Facts about the game

There exist exactly 18368 legal positions, which gives arise to 36736 states of the game (for each position either one the player may be at turn). 240 of these 36736 states are “mate” states; that is, the player to move has no legal move. 16560 states are winning for one of the players. The longest forced win are 5 moves (that is, 9 half-moves, or plys).

Most positions which has a forced win have only one single move out of more than a hundred possible moves which is correct. In other words, there is no way one can win the game against perfect play be just guessing moves.

Some examples:

```
--------------------------------------------
#.xx
#...
##.*
.***

Player * to move - * wins in 5 move(s)

There are 117 available moves:
  1: * wins in 4 move(s)
 80: draw
  4: # wins in 5 move(s)
  1: # wins in 4 move(s)
  2: # wins in 3 move(s)
  1: # wins in 2 move(s)
 28: # wins in 1 move(s)
--------------------------------------------
#.xx
#...
##.*
.***

Player # to move - # wins in 1 move(s)

There are 169 available moves:
  2: # wins in 0 move(s)
136: draw
  1: * wins in 5 move(s)
  1: * wins in 3 move(s)
 29: * wins in 1 move(s)
--------------------------------------------
.###
*#.x
***.
x...

Player * to move - * wins in 3 move(s)

There are 117 available moves:
  1: * wins in 2 move(s)
 79: draw
  5: # wins in 5 move(s)
  3: # wins in 3 move(s)
 29: # wins in 1 move(s)
```

Solution

This problem can be solved by retrograde analyze. The solution can be split up into several steps:

Enumerate the states

Although there are 36736 states, it's unnecessary to actually find a 1-1 mapping between the integers 0..36735 and those states. It's easier to realize that there are 48 positions in the 4x4 grid for each L, and for the neutral pieces there are 8*7/2 remaining combinations. Thus we have 2*48*48*8*7/2 = 129024 "states" (of which most are invalid, but that's no problem).

Build the graph

It's not actually necessary to build the graph explicitly, but it makes the solution easier and more structured. This can be done by looping through all 129024 states above, and for each valid state generate all possible moves. Since we're about to do retrograde analyze, it's important that the edges in the graph go *backwards*; that is, if there's a legal move taking us from state A to state B, we add an edge in the graph from B to A. We also need to know how many legal moves there are from a state (note that this does not equal the number of edges from the state, since those edges are "backward" edges while the count is for "forward" edges - we don't need these edges, just the count).

Process the states

During the process of building the graph, all "mate" positions (positions with no moves) are found as well. These positions are marked as a "win" for one of the players and put in a queue. Also, we say that a state *y* is preceding a state *x* if there exist a legal move from *y* that will result in the state *x*.

Now, we process the queue and for each state *x* in the queue we can deduce one of the following:

- If the player to move in *x* is losing, this means that all states preceding *x* are winning states for the other player. Thus we mark all these states as a win for the other player, and add them to the queue.
- If the player to move in *x* is winning, then let *y* be a position preceding *x*. The player to move in *y* obviously don't want to play the move reaching *x*, so this reduces the number of non-losing moves. That is, we decrease the "legal moves" counter in *y*. But if this counter reaches 0, then there are no moves from *y* that will avoid a loss. Thus we mark *y* as a losing position for the player to move in *y*, and we add *y* to the queue to be processed.

When the whole queue has been processed, we have determined all winning and losing positions in the game. Finding out which move is the best one from a certain position is a simple matter of checking the evaluation of the position after each valid move from the given position and choose the one which is best.

About the test data

There are 10 test cases with different difficulty. Six of the cases are winning positions, which requires “thinking ahead” 1, 2, 3, 3, 4 and 5 moves, respectively. Three of the cases are losing positions, at depth 1, 2 and 4 moves, and one position is a draw.

A simple program simply finding an immediate win would solve one test case.

A naïve min-max search should be able to solve at least 4 cases in a reasonable amount of time (15 seconds).

An optimized min-max search should solve at least 6 cases.

It might be possible to solve all cases using min-max combined with dynamic programming, but this is more tricky than it first appears (what to do when, in the recursion, you encounter a position already on the stack?) and certainly harder than the suggested solution.

Using retrograde analyze, starting from endposition and working backwards, solves all cases. The sample solution needs about 4 seconds on an AMD 1200 MHz to do a complete evaluation of the game.

All input have a unique solution! Thus only a filecompare is required to verify the correct answer.

Important: A participant should not score for the “losing” or “draw” test cases unless at least half (3) of the cases with winning positions are correctly solved!! Otherwise a program with a single printf statement could “solve” 3 of the cases! **This is not mentioned in the current version of the problem statement.**

Task and solution are proposed by Jimmy Mårdell (jimmy@yarin.se)