CETVRTA

We start by observing four points (x1, y1), (x2, y2), (x3, y3), (x4, y4) which form a rectangle. Numbers from the set {x1, x2, x3, x4} take two distinct values (each value two times). If we remove any point, it's x coordinate can be determined by observing which value appears only once. The same holds for the y coordinate.

PEG

For each piece on the board and each direction we check if it is possible to move the selected piece in the selected direction. Caution should be taken not to move the piece off the board.

PRINOVA

Solution that iterates over all odd numbers from [A, B] and calculates the distance to the nearest name is too slow for the given constraints.

Let us solve a modified version of the problem where the solution can be odd or even. Than the solution is one of the following:

- A
- B
- $(P_i+P_j)/2$, for every pair i, j

For each solution candidate we first check if it falls inside [A, B] and than find the distance to the nearest name. The solution is thus the number that maximizes that distance.

In the original problem only odd solutions are allowed. To account for this, we simply substitute each even solution candidate X with numbers X - 1 and X + 1.

ZAPIS

We start by observing the first character in the sequence. That character must be an opening bracket. It's closing bracket can only be located on even position in the sequence (because a regular bracket-sequence has even length)

For each even position x, and for each type of brackets we now check if it is possible to place an opening bracket as the first character and a corresponding closing bracket as x-th carachter. If it is possible then we add to the total number of solutions the number of regular bracket-sequences from second character to (x-1)-st character multiplied by number of regular bracket-sequences from (x+1)-st character to the last character.

By doing this we have reduced the large problem to two smaller subproblems of the same type, so it can be solved using dynamic programming or memoization.

SREDNJI

One obvious observation is that if the median of subsequence is B than the subsequence contains B.

For a subsequence P we define a function delta(P) as difference of elements greater than B and elements smaller than B. Thus B is the median of P if and only if P contains B and delta(P) = 0.

Furthermore, if we split P into subsequences P_L – part that is left of B and P_R – part that is right of B, than P has median B if delta(P_L) + delta(P_R) = 0.

This leads to the following algorithm:

For each subsequence ending with B we calculate the delta function and store the number of subsequences having delta value of d in L(d).

For each subsequence starting with B we calculate the delta function and store the number of subsequences having delta value of d in R(d).

To obtain the solution for each value d from [-N, N] we sum L(d) * R(-d).

STAZA

First we observe that any path ending in city 1 has a corresponding path starting in city 1. It is sufficient to reverse the sequence of roads forming the path. To simplify things, we will be trying to find the longest path starting in city 1.

If we observe the given network we can see that, speaking in graph teory terms, each road is either a part of a single ring, or a bridge.

In order to solve the problem, we must first identify rings and bridges. One of the ways of doing this is by constructing a DFS tree from node 1 and calculating all standard values(discovery time, finish time, lowlink value). Details can be found in the source code.

For a given ring we say that it "hangs" from node X if node X is the highest node in the DFS tree in that ring. For a given bridge we say that it "hangs" from node X if node X is the higher node in the DFS tree.

For each node X we define a subgraph of node X as the union of node X and all subgraphs of all nodes lying in rings "hanging" from X and all subgraphs of all nodes on other side of bridges "hanging" from X.

For each node X we need to find two numbers, circle(X) - path inside subgraph of node X starting and ending in X, and path(x) - path inside subgraph of X starting in X and ending in any node.

circle(X) can be calculated by simple recursion as sum of lengths of all rings "hanging" from X and sum of all circle(Y) for each node Y lying on those rings (because we can take the circle on any node Y and end up back at Y).

When calculating path(X) we need to take into account the following possible scenarios, selecting the one that yields the longest path::

- The path from X ends in X. This leads to path(X) = circle(X).
- We can first make a circle in subgraph of X, and than take one bridge "hanging" from X into a new subgraph giving: path(X) = circle(X) + path(Y) where Y is the node on the other side of the bridge.
- We can make a circle in all rings "hanging" from X except one and than select one city in that ring as the ending. In that case there are two possible ways to arrive to the selected city so we need to find the longer.

The solution is then path(1). Details on how to implement this can be found in the source code.