Problem D. Double Move

There are two parts of this problem:

- For the given set of complete choices (i.e., for $k = n$), find the number of assignments in which Alice wins.
- Run a brute force search on all decisions made by both players to determine which move is optimal, and prune this search to speed it up.

Let's tackle the first part. Consider an initially empty graph on n vertices, numbered 1 through n. Denote this graph by G_0. Recall the i-th declaration (numbered from 1) is a pair a_i, b_i. To build graph G_i, we add an undirected edge $\{a_i, b_i\}$ to G_{i-1}. Note that these graphs may contain multiple edges or loops.

We model the choice made by Claire by orienting each edge towards the vertex corresponding to Claire's choice. That is, an edge $\{a_i, b_i\}$ is substituted by an arc (a_i, b_i) if b_i is selected by Claire and (b_i, a_i) otherwise. Again, this graph may contain multiple edges, loops or edges in opposite directions. In order for the game to continue, the in-degree of each vertex must be at most 1.

Let G be an undirected graph. We define $f(G)$ to be the number of such orientations of graph G. Clearly, $f(G_0) = 1$ and $f(G_1) = 2$. The following values, however, depend on the structure of G_i. Let's learn how to evaluate f.

Consider connected components of f. Clearly, the decisions made in one component don't influence the other. Hence, we can split G into connected components, evaluate f in each of them and then multiply the results.

A connected graph G on m vertices contains at least $m - 1$ edges. Let's break this down based on the actual number of edges.

- If G contains exactly $m - 1$ edges (i.e, it is a tree), there are exactly m valid orientations. To see why this is true, observe that we can root the tree in the only vertex that has in-degree 0, and each edge is oriented away from the root.
- If G contains exactly m edges, then it is a tree plus a single edge. We can orient this additional edge in an arbitrary way. Note that this uniquely determines all remaining edges (as we can reduce this to the previous case where one vertex must have in-degree 0). This means there are exactly 2 ways of orienting the edges in such a graph.
- If G contains at least $m + 1$ edges, there are more edges and vertices. By pigeonhole principle, it is impossible to orient the edges such that the indegree of each vertex is at most 1.

Let $g(i)$ be the number of Claire's choices that end the game after exactly i turns. Clearly, the number of games won by Alice is $g(2) + g(4) + g(6) + \ldots$.

We claim that for i from 1 to $n + 1$, we have $g(i) = f(G_{i-1}) \times 2^{n+2-i} - f(G_i) \times 2^{n+1-i}$. Let's prove this.

Consider G_i. The first i of Claire's decisions are modelled by orienting the edges of G_i, while the remaining $n + 1 - i$ decisions are not. This means that for every fixed orientation of G_i, there are exactly 2^{n+1-i} possible Claire's choices that lead to this directed graph. A similar argument can be made for G_{i-1}. Now, note that the assignments that end the game after exactly i moves are those that have a corresponding valid orientation in G_{i-1}, but not in G_i. This concludes the proof.

Now that we know how to compute the answer for a given set of choices of Alice and Bob, let's concentrate how to find the answer when $k < n$. We will do this by employing the so called minimax algorithm. This algorithm works as follows.

Every list of between 0 and $n+1$ player declarations is called a **game state** s. The value $h(s)$ is called the **evaluation** of the game state, and it represents the number of Claire's choices that result in Alice's win, given that both sides play optimaly **from this state onwards**. Clearly, if s contains exactly $n+1$ player

declarations, then the game is finished and $h(s) = g(2) + g(4) + g(6) + \dots$. To compute the evaluation for a state s that isn't final, consider the set $S(s)$ of all states reachable in one move, i.e., states that append exactly one declaration to s. Then, for Alice's turn

$$h(s) = \max_{s' \in S(s)} h(s')$$

and for Bob's turn

$$h(s) = \min_{s' \in S} h(s')$$

Put together the answer for a starting state s (if Alice starts) is

$$h(s) = \max_{s^{(1)} \in S(s)} \min_{s^{(2)} \in S(s^{(1)})} \max_{s^{(3)} \in S(s^{(2)})} \dots h(s^{(n-k)})$$

The name minimax is coined by this alternation of minimum and maximum.

With all these bits, we can finally discuss the solution.

Subtask 1

In the first subtask, the limits are very small, and we can simply try all possible move sequences. Since each of them can be evaluated in $\mathcal{O}(n)$, and there are $\binom{n}{2}^{n+1}$ of them, the solution runs in $\mathcal{O}(n^{2n+3})$ time. Note that this solution doesn't need the observation on how to compute $f(G)$ by inspecting the connected components of G, and we can use brute force for that step, yielding a solution with running time $\mathcal{O}(n^{2n+2} \cdot 2^n)$.

In the following subtasks, our goal is to reduce the number of states considered. Note that we do not expect the contestants to be able to compute how exactly does the number of states change with each optimisation, but rather to intuitively understand why such optimisations reduce the number of states. Nevertheless, the concepts used in the analysis contain useful topics for contestants' further study.

Subtask 2

Note that $f(G)$ doesn't depend on the actual graph structure, but only on its connected components and whether it contains $m-1$, m or more edges. That is, we only need to store the connected components and their type. Additionaly, any component of the third type (more than m edges) makes answer equal to 0, regardless of further declarations, and we can break immediately.

As there are only two component types, there are

$$\sum_{i=0}^{n} 2^i \left\{ {n \atop i} \right\}$$

states, where $\left\{ {n \atop i} \right\}$ is the Stirling number of the second kind, or the number of ways to partition n elements into exactly i non-empty sets. For $n = 10$, this number is 4 412 798, which yields a fast enough algorithm.

Subtask 3

For subtask 3, notice that after processing the initial declarations, the vertices within a connected component are interchangeable. In other words, we only need to remember the number of vertices and the number of edges in each component. Furthermore, when enumerating choices for adding an edge, we may simply add an edge within a component, or merge two components together.

How many game states do we have? We no longer partition a set of distinguishable elements, but a set of indistinguishable elements. As such, the number of parititions is dictated by the Partition number $p(n)$ – the number of ways of expressing n as a sum of positive integers, regardless of their order. This quantity still grows exponentially with n, but much slower than the previous one.

To pinpoint the number of states, we split the vertices into those that are in the component of first type and those that are in the component of second type, and then partition each separately. Thus, the number of states is

$$\sum_{i=0}^{n} p(i) \cdot p(n-i)$$

where $p(i)$ is the Partition number. For $n = 25$, this quantity is 127 554.

Subtask 4

For subtask 4, notice that there aren't that many possible test inputs as $k = 0$. We can thus run the solution for all cases locally, and submit just the hardcoded answers. The reference solution for the subtask 3 finishes in about two minutes for $n = 35$ and $k = 0$ on an average machine.

Subtask 5

The final observation needed is that we don't need to know the sizes of the components of the second type. This is because the answer doesn't depend on their size, and when we add an edge to them in any way (either by adding an edge within the component, or by connecting it to another component), no component of the first type can be created. Thus, it is sufficient to remember the sizes of components of the first type, and the number of the components of the second type. The total number of states is now

$$\sum_{i=0}^{n} p(i) \cdot (n-i+1)$$

For $n = 35$, this quantity is 403 779.