2019 FHSPS Playoff
May 18, 2019
Online

Filename	Problem Name	Time Limit
abknight	ab Knight	4 seconds
energizer	Energizer Knight	5 seconds
harkness	Harkness Rating System	1 second
king	King	1 second
move	Move Notation	1 second
onedim	One Dimensional k-Rooks	10 seconds
parallel	Parallel Movement	5 seconds
peaceful	Peaceful Bishops	10 seconds
queen	Queen Coverage	3 seconds
superpawn	Super Pawn	2 seconds

A Note and Thank You From the Problem Writer

To everyone involved with the 2019 Florida High School Programming Series Playoff,

I'd like to take this space to explain the theme of the problem set and thank everyone who has helped with the time consuming process of putting together this contest.

Chess is an ancient and classic game. I remember spending a great deal of my free time in elementary school, going to the library and playing chess with classmates. Though a young child can understand the rules, the game has such complexity and beauty that it can capture one's imagination for a lifetime. Compared to when I was growing up, I feel as if fewer children are exposed to chess in school. Due to its ability to teach logic and critical thinking, I feel that chess is an excellent activity for budding Computer Scientists and thought that this problem set would be an opportunity to promote chess, but also marry chess with one of my other loves: competitive programming. I hope you'll agree with me after this contest on the following statement: if we "loosen" some of the rules of the game, we can come up with some pretty creative and interesting competitive programming problems. As a long-time teacher, my style is to write competitive programming problems based on classically taught algorithms. Once I settled on the chess theme, I was pleasantly surprised as to how many of the algorithms we teach for contests can be incorporated into the game of chess. After the contest, I hope you have time to read my problem reviews and editorial on the contest.

I first and foremost want to thank Sharon Barak, who has written all of the previous FHSPS rounds this year and who wrote alternate solutions and verified a number of my problems. In addition to Sharon, Billy Quiroga also wrote alternate solutions and verified several problems in this problem set. I want to thank Kyle Dencker, my colleague who has partnered with me in so many Computer Science education ventures for continuing to organize and run FHSPS in spite of all of his other professional and family duties. Finally, I want to thank Glenn Martin, who wrote the judging system that today's contest will use.

Good luck in the contest and I hope you enjoy the problems!

Sincerely,

Arup Ratan Guha
5/18/2019

ab Knight
Filename: *abknight*
Time Limit: *4 seconds*

In chess, a knight is the only piece that can "jump" over pieces. Namely, a typical knight can move 1 square up, down, left or right, followed by 2 squares in a perpendicular direction. Thus, if a knight is at square (x, y), after its jump it can end up at one of these eight positions:

$(x \pm 1, y \pm 2)$, $(x \pm 2, y \pm 1)$, provided that they are in bounds and there isn't a piece on the destination square.

Now, consider a modified knight called an ab knight, where a and b are both positive integers, not necessarily distinct. From square (x, y), the eight squares the ab knight can reach on a single jump are $(x \pm a, y \pm b)$, $(x \pm b, y \pm a)$, provided that they are in bounds.

For the purposes of this problem, we'll assume that there are no pieces on the board except for the original ab knight. Our goal will be to calculate the fewest number of jumps necessary for the knight to reach each of the other squares on the board.

The Problem
Given the size of a chess board, the values of a and b for the ab knight, and the initial position of the ab knight, determine the fewest number of moves necessary for the ab knight to reach each of the squares on the board, or determine that some squares aren't reachable.

The Input
The first line of input will contain a single positive integer, n ($n \leq 20$), representing the number of input cases to process. The input cases follow, each taking up three lines. The first line of each input case contains two space separated positive integers, r ($3 \leq r \leq 100$) and c ($3 \leq c \leq 100$), representing the number of rows and columns on the chessboard for the input case. The second line of each input case contains two space separated positive integers, a ($a \leq \min(r, c)$) and b ($b \leq \min(r, c)$), representing the values of a and b for the ab knight for the input case. The third line of each input case contains two space separated positive integers, x ($x \leq r$) and y ($y \leq c$), representing the row and column of the initial location of the ab knight. Assume that the top left hand corner of the board is square is row 1, column 1, the bottom left hand corner of the board is square row r, column 1, the top right hand corner of the board is square row 1, column c, and the bottom right corner of the board is square row r, column c.

The Output
For each case, output r lines of c space separated integers, where the j^{th} value on the i^{th} line represent the fewest number of jumps necessary for the ab knight to travel from its initial square to row i column j. *Remember not to output a space after the last integer on each row.* If a square is unreachable by the ab knight, print out -1 instead.

Sample Input

```
2
3 3
1 2
1 1
2 5
1 1
1 2
```

Sample Output

```
0 3 2
3 -1 1
2 1 4
-1 0 -1 2 -1
1 -1 1 -1 3
```

Energizer Knight
Filename: *energizer*
Time Limit: *5 seconds*

As previously discussed in the ab Knight problem, a knight is the only piece that can "jump" over pieces. Namely, a typical knight can move 1 square up, down, left or right, followed by 2 squares in a perpendicular direction. Thus, if a knight is at square (x, y), after its jump it can end up at one of these eight positions: $(x \pm 1, y \pm 2)$, $(x \pm 2, y \pm 1)$, provided that they are in bounds and there isn't a piece on the destination square.

An Energizer Knight can make many, many jumps!!! Consider a single knight starting at square (x_1, y_1) and making exactly k jumps to end up at the target square (x_2, y_2). Naturally since the knight wants to jump around a lot it's allowed to hit the same square multiple times, even the starting square and the target square. The only thing that matters is that it ends up at the target square after precisely k consecutive jumps. In a sequence of consecutive jumps, the ending square of the previous jump is the starting square of the subsequent jump.

Typically, there are many paths of length k between any two squares on a standard 8 x 8 chessboard. For this question, you are asked how many different paths of k consecutive jumps there are between two squares (x_1, y_1) and (x_2, y_2). Since the answer to the query can been exceedingly large, you are to give your answer modulo 10007. We model a path as a sequence of locations $a_0, a_1, a_2, \ldots a_k$, where each a_i is a location on a standard chess board, $a_0 = (x_1, y_1)$, $a_k = (x_2, y_2)$, and for each i, $0 \leq i < k$, $a_i \rightarrow a_{i+1}$ represents a valid jump by a knight. Two paths $(a_0, a_1, a_2, \ldots a_k)$, and $(b_0, b_1, b_2, \ldots b_k)$ are considered different if there exists an i such that $0 \leq i \leq k$, where $a_i \neq b_i$.

The Problem

Given the starting square of a knight (x_1, y_1), a target square for a knight (x_2, y_2) and a number of consecutive jumps to perform, calculate the number of different paths the knight could take modulo 10007.

The Input

The first line of input will contain a single positive integer, n ($n \leq 20$), representing the number of input cases to process. The input cases follow, each on one line. Each case consists of five space separated integers: x_1 ($1 \leq x_1 \leq 8$), y_1 ($1 \leq y_1 \leq 8$), x_2 ($1 \leq x_2 \leq 8$), y_2 ($1 \leq y_2 \leq 8$), and k ($1 \leq k \leq 10^{18}$).

The Output

For each case, output a single line with a single integer representing the number of paths of exactly k jumps between the starting and ending squares modulo 10007

Sample Input	Sample Output
2 2 1 2 5 2 1 1 2 1 3	2 2

Harkness Rating System
Filename: *harkness*
Time Limit: *1 second*

One of the most straight-forward rating systems that has been used to rate chess players is the Harkness System, invented by Ken Harkness in 1956. The system adjusts players' ratings based on all of the results in a tournament after the completion of a tournament. We assume that before the tournament, each player has a rating. During the tournament, each player plays several matches and compiles a win-loss record. Note that an individual match can result in a win, loss or tie. Ties are recorded as ½ a win and ½ a loss. So a player who wins 2 matches, loses 1 match and ties 3 matches has a 3 ½ - 2 ½ win-loss record.

Each player's rating is adjusted based on her win-loss record AND the average rating of the opponents for each of the matches. In particular, let X be the average of a player's opponents, and let P be the percentage of matches won (this includes ties being counted as half a win). The player's new rating would be X + 10*(P - 50) For example, if the average rating of a player's opponents is 1700 and the player won 60% of her matches against those players, then her new rating after the tournament would be 1700 + 10*10 = 1800. If this results in a non-integer value, we take the floor of the calculation to record the rating.

Write a program to update all of the players' ratings in a chess tournament given the players' previous ratings and the participants and outcome of each match during the tournament.

Note: For this problem, if a player's rating is exactly an integer, if doubles are used in intermediate calculations, then it's possible that instead of 1400 being stored in the computer 1399.999999999998 will be stored instead, due to floating point error. One standard fix to the problem is to add a tiny offset (10^{-9} for example) to the final calculation and then take the floor of that value before outputting the rating.

The Problem

Given the initial ratings of several chess players in a tournament, the participants and outcomes of each match during the tournament, produce a sorted list of each player's rating after the tournament.

The Input

The first line of input will contain a single positive integer, n ($n \leq 20$), representing the number of input cases to process. The first line of each input case contains a single positive integer, p ($2 \leq p \leq 30$), representing the number of players in the tournament. The next p lines contain information about each player. Each of these lines contains a string of uppercase letters representing the name of a player, followed by a space, followed by a positive integer, r ($1000 \leq r \leq 3000$), representing the rating for that player. All of the names will be distinct. The following line of input will contain a single positive integer, m ($m \leq 1000$), representing the number of matches played during the tournament. The following m lines will contain information about each match, one match per line. Each of these lines will contain the name of the player with the white pieces, followed by a space, followed by the name of the player with the black pieces, followed by a space, followed either the letter 'W', 'L' or 'T' representing the outcome of the player with the white pieces.

The Output

For each case, output *p* lines. Each line will store information about one of the players in the tournament and these lines will be sorted by the rating of the players ***after*** the tournament. If two players have the exact same rating after the tournament, the tie (for output) is broken by their input ordering. (Whichever of the two people with the same rating who was listed first in the original input list should come first in the final output.) For each player, output their name, followed by a space, followed by their rating. Note that a player's rating may dip below 1000 or go above 3000 after the tournament, even though before the tournament, that was not true of any player.

Sample Input

```
2
5
MAGNUS 2882
GARRY 2851
FABIANO 2844
LEVON 2830
WESLEY 2822
6
MAGNUS GARRY W
MAGNUS FABIANO T
FABIANO LEVON W
LEVON GARRY T
WESLEY LEVON W
GARRY LEVON W
4
JENNA 1800
ANYA 1750
DYLAN 1600
SCOTT 1700
6
ANYA JENNA W
DYLAN SCOTT L
SCOTT ANYA L
JENNA SCOTT W
JENNA DYLAN L
DYLAN ANYA T
```

Sample Output

```
WESLEY 3330
FABIANO 3106
MAGNUS 3097
GARRY 2847
LEVON 2467
ANYA 2033
DYLAN 1750
SCOTT 1550
JENNA 1516
```

King

Filename: *king*

Time Limit: *1 second*

In chess, a king can move 1 space in any of the eight directions: up, down, left, right, up and left, up and right, down and left, and down and right.

We define the "space" of a king to include the square it's on as well as all of the squares it can move to in one move.

Consider trying to place kings on the board such that each square on the board is contained in the space of at least one king. What is the fewest number of kings necessary to achieve the task?

The Problem

Given the size of a chess board, determine the fewest number of kings which would have to be placed on the board so that each square is contained in the space of at least one king.

The Input

The first line of input will contain a single positive integer, n ($n \leq 20$), representing the number of input cases to process. The input cases follow, each on one line. Each input case contains two space separated positive integers, r ($3 \leq r \leq 100$) and c ($3 \leq c \leq 100$), representing the number of rows and columns on the chessboard for the input case.

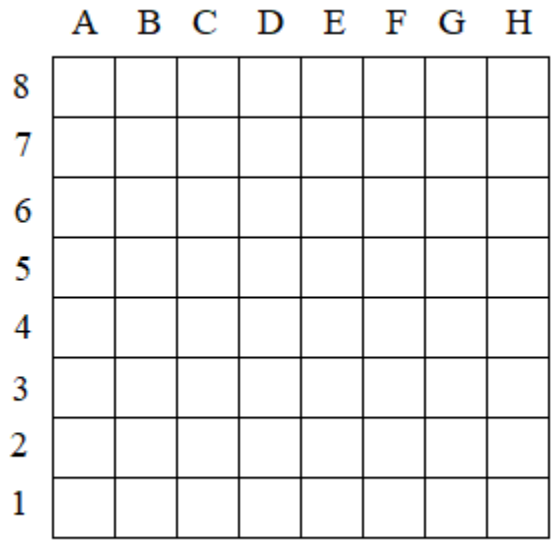
The Output

For each case, output a single line with a single integer representing the answer to the input case.

Sample Input	Sample Output
2 3 3 6 7	1 6

Move Notation
Filename: *move*
Time Limit: *1 second*

Chess has many written notations to save game transcripts. One of the easiest involves specifying the starting location of a piece that moves and an ending location of a piece that moves. A location on the chess board can be identified by its row and column. The rows are labeled 1 through 8 and the columns are labeled 'A' through 'H'. Here is a chess board with the rows and columns labeled (without the squares colored):



The most simple moves leave all squares untouched except for two. In these moves, a single piece vacates one squares and moves to a second square. If there was an opponent's piece on the second square, that piece is captured and the moving piece then occupies the square. If that square was unoccupied, then the moving then occupies that square. We can indicate each white piece as an uppercase letter and each black piece as a lowercase letter. In particular, a king is represented with 'K'/'k', a queen is represented with 'Q'/'q', a bishop is represented with 'B'/'b', a Knight is represented with 'N'/'n', a rook is represented with 'R'/'r' and a pawn is represented with 'P'/'p'. A blank square is indicated with the '.' character.

Given the initial position of the board and a subsequent position of the board for a simple move as described above, determine a written notation for that move. The written notation of a move starts with a two character representation of the location of the piece that moves (the upper case letter of the column followed by the number for the row), followed by a dash ('-'), followed by the two character representation of the location of the piece that moves after the move.

The Problem
Given an explicit description of the chessboard right before a simple move and right after a simple move, produce the written description of the move previously specified.

The Input

The first line of input will contain a single positive integer, n ($n \leq 20$), representing the number of input cases to process. The input cases follow. Each input case will take up 16 lines. The first 8 lines will describe the state of the board before a simple move and the next 8 lines will describe the state of the board right after that simple move. Each move will be valid.

The Output

For each case, output a single line of five characters of the form Mx-Ny, in the format previously described where Mx is the column, row description of the starting location of the piece that moves and Ny is the column, row description of the ending location of the piece that moves.

Sample Input

```
2
rnbqkbnr
pppppppp
........
........
........
........
PPPPPPPP
RNBQKBNR
rnbqkbnr
pppppppp
........
........
....P...
........
PPPP.PPP
RNBQKBNR
rnbqkbnr
ppp.pppp
........
...p....
....P...
........
PPPP.PPP
RNBQKBNR
rnbqkbnr
ppp.pppp
........
...P....
........
........
PPPP.PPP
RNBQKBNR
```

Sample Output

```
E2-E4
E4-D5
```

One Dimensional k-Rooks
Filename: *onedim*
Time Limit: *10 seconds*

A regular rook in chess can move as many squares up, down, left or right as it wants. Since this problem set looks exceedingly difficult, Kyle has asked Arup, the problem setter, to tone it down. Arup has adhered to the request and has determined that the chess board for this problem will be one dimensional, with 1 row only and c columns. In addition, each rook, instead of being able to move as many squares as possible has its own particular value of k such that it can only move k squares to the left or k squares to the right.

In this problem, several rooks are placed on the one dimensional board with length c, the number of columns of the board. The columns are numbered 1 through c, inclusive. For the purposes of the problem we assume that a rook's range isn't limited by other rooks in its way. A rook in column m, with range k can attack each square in the range $[\max(1, m - k), \min(c, m + k)]$, regardless of where other rooks may be.

We are curious about two things: (1) for any square on the board, what is the maximum number of rooks that attack it, and (2) how many squares on the board are attacked by this maximum number of rooks.

The Problem

Given the length of a one dimensional chess board, the placement of several rooks as well as their individual ranges (values of k), determine the maximum number of rooks that can attack any one square as well as the number of squares attacked by that many rooks.

The Input

The first line of input will contain a single positive integer, n ($n \leq 20$), representing the number of input cases to process. The input cases follow.

The first line of each input case contains two space separated positive integers, c ($c \leq 10^{18}$), representing the length of the chess board for the input case and r ($r \leq 10^5$), representing the number of rooks placed on the chess board. The following r lines each describe one rook.

Each of these lines will contain two space separated integers m ($1 \leq m \leq c$) and k ($1 \leq k \leq c$) representing that a rook is placed on column m and has a range of k. It is guaranteed that no two rooks will be placed on the exact same location.

The Output

For each case, output a single line with two space separated integers: the maximum number of rooks that attack a single square, and the number of squares on the board that are attacked by that many rooks.

Sample Input

```
2
10 3
5 1
3 2
9 4
100 2
10 50
70 30
```

Sample Output

```
3 1
2 21
```

Parallel Movement
Filename: *parallel*
Time Limit: *5 seconds*

Over the course of this problem set we've looked at the movements of all chess pieces. For this problem, we will consider a board with different types of pieces. All types except for pawns. Consider a board with several pieces all trying to move in parallel. The rules are that each piece must move from their current square to a currently unoccupied square that they would normally be able to move to, if the board were empty. Each piece that moves has to move to a ***different*** square that was previously unoccupied. Naturally, it's possible that only some of the pieces might be able to move in parallel. (As a trivial case, consider a 3 x 3 board with 5 pieces on it. With only 4 empty squares, at most, four pieces could move in parallel, following these rules, without looking at the restrictions dealing with piece movement.)

For example, for the board shown on the left, the maximum number of pieces that can move is 2. One sequence of 2 parallel moves that is possible is shown to the right:

	B	
B	R	B
R		N

Before

R'	B	
B	R	
	B'	N

After

In this picture, the rook on the bottom left moves to the top left. Note that the bishop doesn't get in the way because when we determine if we can move a piece to an unoccupied square, we imagine that the rest of the board is empty. The bishop in row 2 column 3 moves to row 3 column 2. There are other combinations of 2 parallel moves that can be made, but no combination of three parallel moves. To see this, just note that no piece on the board can move to the top right corner.

The Problem

Given a large chess board with several pieces (rooks, knights, bishops, queens and kings), determine the maximum number of those pieces that move in parallel to different unoccupied squares on the board. Consider the moves valid as long as the piece could make the move on an otherwise empty board.

The Input

The first line of input will contain a single positive integer, n ($n \leq 20$), representing the number of input cases to process. The input cases follow. The first line of each input case contains a single positive integer, s ($3 \leq s \leq 30$) representing the size of the chess board (both the number of rows and columns). Then, there will be s lines each with exactly s characters. The i^{th} of these lines show the contents of the i^{th} row of the board. Rooks will be represented by the character 'R', Knights will be represented by the character 'N', Bishops will be represented by the character 'B', Queens by the character 'Q', Kings will be represented by the character 'K', and unoccupied squares will be represented by the character '.'.

The Output

For each case, output a single line with a single integer representing the answer to the input case.

Sample Input

```
2
3
.B.
BRB
R.N
4
RRRR
B.B.
.B.B
.R..
```

Sample Output

```
2
7
```

Peaceful Bishops
Filename: *peaceful*
Time Limit: *10 seconds*

In chess, a bishop can move on a diagonal as many squares as possible as long as it does not run into another piece or run off the board. Thus, for a bishop at square (x, y), it can move to a square with coordinates ($x \pm k$, $y \pm k$) for any k, provided that the previous conditions hold.

As everyone knows, bishops are a peaceful people and would not prefer to attack anyone, least of all, other bishops. Unfortunately, bishops are territorial. If there is another bishop directly in their path (if they can move to another bishop's square in a single move) they will capture that bishop. Otherwise, bishops are wonderful pieces (and people), so we want as many of them on the board as possible!

In this problem, the goal will be to place as many bishops on the board as possible such that no two of the bishops can directly attack each other. Namely, no two bishops are allowed to be on the same diagonal. Of all such arrangements that maximizes the number of bishops, you are to find the k^{th} of these arrangements in lexicographical order.

We define an arrangement as a sequence of integers. On an n x n chess board, the squares on the first row are numbered 1 through n, on the second row the squares are numbered n+1 to 2n. In general, on row k, the squares are numbered (k-1)n + 1 to kn.

Between 2 different arrangements, ($a_1, a_2, \ldots a_m$), and ($b_1, b_2, \ldots b_m$), where $a_1 < a_2 < a_3 < \ldots < a_m$ and $b_1 < b_2 < b_3 < \ldots < b_m$, the sequence ($a_1, a_2, \ldots a_m$) comes first in lexicographical order if and only if for the minimal i with $a_i \neq b_i$, we have $a_i < b_i$.

The Problem
Given the size of a chess board as well as an integer, *k*, determine k^{th} lexicographically ordered arrangement which maximizes the number of bishops that can be placed on the board such that no two bishops are on the same diagonal.

The Input
The first line of input will contain a single positive integer, *n* ($n \leq 20$), representing the number of input cases to process. The input cases follow, each on one line. Each input case consists of two space separated positive integers: *s* ($s \leq 8$), representing the number of rows and columns of the chessboard for the input case, and *k*, the lexicographical rank of the arrangement desired. It is guaranteed that for the given value of *s*, there are at least *k* such arrangements of bishops.

The Output
For each case, the first line of output should be a single positive integer *m*, representing the maximum number of bishops that can be placed on the board without two being placed on the same diagonal. The following *m* lines should contain two space separated integers each *r* and *c* representing the placement of a single bishop in the k^{th} such arrangement in lexicographical order. These *m* locations themselves should be in lexicographical order (sorted by row, then sorted by column).

Sample Input

```
3
1 1
2 3
3 6
```

Sample Output

```
1
1 1
2
1 2
2 2
4
1 2
3 1
3 2
3 3
```

Queen Coverage
Filename: *queen*
Time Limit: *3 seconds*

In chess, a queen can move as many spaces as she wants in any of the eight directions: up, down, left, right, up and left, up and right, down and left, and down and right, so long as there are no other pieces in her way.

The Problem

Given a large chess board with several queens placed on it, how many squares on the board are either occupied by a queen or can be attacked directly by a queen? (Another way of thinking of the question is: how many squares on the board are already occupied by queens or could have a queen after one move by a queen?)

The Input

The first line of input will contain a single positive integer, n ($n \leq 20$), representing the number of input cases to process. The input cases follow.

The first line of each input case contains two space separated positive integers, s ($3 \leq s \leq 2000$) and q ($0 \leq q \leq 2000$), representing the size of the chess board (both the number of rows and columns) and the number of queens placed on the board, respectively. Then, there will be q lines, each with two space separated integers, x_i and y_i ($1 \leq x_i , y_i \leq s$) representing the row and column of the i^{th} queen.

The Output

For each case, output a single line with a single integer representing the answer to the input case.

Sample Input

```
2
3 1
1 3
4 2
1 3
2 2
```

Sample Output

```
7
14
```

Super Pawn
Filename: *superpawn*
Time Limit: *2 seconds*

Consider a game with only pawns, played on an n x n board where each team has a single pawn on each column of the board and the white pawn on each column is on a lower row number than the black pawn on that row. Each white pawn can be moved any number of squares "forward" or "backward" which means that it must stay in the same column and either increase or decrease its row number, but it must stop before it reaches the square of the black pawn on that column. (For a move to be valid, the piece can not stay in the same place.) Similarly, each black pawn can be advanced any number of squares "forward" or "backward" which means that it must stay in the same column and either decrease or increase its row number, but it must stop before it reaches the square of the white pawn on that column. White moves first. A team loses if it has no possible move left. As a simple example, in the game below, white can win by advancing the pawn in the third column 1 square:

W		W
B	W	
	B	B

The Problem
Given the size of the chessboard and the placement of each of the white and black pawns, determine which team will win, assuming that both teams play optimally.

The Input
The first line of input will contain a single positive integer, n ($n \leq 20$), representing the number of input cases to process. The input cases follow, each on one line. The first line of each input case contain a single positive integer, k ($3 \leq k \leq 10^5$), representing the number of rows and columns for the chessboard for the case (each board is square). The second line of input of each case contains k space separated integers. The i^{th} of these integers, w_i ($1 \leq w_i \leq k$-1), is the row number of the white pawn in column i. The third line of input of each case contains k space separated integers. The i^{th} of these integers, b_i ($w_i < b_i \leq k$), is the row number of the black pawn in column i.

The Output
For each case, output a single line with the string "WHITE" (no quotes) if white would win if both teams play optimally, or "BLACK", if black would win if both teams play optimally.

Sample Input

```
2
3
1 2 1
2 3 3
4
1 1 1 1
4 4 4 4
```

Sample Output

```
WHITE
BLACK
```

Sample Explanation: In the first sample game, white moves its third pawn forward. Black's only move is to move its first pawn backward. Then, white moves its first pawn forward. At this point, all three black pawns are pinned and can't move, so White wins.