The 5th Annual Fall University of Akron Programming Competition

presented by

The University of Akron Computer Science Department

Association for Computing Machinery Student Chapter

November 17, 2018

Rules:

1. There are six questions to be completed in four hours.
2. C, C++, and Java are the only languages available.
3. Data is read from Standard Input and output is sent to Standard Output. Do not prompt for input values in the code you submit to be judged. Do not attempt to read from or write to any files.
4. All programs are submitted as source code only. Submitting compiled information (binary, .class) will result in a Compilation Error penalty.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contest officials (e.g., that might generate a security violation).
6. The input to all problems will consist of multiple test cases. The Sample Input listed on the problem description is not intended to be a comprehensive input set. The input is guaranteed to adhere to the descriptions in each problem; you do not need to check for invalid input.
7. The output to all problems should conform to the format shown in the Sample Output for that problem, including but not limited to capitalization, whitespace, etc.
8. Use of personal electronics during the competition is cause for disqualification. Cell phones must be turned all the way off (not silent mode, not airplane mode, etc.). You may use any written notes, books, or reference materials you bring with you.
9. Programming style is not considered in this contest. You can code in any style or with any level of documentation your team prefers.
10. All communication with the judges will be handled through PC^2. All judges' decisions are final.
11. Source code is compiled on the command line, C and C++ will be compiled with GNU G++ (`g++ sourcefile.cpp`) and Java with the standard Java RE (`javac sourcefile.java`).

A: Squares

A typical chess board consists of a grid of 8 by 8 squares. That leaves it with 64 squares that pieces can be moved to. How many squares of all sizes can be created when given an n by n chess board?

Input

The only line of input contains one integer, n ($1 \leq n, \leq 2000$).

Output

On a single line, output the total number of squares that exist on the board

Sample Input	Sample Output
1	1

Sample Input	Sample Output
2	5

B: Chessymmetry

Juniper is a big fan of symmetry and chess. A chess board consists of an 8 by 8 grid. Juniper is interested in chess boards that are completely filled with pieces from the game of chess such that the board is both horizontally and vertically symmetric. Juniper is only interested in pawns, bishops, rooks, knights, and queens.

A board has two axes, one that runs through the center of the board between the fourth and fifth columns and one that runs through the center of the board between the fourth and fifth rows. The axis between columns is called the horizontal axis, and the axis between rows is called the vertical axis. A board is considered horizontally symmetric if every cell of the board after folding it over the horizontal axis is equal. Note that after doing a fold of the board there will technically be two pieces occupying the same space, it is these two pieces that must be equal. Two pieces are said to be equal if they are the same kind, that is, pawns are equal to pawns, bishops are equal to bishops, and so on. A board is considered vertically symmetric if every cell of the board after folding it over the vertical axis is equal.

Of course, as Juniper has already told you, he is interested in boards that are both horizontally *and* vertically symmetric. He has enlisted your help in figuring out if a given configuration of a chess board (with every space occupied by a piece) is horizontally and vertically symmetric.

Input

There are 8 lines of input that consist of 8 space separated characters. These characters can be either `P`, `B`, `R`, `N`, or `Q`.

Output

Output either `YES` if the board is horizontally and vertically symmetric, or `NO` otherwise.

Sample Input	Sample Output

`P Q N B B N Q P`
`N P R P P R P N`
`R B Q R R Q B R`
`Q P R P P R P Q`
`Q P R P P R P Q`
`R B Q R R Q B R`
`N P R P P R P N`
`P Q N B B N Q P`
	`YES`

Sample Input	Sample Output

`P Q N B B N Q P`
`N P R P P R P N`
`R B Q R R Q B R`
`Q P R P P R P Q`
`Q P R P P R P Q`
`R B Q R R Q B R`
`N P R P P R P N`
`P Q Q B B Q Q P`
	`NO`

C: Rookie Mistake

In a game of chess, the rook piece is able to move in a single straight line up, down, left, or right but not diagonally. The rook can move in this direction as many places as it likes until it is either: 1. Blocked by another piece on his team then the rook must stop in the square on the board before colliding with the teammate. 2. Hits the edge of the chess board then the rook must stop on the last piece. 3. Collides with a piece on the enemy team then the rook will kill the enemy player, and take its place on the board.

It appears that after you thought you won the game, your rook forgot to kill the enemy's king. Since the king is not dead, you know one of your rooks is to blame. This means you know you have at least one rook on the board but in a game of chess, you can have up to two total rooks. Figure out where your rook is and remind him to finish the objective. If that rook is unable to kill the king find your second rook and make sure the job is completed. Make sure you do not kill your own king.

The position of pieces are determined from the X and Y positions of the board where top-left=(0,0) top-right=(7,0) bottom-right=(7,7)

Note to players that understand the real game of chess: These situations may not be valid game ending moves in the real game of chess but these are just fun little problems.

Input

The input consists of 8 lines containing 8 space separated letters. The letters are:

R: A rook is occupying this position

K: Your king is occupying this position

$: The enemy king is occupying this position

N: There are no pieces occupying this position

Output

On a single line that is space separated:

Output the X position of the rook you wish to move (zero indexed)

Output the Y position of the rook you wish to move (zero indexed)

Output the direction (U, D, L, R) you wish to move the rook in (Up, Down, Left, and Right respectively)

Output the number of spaces the rook must move to kill the king.

Sample Input	Sample Output	Sample Input	Sample Output
N N N N N N N N N N N K N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N R N N N N $ N N N N N N N N N	1 6 R 5	N N N N N N N N N N N N N N N N N N N N N N N N N R N K N $ N N N N N N N N N N N N N N N N N N N N N N N N K N N N N N N R N N	5 7 U 4

D: MMORPG

Your company is creating an MMORPG (a Massively Multiplayer Online Role-Playing Game) - for chess. What the heck that will look like is anyone's guess (never let business people decide what games should be made). Now, it wouldn't be much of an online game if you couldn't play online. The fools who are implementing it decided to use UDP, which has turned out to be quite problematic. Their game's been having lots of reliability issues, so your task is to write a tool to help them figure out what's been happening to their data.

UPD is an internet protocol for sending packets. A packet is a small chunk of data to send, along with some metadata. Your game transmits a single packet to the server per frame. Why the metadata? Well you see, even though it's really fast, UDP suffers from several reliability problems. Sometimes packets arrive out of order. Sometimes duplicate packets arrive. Sometimes their data's corrupted. Even worse, sometimes they're dropped and don't arrive at all.

To solve the arriving-out-of-order problem, packets include a sequence number. The first packet sent is assigned a sequence of 0, the second packet a sequence of 1, and so on. To handle packets with duplicate sequence numbers, we just keep the first one we got and ignore the duplicates. To handle corruption, packets include a checksum that was calculated before it was sent. A checksum is the sum of all the ASCII values of the packet's data. For example, if we had the string "cat", 'c' (ASCII 99) + 'a' (ASCII 97) + 't' (ASCII 166) = 312. The packet is considered corrupted if its checksum does not match its pre-transmission value.

To know whether a packet is dropped, we use a Time To Live (TTL) duration. A TTL specifies how long, in number of frames, we're willing to wait before concluding it's been dropped. For example, suppose you had a TTL of 5 and we'd only received packets 0, 1, and 10. Since your TTL is 5, you can deduce that packets 2, 3, and 4 were dropped. After this, you would no longer accept any packet with a sequence less than 5.

Your job is to help your incompetent coworkers find out what happened to each packet. How else are they going to get a raise and make more money than you?

Input

The first line of input consists of one integer, T ($0 \leq T \leq 1000$), the time to live for all packets. Following is a stream of up to 1000 packets. The ith packet consists of a single line containing three things: an integer SEQ[i] ($0 \leq$SEQ[i]≤ 1000), the packet's sequence number, a string with no whitespace $DATA[i]$, the data sent in the packet, and an integer SUM[i] ($0 \leq$SUM[i]$\leq 10^9$), the checksum of the data. The stream of packets ends with a special packet that has its $DATA$ equal to end. It's guaranteed that this packet will arrive last and will have the largest sequence number.

Output

For each packet in the sequence, output its sequence number, followed by a colon and a space. If the packet was dropped, print "dropped". If the packet was received but was corrupted, print "corrupt". Otherwise, print "valid" followed by the data in the packet.

Sample Input	**Sample Output**
5	0: valid hello
0 hello 532	1: valid pawn
2 rook 443	2: valid rook
1 pawn 438	3: corrupt
2 rook 443	4: dropped
3 queen 99999	5: valid end
5 end 311	

Sample Input	**Sample Output**
4	0: valid Ne5
0 Ne5 232	1: dropped
6 Rd4 235	2: dropped
0 Ne5 99999	3: dropped
1 Ke7 231	4: dropped
7 end 311	5: dropped
	6: corrupt
	7: valid end

E: Che-xploration-ss

Albert does not really know much about chess and would like to explore the game to learn more. As a challenge, Albert's friend decided to give him a board with obstacles and a single piece. Albert was tasked to find the minimum number of moves needed to reach any other cell. However, Albert negotiated with his friend that if the minimum number of moves would be larger than 9, then he would report the number as `*` instead.

Of course, Albert does not know how any of the pieces move! His friend decided that he should only learn about the knight, the bishop, the rook, and the queen to start. Recall that the knight is allowed to move two spaces horizontally or vertically and then one space in the opposite horizontal or vertical direction, forming an L shape. The rook is allowed to move vertically or horizontally as much as it wants while staying on the board. The bishop is allowed to move diagonally (both left and right) as much as it wants while staying on the board. The queen can move as either a rook or a bishop. An obstacle on the board will be represented by an `X` and will prevent movement. You can think of this obstacle as an immovable pawn under your own control (i.e. you can not capture it).

Unfortunately, neither Albert or his friend are very good at making sure Albert's answers are correct. That is where you come in. Albert's friend has asked you to write a program that will produce the correct answer so that Albert can easily check his attempt.

Input

The first line of input consists of one character that is either `B`, `R`, `N`, or `Q` representing if the piece is a bishop, rook, knight, or queen respectively followed by two integers, u and v ($0 \leq u, v < 8$), the starting position of the piece. The following 8 lines of input consist of 8 space separated characters. These characters are either `.` or `X` representing an open space and an obstacle respectively.

Output

Output 8 lines with 8 space separated characters each that represents the input grid with labels for the minimum number of moves to reach each location.

Sample Input

```
Q 5 4
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
```

Sample Output

```
2 2 2 2 1 2 2 2
1 2 2 2 1 2 2 2
2 1 2 2 1 2 2 1
2 2 1 2 1 2 1 2
2 2 2 1 1 1 2 2
1 1 1 1 0 1 1 1
2 2 2 1 1 1 2 2
2 2 1 2 1 2 1 2
```

Sample Input

```
R 0 0
. X . . . . X .
. . . X X . X .
X X X . . . X .
. X . . X X . .
. X . X . . . X
. X . . . X . .
. . X X X . X .
. . . X . . . .
```

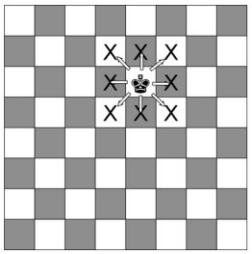
Sample Output

```
0 X 3 4 4 4 X *
1 2 2 X X 5 X *
X X X 6 6 5 X *
. X 8 7 X X * *
. X 9 X * * * X
. X 9 * * X * *
. . X X X * X *
. . . X * * * *
```

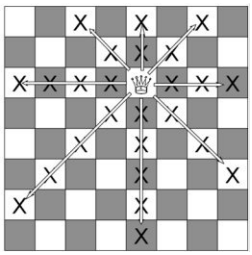
F: Checkmate In One

There are several different types of pieces in chess. We're going to focus on three for this problem: kings, queens, and knights. Below is a description of each piece for those who need a review of the rules. Recall that chess is played on an 8x8 board with pieces that are two colors, typically black and white.

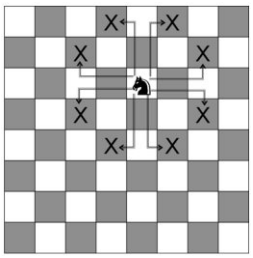
The king is the most important piece in the game. Its move and attack pattern can be seen below. It can move/attack any square that's next to it. Unlike the other pieces, it cannot be captured. It can, however, be attacked (this is called being in check). It's illegal to make any move that puts your own king in check. If the king is in check, the player's next move must be to move out of check. If this is not possible, the player loses (this is called checkmate).



Queens are the most powerful piece in the game. Its move/attack pattern can be seen below. They can move as many spaces they want in any single direction, unless there is a piece blocking their way. That is, queens cannot move or attack through other pieces.



Knights are the only piece that can jump over other pieces. They move/attack in an L-shape, jumping over any pieces it takes to reach that square. They move two spaces in one direction and one space in another.



When a piece is captured, the piece is removed from the board and the attacking piece moves to that location. Pieces can only capture pieces of the opposite color. Under no circumstance can a king be captured. A piece cannot be moved if doing so puts their own king in check.

In our problem, you play as white, an arrogant snob who's wiping the floor with your opponent. Black only has a single piece left: their king. It's your turn. Black is determined to drag out the game forever, but

you've got a hot date later and you don't want to miss it for chess. Again. So your goal is to write a program that answers this simple question: is it possible to checkmate black after a single move?

Input

Input consists of eight lines, describing the board and pieces on it. Each lines consists of eight characters representing the pieces, and seven spaces separating them. White's king, queens, and knights are represented by the characters 'K', 'Q', and 'N' respectively. Black's king is represented as '$', and an empty space is represented as '.'. It is guaranteed that there is exactly one white king and one black king on the board.

Output

Output `yes` if you can checkmate black in one move, or `no` otherwise.

Sample Input

```
. . . . . . . Q
. Q . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
. . . . . . . .
$ . . . . . . K
```

Sample Output

```
yes
```

Sample Input

```
N . . . . . . N
. . . . . . . .
. . . . . . . .
. . . . $ . K .
. . . . . . . .
. . N . . . . .
. . . . . . . .
Q . . . . . . N
```

Sample Output

```
no
```

Sample Input

```
K . . . . . . .
. . . . N . . .
Q . . . . . . .
. . $ . . . . .
Q . . . . . . .
. . . . . . . .
. . . . . . . .
. . N . . . . .
```

Sample Output

```
yes
```