LOPTA

Let's first consider the trivial solution that actually rotates the table in memory each time it is rotated and then simulating the ball's fall, seeing where it will end up. This approach requires $O(K \cdot N^2)$ time, what is too much.

However, by observing that there are only four possible rotations of the table, we can precompute the layouts of the table instead of actually rotating it each time. This will cut our runtime down to $O(N^2+N \cdot K)$, which is better, but still suboptimal.

Additionally, we can also preprocess where the ball will fall if it ends up at some position in one of the four tables, for each of the N^2 positions and four layouts. This preprocessing takes $O(N^2)$ time if done using dynamic programming so the overall time complexity is $O(N^2+K)$.

NOGOMET

We can define a recursive function which sorts one sub-league according to the rules in the problem description (see the reference C99/C++ solution for a commented version of one such function). We obtain the solution by using this sort function to sort the given league.

RIBARI

The key observation here is that we can use a greedy algorithm to check if the given cities can feed X children each. The algorithm considers cities in left to right order. If a city has more than X units of food, it can send all of its excess food to the city directly to its right. If it has less than X units of food, it needs to receive food from the city to its right (note that, unless it is the leftmost city, it may have already implicitly received or sent out food to the cities to its left). If the rightmost city has enough food then X is a candidate solution.

Now note that if X is a candidate solution, X-1 is most certainly also a candidate solution. Similarly, if X is not a candidate solution, X+1 also cannot be a solution. With that in mind, we can use binary search to find the largest X that is a candidate solution. The time complexity of this algorithm is $O(\log_2 MAX \cdot N)$, where MAX is any obvious upper bound on the solution.