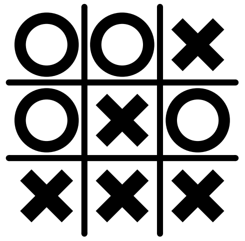
Retrograde Analysis

Introduction by Alain Brobecker, 2014

We solve a problem by starting from the final position
and by going through the game tree backwards.



Which move was played last?

Some games I created problems for, using retrograde analysis:

- ▷ Othello problems created in 2006
- ▷ Tic-Tac-Toe problems created in 2007
- ▷ Some fairy Chess problems in 2009
- ▷ Slidings a game made in PuzzleScript in 2010, challenges made in .c
- ▷ Grizzly Gears *(Author: Raf Peeters)*
- ▷ Secret Game 2023 *(Author: Raf Peeters)*
- ▷ Secret Game 2024, to do *(Author: Raf Peeters)*

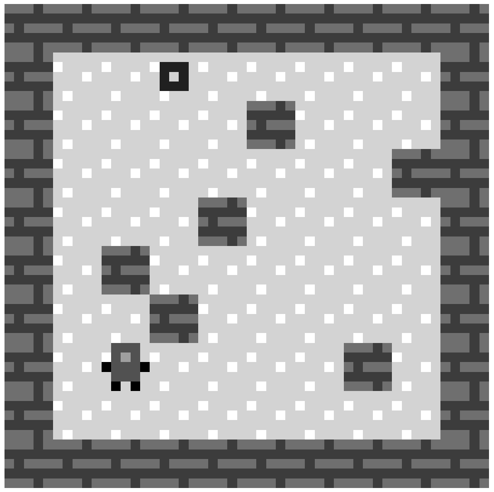
1965: Richard Bellman proposes to create a database to solve Chess and English Checkers by retrograde analysis.

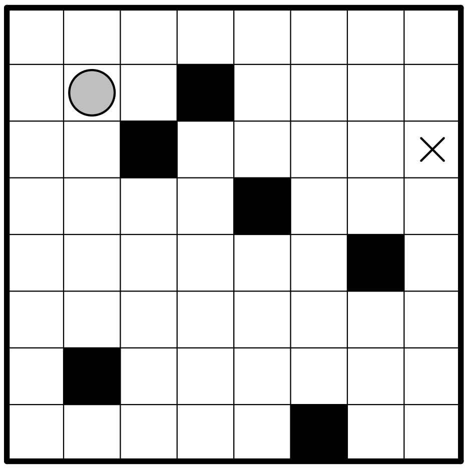
1970: Thomas Ströhlein publishes in his thesis the analysis of the following chess endings: ♔♕ / ♚ , ♔♖ / ♚ , ♔♙ / ♚ , ♔♕ / ♚♜ , ♔♖ / ♚♝ , ♔♖ / ♚♞
(note that ♔♕ / ♚♜ requires to know ♔♕ / ♚ and ♔♖ / ♚ ...)

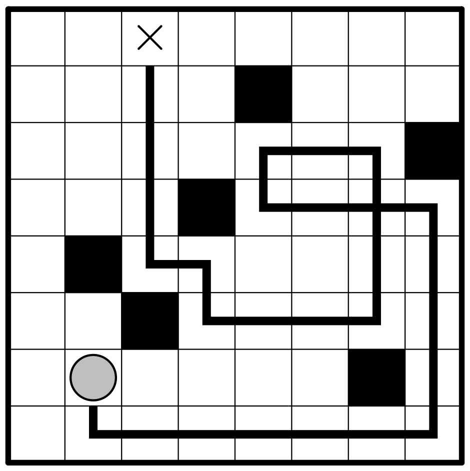
2007: Jonathan Shaeffer and his collaborators have solved the game of English Checkers, after 20 years of effort and using retrograde analysis and forward search: the game is drawn if both players play their best.

In 2014 some Chess endings were solved with 6 pieces and 7 pieces.

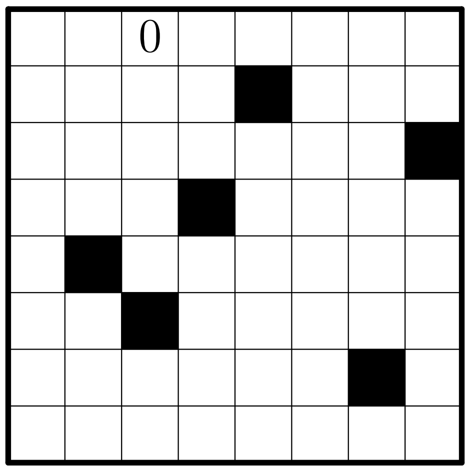
Slidyyyyy: We have to bring the little robot, which slides to the next obstacle, on the square target.



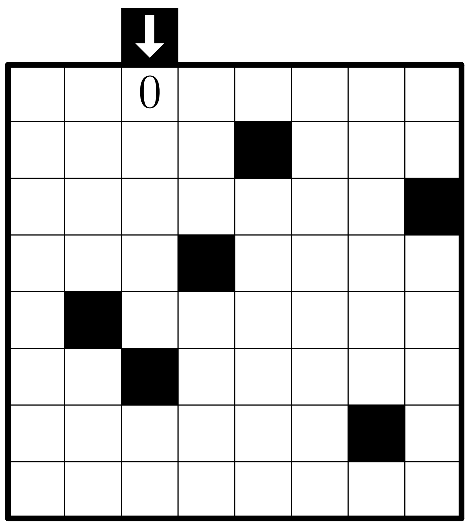




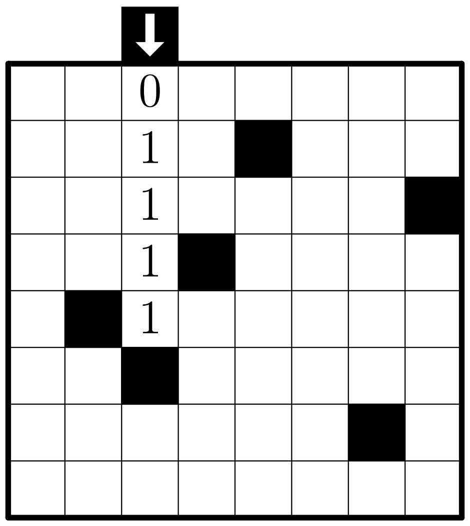
Solution in 11 moves.



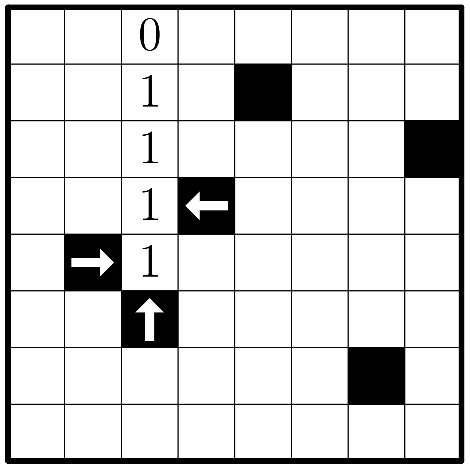
If the ball is on the target square,
we need 0 moves to solve the problem.

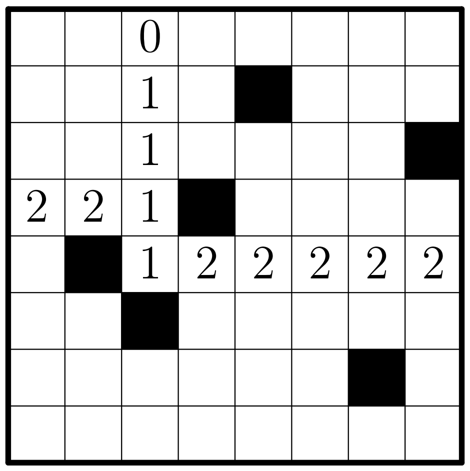


To reach the target square, marked with a 0,
it was necessary to come to a stop on the wall with the arrow.

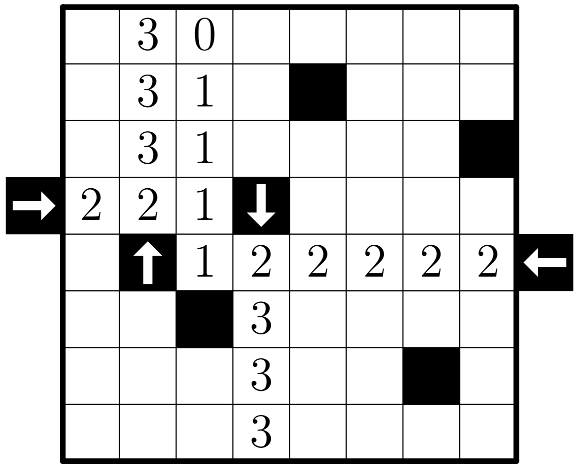


It’s possible when arriving from the squares marked with a 1.

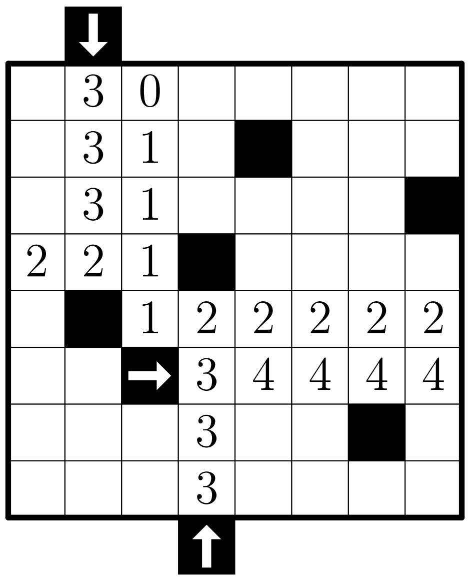

To reach the squares marked with a 1,
it was necessary to come to a stop on a wall with an arrow.



It’s possible when arriving from the squares marked with a 2.
We ignore the squares already marked to always have the minimum.



Etc...



Etc...

11	3	0	7	12	11	5	12
8	3	1	7	■	11	5	12
6	3	1	6	6	6	5	■
2	2	1	■	7	8	5	8
11	■	1	2	2	2	2	2
11	11	■	3	4	4	4	4
11	⑪	11	3	7	11	■	9
10	10	10	3	7	10	10	9

From the circled square it takes 11 move to reach the target square.

11	3	⓪	7	12	11	5	12
8	3	1	7	■	11	5	12
6	3	1	6	⑥	6	⑤	■
2	2	1	■	⑦	8	5	⑧
11	■	①	②	2	2	2	2
11	11	■	③	4	4	④	4
11	⑪	11	3	7	11	■	9
10	⑩	10	3	7	10	10	⑨

We still need to make a forward tracking through the saved game states to show the solution.

1) Identify the game elements

Identify the fixed elements (for a given problem):

- ▷ Playing area (board and obstacles)
- ▷ Aim of the game? (target space, connecting objects...)
- ▷ ...

Identify the variable elements:

- ▷ Whose move it is (in multiplayer games)
- ▷ Pieces that are moving (chess...)
- ▷ Pieces appearing/disappearing (chess, tic-tac-toe...)
- ▷ Pieces whose state changes (othello...)
- ▷ Target space?
- ▷ ...

2) Create the DataBase

Pass n°0:

For every possible position (combination of variable elements):

- ▷ Mark the illegal positions
- ▷ Mark the winning positions with a 0 distance (number of moves to win)
- ▷ Mark other positions with unknown distance

Pass n°N, as long as the number of positions with unknown status decreases:

For all positions with unknown distance, if it allows to go
to a position with known status (normally of distance N-1) by a legal move,
then we mark it as being of distance N.

The positions that stay with an unknown status donot allow to solve the problem.

3) Analyse the Database

In fact the hardest if you want to do it correctly?
We are face with a huge amount of data. Many problems will have multiple solutions, normally we should make a forward analysis of each problems to gather datas for them (number of solutions, "quality of moves"...). Out of all this we should try to assess which problems are interesting or potentially interesting:

- ▷ longuest possible problem
- ▷ problem with only one winning move at each step (allows to simplify the forward tracking)
- ▷ number of solutions
- ▷ switchbacks of a piece, circuit of a piece
- ▷ number of alternation between pieces
- ▷ a piece going to a specific place
- ▷ long moves/short moves
- ▷ ...

It might be a good idea to learn a bit about Structured Query Language.

Example input and trimmed output of Slidings_New.c

Input File:

```
.....
#.x..        x is the target
N2
```

Output File:

```
2 balls, 5x2 board -> 100 positions
#######
#.....#
##....#
#######

WinPos;StartPos;Solution;NbMoves;Alternation ratio;NbSwitchbacks;NbReplacements
N |  win[P] |forced[P]|     win |  forced | illegal |    left
0 |      16 |      16 |      16 |      16 |      28 |      56
1 |      18 |      16 |      34 |      32 |      28 |      38
2 |      16 |      10 |      50 |      42 |      28 |      22
3 |      14 |       6 |      64 |      48 |      28 |       8
4 |       6 |       2 |      70 |      50 |      28 |       2
5 |       2 |       0 |      72 |      50 |      28 |       0
c1;0a2,1b2;1b1,0e2,0e1,0c1;m=4;a=0.333;s=0;r=0;
c1;0a2,1d1;1d2,0c2,0c1;m=3;a=0.500;s=0;r=0;
c1;0e2,1e1;1b1,0e1,0c1;m=3;a=0.500;s=0;r=1;
c1;0b2,1e2;1c2,1c1;m=2;a=0.000;s=0;r=0;
c1;0b2,1e1;0b1,1c1;m=2;a=1.000;s=0;r=0;
c1;0d1,1e1;0b1,1c1;m=2;a=1.000;s=0;r=0;
```

Example input and trimmed output of Slidings_New.c

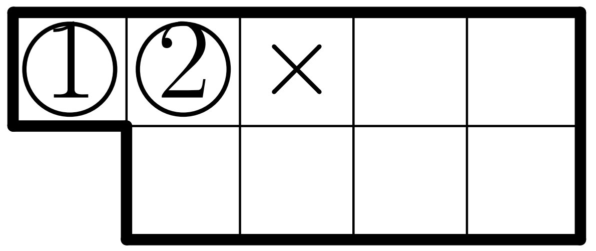
Input File:

```
.....        we don’t give the target
#....
N2
```

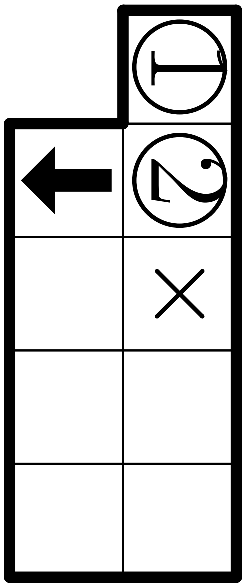
Output File:

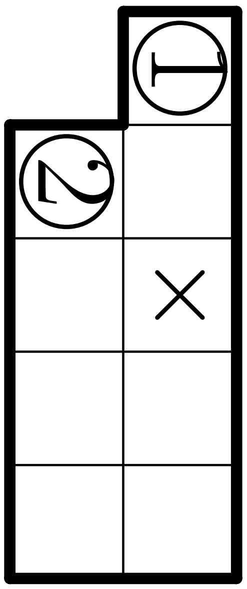
```
2 balls, 5x2 board -> 100 positions
#######
#.....#
##....#
#######

WinPos;StartPos;Solution;NbMoves;Alternation ratio;NbSwitchbacks;NbReplacements
c2;0a2,1b2;1b1,0e2,1b2,0c2;4;1.000;1;0      see on next pages
c2;0d1,1e1;0b1,1c1,1c2;3;0.500;0;0
d2;0b1,1c1;1e1,0d1,0d2;3;0.500;0;0
c1;0a2,1b2;1b1,0e2,0e1,0c1;4;0.333;0;0
c1;0a2,1d1;1d2,0c2,0c1;3;0.500;0;0
c1;0e2,1e1;1b1,0e1,0c1;3;0.500;0;1
d1;0a2,1b2;1e2,0d2,0d1;3;0.500;0;0
d1;0a2,1c2;1e2,0d2,0d1;3;0.500;0;0
d1;0b2,1c2;1e2,0d2,0d1;3;0.500;0;0
d1;0a2,1e1;1e2,0d2,0d1;3;0.500;0;0
d1;0e2,1e1;1b1,0e1,1d1;3;1.000;0;1
```

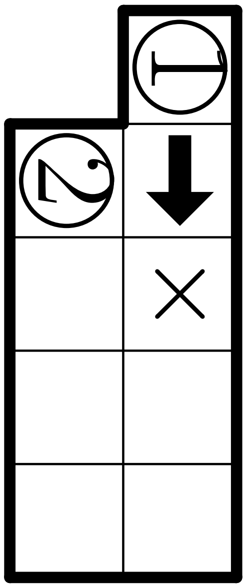
11 possible problems

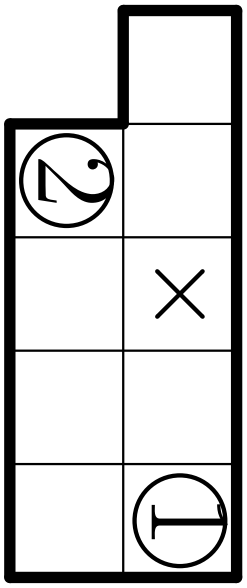


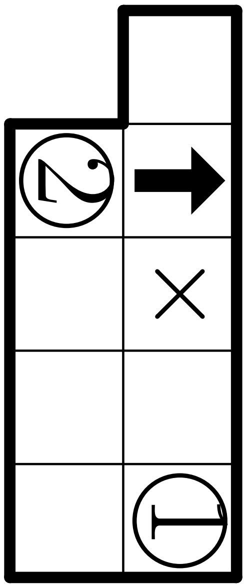
Balls move in a straight line to the next obstacle (a wall or another ball). The cross is the goal to reach, it is not an obstacle and therefore does not stop the balls. Yet the goal is that one of the balls stops on the cross in 4 shots. The solution is unique.

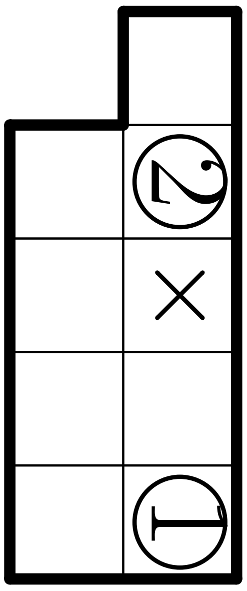

1
2

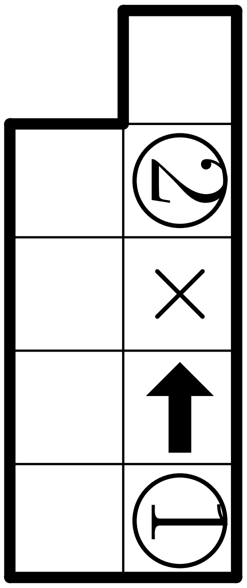

1
2
×

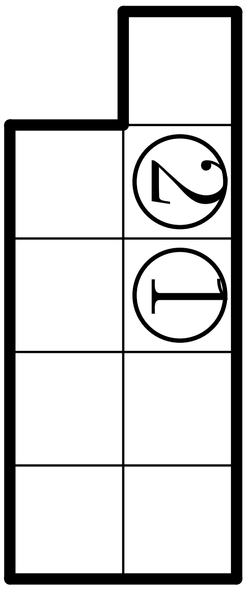

1
2


2
×
1


2
1


2
×
1


2
1


2
1