Bishop solution

Proposed by Pătcaş Csaba

1 Solving the problem in exponential time

The first idea to solve the problem would be to generate all the possible configurations, by placing each of the k bishops in every possible square on the board in such a way that no two bishops attack each other. This can be done using the backtracking method, as it can be seen in *bishop1.cpp*.

We can observe, that the board has $n+m-1$ diagonals, and of course we can place at most one bishop on a diagonal, because otherwise, the bishops would attack each other. Using this observation we can solve the problem by selecting k diagonals from the $n+m-1$ available in every possible way, then placing the k bishops on the selected diagonals, with one bishop on each diagonal (*bishop2.cpp*).

Using some chess knowledge we can improve the above solutions. The squares of the chessboard are colored black and white, depending on the parity of the sum of their coordinates. A bishop on a white square can never reach a bishop on a black square, and vice-versa. This property leads to the following solution: let us place i bishops on the white squares and $k-i$ bishops on the black squares, such that no two white bishops attack each other and no two black bishops attack each other. By multiplying the number of placements of the white bishops with the number of placements of the black bishops we get the total number of configurations for the actual i. By adding these products for every $i = 0 \dots k$ we get the final solution (*bishop3.cpp*).

2 Performance comparison

We will try to place eight bishops on a chessboard of normal sizes, to compare the performance of the three algorithms. We will use a computer with

AMD Barton 2500 processor, 768 Megabytes memory and Windows XP Professional operation system. We will run the programs under Visual Studio 2005 in Release mode. The running time can be seen in the following table:

Program	Time
bishop1.cpp	16.688s
bishop2.cpp	16.813s
bishop3.cpp	0.015s

Surprisingly, *bishop1.cpp* slightly beat *bishop2.cpp*, but *bishop3.cpp* clearly outclassed both. Nevertheless, the execution of *bishop3.cpp* for the worst-case scenario, $n = 13, m = 13, k = 15$ took 151.063 seconds, which is too slow for the time limit on its own, not considering the multiple input.

3 Possible working solutions

The key observation to solve the problem is, that there are only 8281 possible inputs, so we can precalculate all of them. This took 1 hour and 45 minutes on the specified machine, so it can be safely done in the 5 hour contest time. By using the symmetry $solution_{n,m,k} = solution_{m,n,k}$ the time needed to generate all the solutions can be halved.

Alternative solutions do exist, but we won't go in details, because they are somewhat hard, and not needed to solve the problem for these limits.

For instance we can use the memoization technique to build a matrix with $(n + m - 1)$ rows and 2^{n+m-1} columns. Adding the elements from the last row for which the binary representation of the column number has exactly k bits equal to 1 we get the solution. Because this matrix may need too much memory, we can build two similar smaller matrices for each color.

In fact, the problem can be solved in polynomial time, using a dynamic programming approach. That way, we can avoid the precalculation step, because this solution is fast enough to calculate the number of possibilities in run-time.