PRASE

One way to solve the problem is to, when a child takes a piece of food, examine all previously taken pieces and count how many were taken by that child and how many by the others.

MAGIJA

The cell (i, j) in the given quarter is copied to three other cells. The coordinates of those cells are (depending on the implementation):

- If the upper left cell is (0, 0), the coordinates are (2R-i-1, j), (i, 2C-j-1) and (2R-i-1, 2C-j-1).
- If the upper left cell is (1, 1), the coordinates are (2R-i+1, j), (i, 2C-j+1) and (2R-i+1, 2C-j+1).

After copying the cells we add the error, changing the specified cell.

MARATON

Instead of copy-pasting the checking code 8 times for each of the directions, we can store in an array the row and column components of moving in the 8 directions. We can then use the same block of code to process all 8 directions. This is a common technique in problems on grids where we are required to do something in more than one direction.

KAMEN

Before Domeniko throws in the first rock, let's calculate and remember, for each column, the path a rock would take if Domeniko threw it in that column.

Now, when Domeniko says which column he wants to throw a rock into, we know exactly where the rock will settle – the last cell on the path we previously calculated for that column.

In order to be able to do the same for each subsequent rock, we must refresh each path on which the last cell is the one the thrown rock ended at. We can do that by simulating the effect of gravity from the second last cell in the path (as the path before that didn't change).

The overall time complexity is O(C(R+N)).

V

If X is larger than say 10^5, then B/X is at most 10^6, meaning that there is at most one million multiples of X to check. A "naive" solution that checks each and verifies that it is composed of the given digits is efficient enough.

If X is at most 10^5, then integers divided by X give remainders no larger than 10^5-1.

We will construct the function f(n, prefix), the number of multiples of X between A and B, if we still have to place *n* more digits (of 11) and *prefix* represents the digits already placed.

To calculate the values of this function, first observe that, for any n and prefix, we can only form numbers between prefix$\cdot 10^n$ and prefix$\cdot 10^{n+1}$-1, inclusive. Based on this we recognise three cases:

- If all numbers are between A and B, then the actual prefix doesn't matter when forming the rest of the number, just its remainder when divided by X. The limited range of the remainders reduces the state space and we can memoize the values of the function based on the remainder.
- If all numbers are less than A or greater than B, then the value is zero.
- If some (but not all) numbers are between A and B, then the value of the function doesn't depend only on the remainder of dividing the prefix by X, so it isn't possible to memoize the value of the function on the remainder. Luckily, there are few such cases (proportional to the length of the number).

You can find a concise implementation in the provided source code, contributed by Luka Kalinovčić.

PROSTOR

We'll say a rectangle with coordinates (x_1, y_1, z_1, x_2, y_2, z_2) is of type X if it is "thin" in the x dimension, i.e. if x1 equals x2. Rectangles of types Y and Z are similarly defined.

Define a function count(X, Y) that does the following:

Select all rectangles of types X and Y and count how many pairs of rectangles there are such that at least one of them is of type X and they share a point.

The overall solution is count(X, Y) + count(Y, Z) + count(Z, X).

Algorithm for count(X, Y):

Imagine that the y coordinate represents time. Let time flow and observe the (x, z) plane representing space at some moment in time. Rectangles of type X become vertical line segments while rectangles of type Y remain rectangles.

Rectangles of type X have some positive lifetime (because $y_1 < y_2$), while rectangles of type Y exist at only one instant (because $y_1 = y_2$).

As time flows, three types of events can occur:

1. A type X rectangle starts
2. A type Y rectangle starts and ends immediately
3. A type X rectangle ends

We sort the events by time and process them one by one.

To count all pairs of rectangles sharing a point, such that one is of type X and another is of type Y, we need to, whenever a rectangle (x_1, y, z_1, x_2, y, z_2) of type Y comes to life, answer the question "how many vertical line segments in a set intersect the rectangle (x_1, z_1, x_2, z_2)?".

To count all pairs of rectangles sharing a point, such that one is of type X and another is of type X, we need to, whenever a rectangle (x_1, y, z_1, x_2, y, z_2) of type X comes to life, answer the question "how many vertical line segments in a set intersect the (degenerate) rectangle (x, z_1, x, z_2)?".

Both questions can be quickly answered if we store the starting and ending points of all vertical segments in a 2-dimensional Fenwick tree (a la IOI 2001 mobiles).

The overall time complexity of the algorithm is $O(N \log N + N \log^2 M)$ where M is the largest allowed coordinate (999 in this problem).

For further details see the provided source code.