Problem A. New Year and Naming

For convenience, let's zero-index the sequences: $s_0, s_1, \ldots, s_{n-1}, t_0, t_1, \ldots, t_{m-1}$.

By the rule in the statement, the first part of the name cycles every n years and the second part of the name cycles every m years. Thus, the answer is a concatenation of $s_{(y-1) \mod n}, t_{(y-1) \mod m}$.

Problem B. New Year and Ascent Sequence

Define a non-increasing sequence to be a sequence where $a_i \geq a_{i+1}$ for all i. A sequence does not contain an ascent if and only if it is a non-increasing sequence. Thus, we want to count the number of pairs of sequence whose concatenation is not a non-increasing sequence.

If c is the number of pairs whose concatenation is a non-increasing sequence, the answer is therefore $n^2 - c$. We will focus on computing c.

If a given sequence is not a non-increasing sequence, concatenating it with another sequence does not change this property. Thus, we will only consider merging two non-increasing sequences.

For non-increasing sequences s, t to be mergeable as a non-increasing sequence, the last element of s must not be smaller the first element of t. For all given non-increasing sequence, let's denote f_i, l_j to be the first/last element of the respective sequence, we want to count the number of pairs $1 \leq i, j \leq n$ such that $l_i \geq f_j$.

Counting this can be done in numerous ways. One way is to sort the sequence f, and do a binary search for each l_i to find how many elements are at most l_i. The time complexity of this is $O(n \log n)$.

Problem C. New Year and Permutation

By the definition of happiness, we are counting the number of tuples (p, l, r) such that:

- p is a permutation of length n
- l, r is an integer with $1 \leq l \leq r \leq n$
- $[l, r]$ is a framed subsegment in p

Instead of fixing the permutation p as written in the statement, let's fix the integer l, r and find the number of permutations of length n with $[l, r]$ as a framed segment.

If $[l, r]$ is a framed subsegment, all numbers in the range $[min_{i=l}^{r} p_i, max_{i=l}^{r} p_i]$ belong exactly to the framed segment $[l, r]$. Thus, if we fix the order of numbers inside the segment, we can consider it as a single point, instead of a whole subsegment.

Let $len = r - l + 1$. We have $len!$ possible orders inside the segment. If we fix this order, the subsegment shrinks to a single number, and then we have to permute the $(n - len + 1)$ remaining numbers.

This gives the formula $(n - len + 1)! \times len!$ for subsegment $[l, r]$. If we precompute the factorial and add this value for all $1 \leq l \leq r \leq n$, this can be calculated in $O(n^2)$ time. By observing that there exist exactly $n - len + 1$ pairs with $1 \leq l \leq r \leq n, r - l + 1 = len$, we can simply multiply this number for each length, giving a $O(n)$ time solution.

Problem D. New Year and Conference

We only have to check if there exists a pair (a subset of size 2) of lectures that is venue-sensitive. If such a pair exists, then obviously Hyunuk is not happy. If there is no such pair, no subset can be venue-sensitive, as that means there is an intersecting interval pair in some venue but not others.

This gives a simple $O(n^2)$ algorithm: Enumerate all pairs of lectures, and check if both lectures intersect in venue A and B, or don't intersect in venue A and B. This is too slow, so we must do better.

We manage an interval set S for all time points t. For all lectures x that contain time point t at venue A, interval set S includes lecture x's time interval in venue B.

If we are to maintain those time points in sequential order, each lecture can be thought of as two events. For a lecture (sa_i, ea_i, sb_i, eb_i), we have two corresponding events.

- Event adding interval [sb_i, eb_i] to S at time sa_i
- Event removing interval [sb_i, eb_i] from S at time $ea_i + 1$.

All lectures whose time intervals intersect in venue A must also intersect in B. Therefore, after an adding event occurs, added time interval [sb_i, eb_i] must intersect all time intervals already in set S.

An interval $[c, d]$ does not intersect an interval $[a, b]$ if and only if $d < a$ or $b < c$. Therefore, we can only care about intervals with the largest starting point (maximizing c) and the smallest ending point (minimizing d). Let s_{max} be the largest value among the starting point of interval in the set S, and e_{min} be the smallest value among the ending point of interval in the set S. The newly added interval [sb_i, eb_i] must satisfy the condition $s_{max} \le eb_i$ and $sb_i \le e_{min}$.

This can be computed with $O(\log n)$ time complexity using a data structure like `std::multiset`. Since there are $O(n)$ events, the total time complexity is $O(n \log n)$.

Note that, since this algorithm only checks whether all intervals intersecting in A intersects in B, but not in the other direction, you need to check the same condition swapping both venues. This can be easily done by swapping time intervals in the venue A and B and calling the same function.

Problem E. New Year and Castle Construction

Writers know of at least three different methods to approach this problem. The following approach is not the easiest one, but the most beautiful one in my opinion.

We will take the approach of calculating $\sum_{p \in S} f(p)$ itself, instead of computing each term independently. Consider all 5-point subsets out of n points. As no three points are collinear, there are three cases:

- Convex hull has 3 vertices. We denote the number of such subsets as x_3. In this case, the two vertices strictly inside the hull can be enclosed by other vertices (Draw a straight line connecting those two points, and extend the non-intersected edge toward one of them, to enclose the other). The contribution to the answer is $2 \times x_3$.
- Convex hull has 4 vertices. We denote the number of such subsets as x_4. In this case, the point not in the hull can obviously be enclosed, but the points on the hull cannot be. The contribution to the answer is x_4.
- Convex hull has 5 vertices. We denote the number of such subsets as x_5. In this case, no point can be enclosed. The contribution to the answer is 0.

We can see that the answer we want is $2x_3 + x_4$. On the other hand, $x_3 + x_4 + x_5 = \binom{n}{5}$, by definition.

Now here comes the magic. If we can compute the value $3x_3 + 4x_4 + 5x_5$ somehow, then the answer can be easily derived. It turns out this can be done in $O(n^2 \log n)$ time, by exploiting the form $\sum_{i=3}^{5} i \times x_i$. This quantity corresponds to the sum of edges of the convex hull for all 5-point subsets. Then, we can instead try to enumerate all $O(n^2)$ possible edges, and count the number of 5-point subsets that contain each edge.

Consider each edge as being directed. In other words, we'll only consider edges where (i, j) are adjacent in the counterclockwise direction. Then the number of possible 5-point subsets is simply $\binom{l_{i,j}}{3}$, where $l_{i,j}$ is the number of vertices that lies left to the directed vector (i, j). $l_{i,j}$ can be obviously counted in $O(n^3)$ time. If we perform an angular sort for fixed i, we can optimize this to $O(n^2 \log n)$ time by using two pointers.

Challenge: Find x_5 in $O(n^2 \log n)$.

Problem F. New Year and Social Network

This problem can be cracked in polynomial time with various approaches. I'll introduce a few of them.

$O(n^3)$

As stated in the problem, this problem is a straightforward bipartite matching problem. With a simple DFS or union-find, you can construct a bipartite graph in the statement, and use well-known maximum matching algorithm.

$O(n^{1.5} \log n)$

By using a similar solution to *Codeforces 786E: ALT*, you can reduce the time complexity to $O(n^{1.5} \log n)$. The author didn't manage to squeeze this approach in time.

$O(n^2)$

By looking at the sample, it seems that $m = n - 1$ always holds for some reason. In other words, the given graph always has a perfect matching. It turns out that this guess is true. Let's try to prove it using induction.

Inductive Proof. Attempt 1

We induct on the quantity $|E(T_1) - E(T_2)|$. If the quantity is zero, then we can find an obvious bijection. Otherwise, we can find some edge $e \in E(T_1) - E(T_2)$. Since our goal is to reduce the difference, it would be reasonable to replace this edge to some other edge $f \in E(T_2) - E(T_1)$. Hopefully, we can show the following:

Lemma 1. For any $e \in E(T_1) - E(T_2)$, there exists some edge $f \in E(T_2) - E(T_1)$ such that $T_1 - \{e\} + \{f\}$ is a tree.

Proof. $T_1 - \{e\}$ have two components. If there exists an edge f that connects these two parts, we are done. Otherwise, T_2 has no edges connecting these parts, finding a cut.

Then, we can simply add the bijection $\{e \to f\}$, replace the edge e with f in T_1, and continue. Right? Sadly, this is wrong as it is changing the tree T_1. Although we found the pair of tree $(T_1 - \{e\} + \{f\}, T_2)$ with less value $|E(T_1) - E(T_2)|$, finding a bijection over $T_1 - \{e\} + \{f\}$ to T_2 is a different story, thus the proof fails.

Inductive Proof. Attempt 2

It would be good to make T_1 static, since otherwise our induction will fail. Then, what about repeating induction over $(T_1, T_2 + \{e\} - \{f\})$? Everything is good, but then we should also guarantee $T_2 + \{e\} - \{f\}$ is a tree. Is it possible?

Lemma 2. For any $e \in E(T_1) - E(T_2)$, there exists some edge $f \in E(T_2) - E(T_1)$ such that $T_1 - \{e\} + \{f\}$ and $T_2 + \{e\} - \{f\}$ is a tree.

Proof. $T_2 + \{e\}$ has exactly one cycle C. Since $T_2 + \{e\} - \{f\}$ should remain acyclic, $f \in C$ should hold. The cycle consists of edge e and path from T_2, we should find such f from the path in T_2, such that its endpoint connects two different component of $T_1 - \{e\}$. Label the vertices in the path by its component membership of $T_1 - \{e\}$. Since e itself connects the different component, there should exist a edge in the path, which connects different component. Take it.

Now, as $T_1 - \{e\} + \{f\}, T_2 - \{f\} + \{e\}$ is both acyclic, we can create the bijection $\{e \to f\}$, and also take $(T_1, T_2 - \{f\} + \{e\})$ for the induction stage. This proof also gives a straightforward $O(n^2)$ algorithm for finding the answer.

$O(n \log n)$

When we fix $e \in E(T_1) - E(T_2)$, finding the component where each vertices lie is easy: This can be done by preprocessing T_1 with a single depth-first search. Using this to find the f is a different story: We have dynamically changing tree of T_2, which is hard to maintain.

On the other hand, since Lemma 2 is symmetric, we can rather fix $f \in E(T_2) - E(T_1)$ and find e instead. In this case, if we can know which subtree that a vertex belongs for $T_2 - \{f\}$:

- If $f = \{u, v\}$, find the path between $u \to v$ in T_1
- If $LCA(u, v)$ belongs to same component in u, then find f in path $v \to LCA(u, v)$. Otherwise, find f in path $u \to LCA(u, v)$.
- Now, you can find e using binary search on paths, which can be implemented with binary lifting

Since T_1 is static, we can afford all these operation in $O(n \log n)$ time preprocessing. Now the point is to maintain T_2 to support subtree membership query. This can be done straightforwardly using Link-Cut tree or Euler Tour tree using $O(\log n)$ query time, which results in $O(n \log^2 n)$ time, but this is too slow (at least for us) to pass.

Now from this point, we will assume $E(T_1) \cap E(T_2) = \emptyset$. We have a freedom over choosing f, so let's choose the f as the edge connecting the leaf. The good property of leaf is that the subtree membership query is very easy: You can simply compare the vertex number. So, if we choose such f, we can find e easily.

What about changing the tree T_2 to $T_2 - \{f\} + \{e\}$? Note that, After T_2 becomes $T_2 - \{f\} + \{e\}$, we don't have to care about e because it's in the intersection of T_1 and T_2. We can think about **contracting** the edge e: After adding an edge e we can not delete, so in the component point of view, they are always in the same component: So we can remove the leaf edge f, and contract the ends of e. For the data structure to maintain these components, we can use Union-Find.

Challenge: Lemma 2 works for the general matroid base set. So for any pair of matroid base set from same ground set, perfect matching exists in the exchange graph. Prove it.

Problem G. Seollal

The problem asks us to find a spanning tree on a grid graph, where some vertices are not allowed to be leaves. Let's denote such set of vertices as S. Note that, per the problem statement, this set is independent. Thus, there exists no edge connecting two vertices in S. Now, we present the following theorem.

Theorem. We can find a desired spanning tree if and only if there exists an acyclic edge subset where every vertex in S has a degree exactly 2.

Proof.

$\leftarrow$: Since it is an acyclic edge subset, we can obviously expand it to a spanning tree by arbitrarily adding edges to connect different connected components. Vertices with degree at least 2 are not leaves.

$\rightarrow$: As noted above, S is independent, and it has degree at least 2 in the desired spanning tree. Thus, we can pick any two incident edges for each vertex in S.

Now, all that is left is to find such an acyclic edge subset. In general, as S is independent, we can pick two arbitrary incident edges, but this does not guarantee that the subset is acyclic. One can try to use a non-polynomial solution strategy, such as storing connection profiles in a DP, but the input is a bit too large for that.

Our solution is polynomial, and we will use a stronger tool that is known as *matroid intersection*. From now on, we will assume that the reader has familiarity with matroids as well as how to compute the intersection of two matroids.

Let's relax the degree condition to *at most 2* instead of exactly 2. Now, the acyclic condition corresponds to a graphic matroid, and the degree condition corresponds to a choice matroid. By using Edmond's algorithm, we can find an independent set of maximum size. The exact degree condition is satisfied if and only if the maximum independent set has size $2|S|$, so we can know whether such an edge set exists, and if it exists we can also get a certificate which is then used to print the answer.

In our matroid, we can implement Edmond's matroid intersection algorithm in $O(E^3)$ time where E is the size of the ground set. As E is about $2nm$, the final time complexity is $O(n^3m^3)$ with low constants, just like any other augmenting-path algorithms.

Trivia 1. This problem's working name was `yet-another-nowruz`. You may want to compare the statement of this problem and IOI 2017 Nowruz ;)

Trivia 2. Huge applause to `kriii` who created amazing tests to break random solutions of G. We were very afraid of random solutions because of the huge difficulty gap between ABCDE/FG. Hopefully, the first 80 tests were good enough for the contest. But that's not the end of the story: we have 140 tests ;)