Statement

We have a $d \times d$ chessboard. Both rows and columns of the board are numbered from 0 to $d-1$.

On the board are n rooks — some white, others black. For each rook you are given its row r_i, its column c_i and its type t_i: either 'W' or 'B'.

You want to make one valid move with a white rook, followed by one valid move with a black rook. Count the ways in which this can be done.

Rook moves

A rook moves horizontally or vertically, through any number of unoccupied squares. It may end its move either on an unoccupied square, or on a square occupied by a piece of the opposite color. In the latter case, the opposite-color piece is captured and removed from the board.

The rook must actually move — its destination square must be different from its starting square.

Input format

The first line of each input file contains the number t of test cases. The specified number of test cases follows, one after another.

Each test case consists of one or more lines.

- Line 1: the numbers d and n.
- Lines 2 to $n+1$: numbers r_i and c_i and letter t_i describing one of the rooks.

All rooks are guaranteed to be on mutually distinct squares of the chessboard.

Output format

For each test case, output a single line with a single integer: the desired number of ways, modulo 10^9+7.

Example

input

```
2
4 2
0 0 W
3 0 B
2 4
0 0 W
0 1 B
1 0 B
1 1 W
```

output

```
27
8
```

Test case 1: If the white rook moves within row 0 (3 possible moves), there are always six possible moves for the black rook. If the white rook moves one or two steps towards the black rook, the black rook will then have fewer possible moves. Note that the white rook could also take the black rook, but this would leave us with zero black rooks and thus zero ways to make a valid move with a black rook.

Test case 2: Either white rook can take either black rook (4 possiblilities). Then, the remaining black rook can either move to an empty square or take the white rook that did not move (2 possiblilities).

Subproblem R1 (20 points, public)

Input file: R1.in

Constraints:

- $t \leq 30$
- in each test case, $1 \leq d \leq 100$ and $1 \leq n \leq 100$

Subproblem R2 (33 points, public)

Input file: R2.in

Constraints:

- $t \leq 30$
- in each test case, $1 \leq d \leq 10^9$ and $1 \leq n \leq 5\,000$

Subproblem R3 (47 points, secret)

Input file generator: R3gen.py, R3gen.cpp

Constraints:

- $t \leq 30$
- in each test case, $1 \leq d \leq 10^9$ and $1 \leq n \leq 10^6$

The input file has about 500 MB, which is why we provide a generator instead of a direct download. Both versions of the generator (one in Python3, the other in C++) write the content of the file `R3.in` onto their standard output. (E.g., you can run `python3 R3gen.py > R3.in` to generate the input file locally.)

The generator has an internal checksum to verify that it ran correctly, but you can also double-check it manually: the last line of the correctly generated input should be "371311798 638638850 B".