A Knightly Pursuit

In chess, game pieces move about an chessboard in a fashion defined by their type. The object of the game is to capture opposing pieces by landing on their squares, and eventually trapping the king piece.

In our version of the game, we shall use a variable sized board with only 2 pieces on it: A white pawn which moves relentlessly towards the top row of the chessboard one square at a time per move; and a black knight which can move from its current location in any of up to eight ways: two squares up or down and one square left or right, or one square up or down and two squares left or right. The knight must remain on the board at all times; any move that would take it off the board is therefore disallowed. In the diagram below, the knight's position is labelled K and its possible moves are labelled 1 to 8.

```
. . . . . .
. . 8 . 1 . .
. 7 . . . 2 .
. . . K . . .
. 6 . . . 3 .
. . 5 . 4 . .
. . . . . .
```

The pawn moves first; then the knight and pawn alternate moves. The knight tries to land either on the square occupied by the pawn (a win) or on the square immediately above the pawn (a stalemate). If the pawn reaches the top row of the board the game ends immediately and the knight loses (a loss).

Input

The first line of input contains a positive integer, n, the number of games to analyze. For each game there are six lines on input:

r, the number of rows in the chessboard.
c, the number of columns in the chessboard.
pr, the row of the starting position of the pawn.
pc, the column of the starting position of the pawn.
kr, the row of the starting position of the knight.
kc, the column of the starting position of the knight.

All numbers in the input don't exceed 100. (Thanks to Blue Mary for pointing that out).

The pawn and the knight will have different starting positions. Row 1 is at the bottom of the board and Row r is at the top of the board. Column 1 is at the left and column c is at the right.

Output

If the knight can win and, output the minimum number of moves it must make to do so. If the

knight cannot win, your program should determine if it can cause a stalemate and, if it can, the minimum number of moves it must make to do so. Finally if the knight cannot win or cause a stalemate, your program should compute the number of moves the knight makes before the pawn wins.

Example

Input:

```
3
99
99
33
33
33
35
3
3
1
1
2
3
99
99
96
23
99
1
```

Output:

```
Win in 1 knight move(s).
Stalemate in 1 knight move(s).
Loss in 2 knight move(s).
```