sketch of a proof: consider any such $Q(x)$ with degree $d \mid n$. Due to Lagrange theorem, the multiplicative group of polynomials modulo $Q(x)$ has size $p^d - 1$, hence $x^{p^d} \equiv x \pmod{Q(x)}$, thus $Q(x)$ divides $x^{p^n} - x$ (it's divisible by $x^{p^d} - x$).

Further, no divisor of $x^{p^n} - x$ can have degree not dividing n, and $x^{p^n} - x$ has no multiple divisors since it is coprime with its derivative.

I. Irreducible Polynomials

Let $N(n)$ with the number of monic irreducible polynomials of degree n, then (by counting degrees) we have

$$p^n = \sum_{d|n} dN(d),$$

I. Irreducible Polynomials

Let $N(n)$ with the number of monic irreducible polynomials of degree n, then (by counting degrees) we have

$$p^n = \sum_{d|n} dN(d),$$

thus by Moebius inversion

$$N(n) = \frac{1}{n} \sum_{d|n} \mu(n/d) p^d.$$

I. Irreducible Polynomials

Let $N(n)$ with the number of monic irreducible polynomials of degree n, then (by counting degrees) we have

$$p^n = \sum_{d|n} dN(d),$$

thus by Moebius inversion

$$N(n) = \frac{1}{n} \sum_{d|n} \mu(n/d) p^d.$$

Finally, this can be computed straightforwardly by factorizing n.

J. John and David Blaine

Construct two multisets of integers A and B such that $|B| < |A| = n$ and $\prod_{x \in A} +1 = \prod_{x \in B}$, and we can obtain multisets A' and B' by removing some duplicates from A and B so that $\prod_{x \in A'} +1 = \prod_{x \in B'}$. All integers should be larger than 1 and less than 2^{63}.

J. John and David Blaine

A solution for $n = 3$ is given in the sample (in fact, it's unique). There is no solution for $n = 2$ (obvious) and $n = 4$ (the proof is long, so we skip it).

J. John and David Blaine

A solution for $n = 3$ is given in the sample (in fact, it's unique). There is no solution for $n = 2$ (obvious) and $n = 4$ (the proof is long, so we skip it).

For $n \geqslant 5$ we can take:
$A = \{2 \times (n-1), 2^{n-3} - 1\}$,
$B = \{(2^{n-2} - 1) \times 2\}$,
$A' = \{2, 2^{n-3} - 1\}$,
$B' = \{2^{n-2} - 1\}$.

This works when $n \leqslant 65$ (otherwise the numbers get too large). When $n > 65$, we will try to take $n' < n$ and split $2^{n'-3} - 1$ into $n - n' + 1$ small enough factors (we also need to split $2^{n-2} - 1$).

J. John and David Blaine

Note that $2^{2k} - 1 = (2^k - 1)(2^k + 1)$ and $2^{3k} - 1 = (2^k - 1)(2^{2k} + 2^k + 1)$. Applying one of these relations for different values n' (and factorizing the resulting factors explicitly), we eventually can cover all n between 66 and 100.

J. John and David Blaine

Note that $2^{2k} - 1 = (2^k - 1)(2^k + 1)$ and $2^{3k} - 1 = (2^k - 1)(2^{2k} + 2^k + 1)$. Applying one of these relations for different values n' (and factorizing the resulting factors explicitly), we eventually can cover all n between 66 and 100.

Complexity: may precompute everything, so not important.