2015/2016 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 1
The Magical 3

Three is a magic number.
Yes it is; it's a magic number.
Somewhere in the ancient, mystic trinity,
You get three as a magic number.
— *Schoolhouse Rock*

According to Pythagoras and the Pythagorean school, the number 3—which was named "triad"—is the noblest of all digits, as it is the only number to equal the sum of all the terms below it. It is also the only number whose sum with those below equals the product of them and itself ($3 + 2 + 1 = 3 \times 2 \times 1$). Your task is to find the magic—the magic 3, that is—when it can occur as the last digit in a representation of a positive integer in some base. Consider, for example, the number eleven. It can be represented as $1\mathbf{3}_8$ and $2\mathbf{3}_4$. Your team is to write a program that will find the smallest base for a given positive integer where the integer's representation in that base ends in 3.

Each line of input to your program will contain one positive integer less than 2^{31}, expressed in base ten, starting in the first column. Input to your program will terminate with the end-of-file.

For each number in the input, print a line containing the smallest base for which the input number has a representation that ends in 3, or the message "No such base" if there is no base that has a representation of the number which ends in 3. No leading or trailing whitespace is to appear on an output line.

Sample Input

```
11
123
104
2
3
2103723004
```

Output for the Sample Input

```
4
4
101
No such base
4
2103723001
```

2015/2016 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 2
Liar, Liar, Pants on Fire

The Swamp County Sheriff's Department (SCSD) gathers extensive witness statements during crime investigations. Unfortunately, insufficient manpower is available to analyze witness statements. The secretarial pool has transcribed witness statements into timelines. Your team has been hired to write a program that will compare timelines between witnesses. Your program will compare each timeline with all others to identify inconsistencies that merit further investigation.

Input to your program is a series of witness statement blocks for a single 24-hour period (00:00–24:00). The first line of each witness statement block begins with a plus sign followed by the witness name (one to thirty printable characters, not including commas). The remaining lines within the block are of the form:

startTime,stopTime,location,personBeingReportedOn

where:

- *startTime* and *stopTime* are in 24-hour HH:MM format—*stopTime* will be after *startTime*,
- *location* is a string of one to thirty printable characters without any commas,
- and *personBeingReportedOn* is a string of one to thirty printable characters without any commas, terminated by end-of-line.

A witness can account for his/her own whereabouts or a different person's whereabouts. The person being reported on may or may not be a witness who has given statements to the SCSD. Witness statements are not sorted by time. There will be no more than fifty distinct people named in the various witness statements.

Your program is to construct a timeline of the whereabouts of each witness who made statements to the SCSD for the full 24 hours, in chronological order, based on the statements of all witnesses. Witness reports are to appear in the same order as they appear in the witness statement input. The timeline for each witness is to begin with a line starting with a "+" followed by the name of the witness. Each distinct block of time is to be reported by printing a line with the *startTime,stopTime* range, a single space, and one or more of of the following codes: "S" for self-reported, "A" for accounted for by another witness's account, "U" for unaccounted for, and "C" for any time period for which there are conflicting statements. Multiple codes are to be printed in ascending alphabetical order. Contiguous blocks of time with the same set of codes are to be reported as a single *startTime,StopTime* range. No leading or trailing whitespace is to appear on an output line.

Problem 2
Liar, Liar, Pants on Fire (continued)

Sample Input

```
+Benjamin
07:30,08:30,Benjamin Residence,Benjamin
08:30,10:30,Las Vegas Strip,Benjamin
09:00,10:00,Las Vegas Strip,Sam
09:00,10:00,Las Vegas Strip,Francis
+Sam
01:00,08:00,Bally's Bar,Sam
08:15,10:30,Las Vegas Strip,Benjamin
09:00,10:00,Las Vegas Strip,Benjamin
02:00,07:00,Bally's Bar,Francis
07:00,10:30,Las Vegas Strip,Sam
+Francis
02:00,07:30,Bally's Bar,Francis
08:00,11:00,Las Vegas Strip,Francis
07:30,08:00,Downtown,Francis
09:00,10:00,Las Vegas Strip,Benjamin
```

Output for the Sample Input

```
+Benjamin
00:00,07:30 U
07:30,08:15 S
08:15,08:30 ACS
08:30,10:30 AS
10:30,24:00 U
+Sam
00:00,01:00 U
01:00,07:00 S
07:00,08:00 CS
08:00,09:00 S
09:00,10:00 AS
10:00,10:30 S
10:30,24:00 U
+Francis
00:00,02:00 U
02:00,07:00 AS
07:00,09:00 S
09:00,10:00 AS
10:00,11:00 S
11:00,24:00 U
```

2015/2016 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 3
Rolling Correlation

The *correlation* of two time series X and Y is defined as:

$$\rho = Corr(X,Y) = Cov(X,Y)/\sqrt{Cov(X,X)Cov(Y,Y)}$$

where $Cov(X,Y)$ is computed as

$$Cov(X,Y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

and $\bar{x}$ is the arithmetic mean of all xs. Similarly, $\bar{y}$ is the arithmetic mean of all ys.

(Note that the correlation is *undefined* when one or both of the variables are constant.)

For the purpose of this problem, we will define the correlation as *noticeably positive* if $\rho \geq 0.001$.

If X and Y are two long time series of data, instead of computing the correlation for the whole series, sometimes it is useful to look at the correlation within a particular time window w.

For each $i \geq w$ we can compute the correlation of X^i and Y^i, where:

$$X^i = \{x_{i-w+1}, x_{i-w+2}, \ldots, x_{i-1}, x_i\}$$

and

$$Y^i = \{y_{i-w+1}, y_{i-w+2}, \ldots, y_{i-1}, y_i\}$$

The *rolling correlation* is the series of correlations over time for a sliding time window.

In this problem we want to compare a time series X_1 to $(p-1)$ other time series $X_2, \ldots, X_p$. Your team's program is to determine the percentage of the time that the rolling correlation for a given time series pair is noticeably positive.

The first line of the input will contain three integers: the size of each time series $n \leq 10^6$, the number of variables $p \leq 100$, and the time window $w \leq n$, separated by single spaces. The next n lines will contain observations for each series at subsequent time points, also separated by single spaces. All observations will be floating point numbers.

The output should contain $(p-1)$ lines, each with a percentage value describing the percentage of the time the rolling correlation of X_1 and X_k (for $k = 2, \ldots, p$) is noticeably positive. Each percentage value should be rounded to two digits after the decimal point and immediately followed by a percent sign. No leading or trailing whitespace is to appear on an output line.

Problem 3
Rolling Correlation (continued)

Sample Input

```
11 3 5
1 5.5 -5
2 6.5 -4
3 7.5 -3
4 8.5 -2
5 9.5 -1
6 10.5 0
7 11.5 -1
8 12.5 -2
9 13.5 -3
10 14.5 -4
11 15.5 -5
```

Output for the Sample Input

```
100.00%
42.86%
```

2015/2016 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 4
Connect

The "Manhattan" or "taxi" distance between any two points on a grid is defined as the shortest distance measured along the grid lines. For example, given a one-unit grid and the points A, B and C on the diagram shown as Figure 1, the distances are as follows:

A to B:	3
A to C:	4
B to C:	1
B to A:	3
C to A:	4
C to B:	1

An interesting question then is: Given n distinct points on a grid, how many distinct shortest distances are there between every pair of points? In the example above, the possible distances are (1, 3, 4) so the answer to the question is "three distinct distances." (As more points are added, eventually that number will not increase as fast as n.) Your team is to write a program that will determine the number of distinct distances between selected points on a grid.

Input to your program will be a map represented by an array of characters terminated by end-of-file. The points are "x" characters and spaces are "0" characters. The map is a square: the number of characters on each line will be equal to the number of lines. The maximum size of the map is 80×80. The distance between each adjacent character counts as one unit of distance. The grid in Figure 1 would be represented as:

```
000000
00x000
000000
000x00
000x00
000000
```

Your program's output is to be the count of distinct shortest distances between all pairs of points on the map. Print the count on a line with no unnecessary leading zeroes and no leading or trailing whitespace.

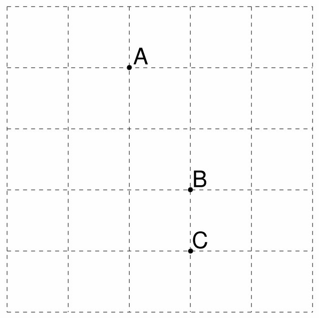


Figure 1. Example "Manhattan Distance" Grid.

Problem 4
Connect (continued)

Sample Input

```
00000000000000000000
000x00x0000000000000
000000000000000000x0
0x000000000000000x00
0000000000000000000x
00000000000000000x00
00000000000000000000
00000000000x00000000
00000000000000000000
0x00000000000000x000
0000x000000000000000
00x0000000000x000000
00x000000x000x0x0000
0x00000000x000x00000
00000000000x00000000
0000000000000000000x
00000000000000000000
00000000000000000000
0000x0000x0000000000
00000000000x00000000
```

Output for the Sample Input

```
30
```

2015/2016 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 5
Card Chain

Card Chain is a game of cards for two players, using a standard 52-card deck that has 4 suits (spades, hearts, diamonds, and clubs) of 13 cards each (the numbers 2 to 10, jack, queen, king, and ace). Each player is dealt a hand from a shuffled deck; each player in turn discards as many cards as possible during her turn. The discard process follows a set of rules (given below). The winner is the first player to discard her last card.

Given a particular pair of hands, there may be multiple possible outcomes. Your team is to write a program that will analyze a pair of hands to find the optimal discard strategy for each player and print the resulting score.

Each card will be represented by two ASCII characters as follows:
1st character (rank): `23456789TJQKA` - for 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, king, ace.
2nd character (suit): `shdc` - for spades, hearts, diamonds, clubs.
Examples: `6h` for the 6 of hearts, `Ac` for the ace of clubs, `Td` for the 10 of diamonds, etc.

The process of discarding cards is referred to as *chaining*. Multiple cards can be chained together, as long as every card follows one of these rules:

- A card being discarded has the same rank as the previous card in the chain (e.g., `8h-8s`).
- A red card being discarded is of the same suit and a higher rank than the previous card in the chain (e.g., `3h-7h`, `9d-Kd`).
- A black card being discarded is of the same suit and a lower rank than the previous card in the chain (e.g., `Ks-5s`, `7c-4c`).

For the purposes of chaining, aces are special as they can have both the lowest and the highest rank:
`A < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < T < J < Q < K < A`

Here are two examples of legal card chains: `8h-8s-5s-3s-3d-5d-Kd`, `Td-Kd-Ad-3d-Jd-Js-5s-As-Qs`.

Play proceeds as follows: Two players are each dealt N ($3 \leq N \leq 13$) cards turned up so that both players can see both hands. One additional card is turned face up to start a pile on the table. The rest of the deck is not used.

The two players in turn try to discard their cards onto the pile as fast as possible, following these rules:

1. When it is a player's turn, she must continue a chain based on the card on the top of the pile if it is possible to do so. She must continue chaining cards until no card can be chained to the previous one. She is free to choose between the multiple possible chains that can be discarded; but she must play one of them, and cannot stop a chain early. When she can no longer legally discard any of her cards, the turn passes to the other player.

2. If a player cannot chain any of her cards to the last pile card during her turn, she must discard exactly one card according to the rules below, and then the turn passes to the other player:
   - A black card can be discarded if it is the highest ranked black card (in case of a tie, either card may be discarded).
   - A red card can be discarded if it is the lowest ranked red card (in case of a tie, either card may be discarded).
   - Note that in this situation a red ace must be treated as the lowest-ranked red card, and a black ace must be treated as the highest-ranked black card.

The game ends when one player discards her last card, and the points she gains are equal to the number of cards left in the other player's hand.

Problem 5
Card Chain (continued)

Input to your program will have the number N on the first line, the cards dealt to the first player separated by spaces on the second line, the cards dealt to the second player separated by spaces on the third line, and the initial pile card on the fourth line. Multiple games can be given in the same input file. All input will begin in the first column of each line.

Your program is to print the optimal final score for each game on a separate line without leading or trailing whitespace. Print the final score as a positive number if the first player is guaranteed to win. Print the final score with a leading minus sign if the second player is guaranteed to win.

Sample Input

```
10
Th Ad Qd Qs Ts 9s 8s 6s 4s 2s
As Js 7s 2h 6h 6d 7d 8d 9d 8c
Ac
3
As Qs 3s
Ad Qd 3d
Jh
```

Output for the Sample Input

```
3
-2
```

Analysis of the first game:

Note that in this example, player one has only one choice for the first card to play. Her only legal play is to start with `Ad`. After that she must follow with `Qd-Qs-Ts`. At this point she can either play `Th` to end her turn or play `9s-8s-6s-4s-2s`. The latter is indeed the optimal choice, leaving her with only `Th` in hand. (We will not explore here what would happen if player one played `Th` instead, but it is in fact a worse play, resulting in a lower score for player one.)

Next it is player two's turn. Because the top card on the pile is `2s`, she can start with either `2h` or `As`. At this point her goal is to discard as many cards as possible in one turn and minimize her loss, because player one is guaranteed to win on the next turn. There are a few possible chains: `As-Js-7s-7d-8d-9d`, `As-Js-7s-7d-8d-8c`, `2h-6h-6d-7d-8d-9d`, `2h-6h-6d-7d-8d-8c`, and `2h-6h-6d-7d-7s-As-Js`. The last one is the best choice, leaving her with `8d 9d 8c` in hand.

In summary, the optimal series of plays results in the following game:
Player one: `Ad-Qd-Qs-Ts-9s-8s-6s-4s-2s`
Player two: `2h-6h-6d-7d-7s-As-Js`
Player one: `Th`
Player two is left with `8d 9d 8c`; player one wins 3 points.

Analysis of the second game:

In this example there are no legal chains for player one and she must start by discarding `As`. Player two can then play her entire hand and win by 2 points.

2015/2016 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 6
A Classy Problem

In his memoir *So, Anyway*, British comedian John Cleese writes of the class difference between his father (who was "middle-middle-middle-lower-middle class") and his mother (who was "upper-upper-lower-middle class"). These fine distinctions between classes tend to confuse non-British readers, so your team is to write a program to sort a group of people by their classes to show the true distinctions.

For this problem, there are three main classes: upper, middle, and lower. Obviously, the highest is upper and the lowest is lower. But there can be distinctions within a class, so upper-upper is a higher class than middle-upper, which is higher than lower-upper. However, all of the upper classes (upper-upper, middle-upper, and lower-upper) are higher than any of the middle classes. These distinctions can be carried further: the lower-upper class can itself be divided into upper-lower-upper, middle-lower-upper, and lower-lower-upper classes; and each of those classes can be similarly divided...

A class description with leading middle- modifiers is equivalent to a class description without them: upper class and middle-upper class are equivalent, as are middle-middle-lower-middle and lower-middle. However, lower-middle-upper class is higher than lower-upper class.

Input to your program will consist of information about up to 100 people, one person per input line. Each line contains the name of a person, followed by a colon, a single space, and the class of the person. The name contains up to thirty lower-case characters, possibly including spaces. The class is a string consisting of a non-empty sequence of up to ten of the words "upper", "middle", or "lower", separated from each other by hyphens. This string will be followed by a single space and the word "class". No two people will have the same name.

Your program is to print the list of names from highest to lowest class, one per line. No added leading or trailing whitespace is to appear on an output line. If two people have the same or equivalent classes, they are to be listed in alphabetical order by name.

Sample Input

```
mom: upper-upper-lower-middle class
dad: middle-middle-middle-lower-middle class
queen elizabeth: upper-upper-upper class
chair: lower-lower class
uncle bob: middle-middle-lower-middle class
```

Output for the Sample Input

```
queen elizabeth
mom
dad
uncle bob
chair
```

2015/2016 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 7
$2 \times 2 \times 2$

The $2 \times 2 \times 2$ Rubik's Cube is the easier version of the famous $3 \times 3 \times 3$ Rubik's Cube. The current world record for solving the $2 \times 2 \times 2$ cube by hand in competition is 0.58 seconds. You have a chance to set the world record for solving the cube by a program in a regional ICPC competition! Can your program beat a human champion?

The traditional coloring of the six sides of the cube uses the colors *White (up)*, *Green (front)*, *Red (right)*, *Orange (left)*, *Blue (back)*, and *Yellow (down)*. To represent the 24 squares on the surface of the cube we will use an ASCII notation like this:

```
  WW
  WW
OOGGRRBB
OOGGRRBB
  YY
  YY
```

In this notation, the four characters in the center of the cross represent the front of the cube, the four above them represent the top of the cube, etc.

To solve (or scramble) the cube we will use a series of moves, each consisting of rotation of one half of the cube by 90 degrees while holding the other half in place. We will use the following notation:

`R F U L D B`: rotate the right/front/up/left/down/back half by 90 degrees clockwise

`R' F' U' L' D' B'`: rotate the right/front/up/left/down/back half by 90 degrees counter-clockwise

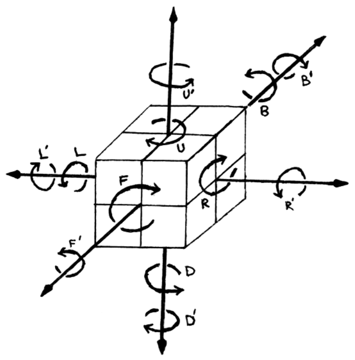


Figure 1. The 12 basic moves.

For example, applying the R' move on a solved cube would result in the following state:

```
  WB
  WB
OOGWRRBY
OOGWRRBY
  YG
  YG
```

Problem 7
$2 \times 2 \times 2$ (continued)

Your team's task is to write a program that solves scrambled cubes by reading their ASCII representations and writing out solutions as sequences of the 12 basic moves. Cubes will be separated from each other in the input by empty lines, and the last cube will be followed by end-of-file. Each solution is to be printed on a line by itself without leading or trailing whitespace. You will not be given a cube that is already solved.

There are many different solutions possible for each case, and any solution will be accepted regardless of its length. However, the solution must put the cube in its canonical solved state, that is with *White* at the top, *Green* at the front, etc. No illegal input will be given.

Sample Input

```
  BB
  GR
YWOWGWRR
OWOGYRYB
  BO
  YG

  BG
  BR
RRYGYOYW
RWGOYOBW
  OB
  GW
```

Output for the Sample Input
(Not the only possible solutions)

```
FUR
B'L'D'D'UR'FLFD'F'B
```

2015/2016 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 8
Persistence

Given a positive integer expressed in base ten, consider the series of numbers that will be produced where each successive term is the product of the decimal digits of the previous term. Eventually a term will have a single digit.

For example:

679	$6 \times 7 \times 9$	$\Rightarrow$	378
378	$3 \times 7 \times 8$	$\Rightarrow$	168
168	$1 \times 6 \times 8$	$\Rightarrow$	48
48	4×8	$\Rightarrow$	32
32	3×2	$\Rightarrow$	6

The number of steps this takes is called the *persistence*; thus the persistence of 679 is 5, the number of steps it takes to get to a single digit number. Your team is to write a program that will determine the persistence of positive integers.

Input to your program will be a series of lines terminated by end-of-file. Each line will consist of a positive number of up to nine decimal digits, starting in the first column with no leading zeroes.

For each input number, print a line containing the number exactly as input, followed by a single space and its persistence with no leading zeroes. No leading or trailing whitespace is to be printed on an output line.

Sample Input

```
5
10
25
39
679
6788
26888999
```

Output for the Sample Input

```
5 0
10 1
25 2
39 3
679 5
6788 6
26888999 9
```

2015/2016 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 9
Racing Gems

You are playing a racing game. Your character starts at the x axis line ($y = 0$) and proceeds up the racetrack, which has a boundary at the line $x = 0$ and $x = w$. The finish is at $y = h$, and the game ends when you reach that line. You proceed at a fixed vertical velocity v, but you can control your horizontal velocity to be any value between $-v/r$ and v/r, and change it at any time.

There are a set of gems at specific points on the race track. Your job is to collect as many gems as possible (they all have the same value).

How many gems can your program collect? You may start at any horizontal position you want (but your vertical position must be 0 at the start).

Input will be as follows. The first line will contain four integers, separated by spaces: n (the number of gems), r (the ratio of vertical velocity to maximum horizontal speed), w (the width of the track), and h (the height of the finish line). Following this will be n lines, each containing an integer x and y coordinate, containing the coordinate of a gem. All gems will lie on the race track. None will be on the start line.

Input value ranges are as follows: $1 \leq r \leq 10$, $1 \leq n \leq 10^5$, $1 \leq w \leq 10^9$, and $1 \leq h \leq 10^9$.

Your program is to print a line giving the maximum number of gems that can be collected. No leading or trailing whitespace is to appear on the output line.

Sample Input 1

```
5 1 10 10
8 8
5 1
4 6
4 7
7 9
```

Output for Sample Input 1

```
3
```

Sample Input 2

```
5 1 100 100
27 75
79 77
40 93
62 41
52 45
```

Output for Sample Input 2

```
3
```

Sample Input 3

```
10 3 30 30
14 9
2 20
3 23
15 19
13 5
17 24
6 16
21 5
14 10
3 6
```

Output for Sample Input 3

```
4
```

2015/2016 SOUTHERN CALIFORNIA REGIONAL
ACM INTERNATIONAL COLLEGIATE PROGRAMMING CONTEST

Problem 10
Megachess

Your team has been hired by a group of chess enthusiasts who are interested in trying out new versions of the game. Specifically, they are trying much larger square game boards (as large as 1000 squares on a side) and much larger piece counts (although there is still only one king per player).

With very large boards and the potential for thousands of pieces, it becomes fairly difficult to easily determine if a piece is in danger of capture. Your team is to write a program that will, given the positions of the pieces on the board, determine if a given piece is in danger of capture. You do not need to determine if the positions of the pieces are legal, only if the given piece would be in danger if the pieces were positioned as stated.

For those unfamiliar with the rules of chess, a description and diagrams of how the various pieces capture follows the problem description.

Input to your program will begin with a line containing two values: the number of squares on a side (s, in the range 3 to 1000 inclusive) and the number of pieces currently on the board. This will be followed by a list of pieces, one per line. Each piece will be represented by its color (B or W), its rank (K=King, Q=Queen, R=Rook, B=Bishop, N=Knight, P=Pawn), its row (in the range 1 to s, 1 being the bottom row), and its column (in the range 1 to s, 1 being the left column). Fields will be separated from each other by single spaces, and there will be no leading or trailing whitespace on an input line. The piece whose danger of capture is to be determined will be given first.

Your program is to analyze the board and determine if the piece is in danger of capture. If it is in danger, print the string "In Danger" on a line by itself; if not, print the string "Not in Danger" on a line by itself. No leading or trailing whitespace is to appear on the output line.

Sample Input 1

```
202 4
B K 143 74
W N 141 73
B Q 142 74
W K 1 102
```

Output for Sample Input 1

```
In Danger
```

Sample Input 2

```
725 5
B R 41 643
B K 63 621
W B 85 599
W K 37 645
B N 39 644
```

Output for Sample Input 2

```
Not in Danger
```

Sample Input 3

```
19 3
W K 12 7
B P 13 8
B K 14 8
```

Output for Sample Input 3

```
In Danger
```

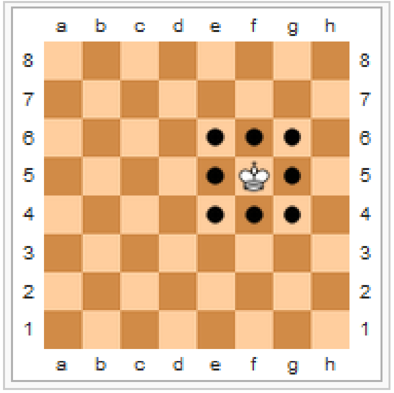
Movement

The player with the white pieces always moves first. After the first move, players alternately move one piece per turn (except for castling, when two pieces are moved). Pieces are moved to either an unoccupied square or one occupied by an opponent's piece, which is captured and removed from play. With the sole exception of *en passant*, all pieces capture by moving to the square that the opponent's piece occupies. A player may not make any move that would put or leave his or her king under attack. A player cannot "pass"; at each turn they have to make a legal move (this is the basis for the finesse called zugzwang). If the player to move has no legal move, the game is over; it is either a checkmate (a loss for the player with no legal moves) if the king is under attack, or a stalemate (a draw) if the king is not.

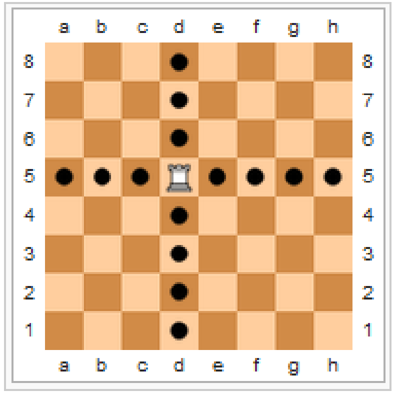
Each chess piece has its own style of moving. In the diagrams, the dots mark the squares where the piece can move if no other pieces (including one's own piece) are on the squares between the piece's initial position and its destination.

- The king moves one square in any direction. The king has also a special move which is called *castling* and involves also moving a rook.
- The rook can move any number of squares along any rank or file, but may not leap over other pieces. Along with the king, the rook is involved during the king's castling move.
- The bishop can move any number of squares diagonally, but may not leap over other pieces.
- The queen combines the power of the rook and bishop and can move any number of squares along rank, file, or diagonal, but it may not leap over other pieces.
- The knight moves to any of the closest squares that are not on the same rank, file, or diagonal, thus the move forms an "L"-shape: two squares vertically and one square horizontally, or two squares horizontally and one square vertically. The knight is the only piece that can leap over other pieces.
- The pawn may move forward to the unoccupied square immediately in front of it on the same file, or on its first move it may advance two squares along the same file provided both squares are unoccupied (black "•"s in the diagram); or the pawn may capture an opponent's piece on a square diagonally in front of it on an adjacent file, by moving to that square (black "x"s).
  The pawn has two special moves: the *en passant* capture and pawn promotion.

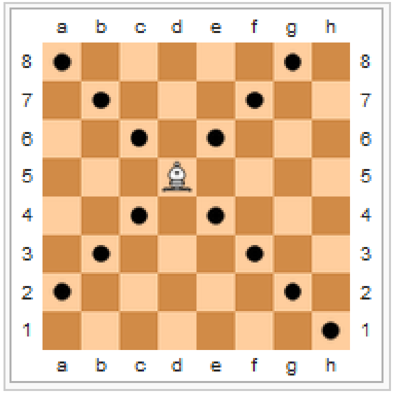
Moves of a king



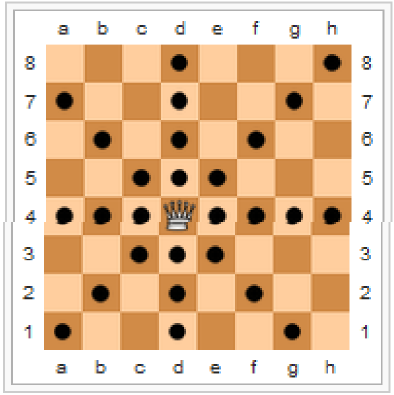
Moves of a rook



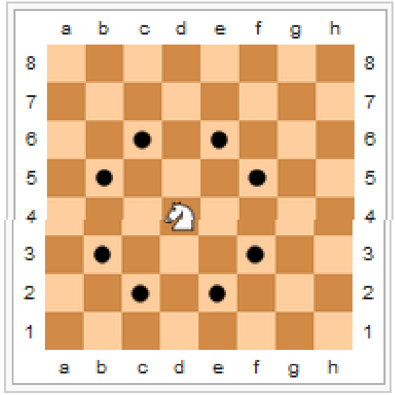
Moves of a bishop



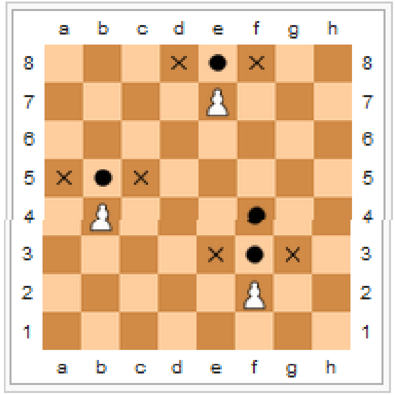
Moves of a queen



Moves of a knight



Moves of a pawn



Note that white pawns move to higher-numbered rows (ranks), while black pawns move to lower-numbered rows.