MAGNETI

For each sequence of L consecutive starting (unit-length) magnets we check if we can make a magnet of length L from them, and calculate how many flips we need.

Let a[i] be the orientation of magnet i.

If any of these is satisfied, then the sequence of L magnets starting with magnet i **cannot** form a magnet of length L:

- a[i] equals a[i-1]
- a[i+L] equals a[i+L-1]
- a[i] does not equal a[i+L-1]

The above does not hold for boundary cases, we check these separately.

If a sequence of L consecutive starting magnets forms K magnets after the initial joining, and if it is possible to make a magnet of length L from them, then it is easy to show that it can be done with K/2 flips.

KARTA

The first step is to isolate the names and positions of all times. Then find all pairs of adjacent names and positions.

We get a graph in which names and positions are vertices, and there is an edge between them if they are adjacent. Such a graph is called bipartite, because the vertices are split into two groups and there are no edges between vertices in the same group.

The task asks us to match each name to an adjacent position. The problem of maximum bipartite matching is common in graph theory and there are efficient algorithms for solving it.

However, the task guarantees that the matching is unique. It can be proven from this fact that there is always a name or position that provides only a single option for pairing (this does not hold for the general bipartite matching problem). The claim can be proven by contradiction, by trying (and failing) to construct a graph with a unique solution in which all vertices are of degree two or greater.

SLIKA

Let Z be the colour for which we are trying to determine the interval I from the problem statement, and let

$$I = [minZ, maxZ].$$

Observe that it is simple to find minZ, the dimension of the smallest possible square of colour Z. If the digit Z does not appear at all in Mirko's image, then it is possible that square Z had a side only 1 pixel long. If not, let minR, maxR, minC, maxC be the indices of the first and last rows and columns in which Z can be found. Then it is clear that

$$minZ = \max \{ maxR - minR + 1, maxC - minC + 1 \}.$$

The complexity of calculating minZ for one digit is $O(N^2)$.

Calculating maxZ is harder, requiring the use of dynamic programming.

We say that a square is **good** if all its pixels are coloured Z or greater. Consider pixel (R, C) as the lower-right pixel of some good square and let M(R, C, Z) be the length of the largest such square. Then:

1. (R, C) must be coloured Z or larger
1. M(R, C, Z) <= M(R–1, C, Z) + 1;
2. M(R, C, Z) <= M(R, C–1, Z) + 1;
3. M(R, C, Z) <= M(R–1, C–1, Z) + 1.

The opposite also holds. That is,

M(R, C, Z) = 0, if pixel (R, C) is not coloured Z or greater;

M(R, C, Z) = 1 + min { M(R–1, C, Z), M(R, C–1, Z), M(R–1, C–1, Z) }, otherwise.

How does M(R, C, Z) give us maxZ? The square we are looking for in the task is the largest good square such that all occurrences of the digit Z are inside it. In other words, its first row is at most minR, last row at least maxR, and similarly for columns. To calculate all M(R, C, Z) we need $O(K \cdot N^2)$ operations. Because the recursive formula only uses the previous row, we can discard the calculated values of M for earlier rows. The memory complexity is only $O(K \cdot N)$ with this.