The Queen Problem

Paliukhovich Anton

1 Statement

Given a chessboard of size N and K queens, which are already placed on the board so that they do not beat each other. At the same time, the queens in this problem are unusual – in addition to the standard vertical, horizontal and diagonal movements, they can also walk like a chess knight for an arbitrary number of steps. That is, if the queen is at position (x, y), it can go to position (x + n, y + 2n), (x + 2n, y + n), (x - n, y + 2n), (x - 2n, y + n) for any integer (not necessarily positive) number n (the position must be within the board).

You need to arrange the remaining N-K queens on the board so that no two queens beat each other as a result.

2 Input

The first line of input data contains two integers N and K — the size of the board and the number of queens already placed ($1 \leqslant N \leqslant 10000, 0 \leqslant K \leqslant N/2$). The next K lines contain pairs of integers (xi, yi) describing the current arrangement of queens on the board ($1 \leqslant x_i, y_i \leqslant N$).

3 Output

The first line of the output should contain a single word YES or NO, meaning whether the queens can be placed on the board so that they do not beat each other. If the queens can be placed, then there should be N numbers $y1, y2, ..., yn$, each on its own line. The number yi must describe the column in which the queen stands on row i. The resulting solution must be consistent with the original arrangement of K queens.

4 Solve

In my solution, a vertex is a permutation p of size n, where (i, p[i]) are the coordinates of the queens. This ensures that the queens will stand in different columns and different rows. It remains to find such a permutation of queens that they stand in different diagonals. Neighbors are permutations that differ in exactly two positions.

As a functional for minimization, I considered two functions:

1. The number of pairs of queens standing on the same diagonal.
2. The sum of all queens, in how many occupied diagonals is the queen.

To solve the problem in both cases, you need to zero the functionality. These functions are convenient,since they can be recalculated as a constant. In practice, the 2nd functional proved to be better. Apparently, this motivated the queen more to leave the occupied diagonals.

At first, to minimize the functional, I used annealing, with a geometric decrease in temperature, the coefficient was selected depending on the number of iterations. As a transition, I selected two random indexes from those that were not in the occupied set from the input, and changed

their values in the permutation. Then, I decided to take one vertex not randomly, but in a circle. The quality has increased, this is due to the fact that with a random selection, many indexes will rarely be selected, and so we guarantee that we will consider all the indexes in one round.

But, still, in situations where the size of n is less than 150, it worked poorly, since the solution it was unstable, due to the fact that few permutations satisfied the condition of the problem. Then, I decided to call the annealing procedure 5 times if the size is small. It didn't help much. After that, I made the temperature a constant parameter, depending on the size, and selected different temperatures for different sizes. This ensures that if we get to the local optimum, then we can get out of there.

For large n sizes, annealing is not suitable, as it does a lot of unnecessary and suboptimal actions. On the other hand, it is easier to find a solution, since there are enough suitable permutations. To solve this problem, I used local search, I started a priority queue on the indexes, as a comparison I took the number of occupied diagonals on which the queen stands, and as the second parameter for comparison I took random so that there were no loops. During the iteration, I took two elements from the top of the queue and changed them, if it is profitable. It didn't work. Then I took the first item in the queue and with probability 0.7 is the second element, with probability 0.2 is the third, and with probability 0.1 is the fourth. It didn't work either. Then I decided to take the first element in the queue, and as the second in a circle all the indexes that are not occupied by the condition. And this solution has already worked.

For a complete solution, I combined all three algorithms.