Problem Statement

Harry is playing magical chess. In this version of the game, all pieces move the same way as in regular chess, but players can cast some magical spells. Unfortunately, Harry's opponent, Joe, has captured all of Harry's pieces except one knight; Joe, on the other hand, still has a queen and a rook. The only chance Harry has to win this game is to cast a spell, "Haste", that will allow Harry's knight to make several moves in a row. You should determine the minimal number of moves the knight needs to capture both the rook and the queen, assuming neither of them moves. You may capture them in either order - rook first or queen first.

You will be given a String, knight, containing information about the knight. You will also be given a String, queen, and a String, rook, giving you information about Joe's pieces. knight, rook and queen will be formatted as "cr", where c is a character between 'a' and 'h', inclusive, determining the column on the board ('a' is the first column, 'h' is the last), and r is a digit between '1' and '8', inclusive, determining the row (1 is the lowest, 8 is the highest).

Definition

Class: CaptureThemAll

Method: fastKnight

Parameters: String, String, String

Returns: int

Method signature: int fastKnight(String knight, String rook, String queen)

(be sure your method is public)

Notes

- A knight's jump moves him 2 cells along one of the axes, and 1 cell along the other one. In other words, if knight is in the (0,0) now, it can be in (-2, -1), (-2, 1), (2, -1), (2, 1), (-1, -2), (1, -2), (-1, 2) or (1, 2) after his move.

- The knight will capture one of Joe's pieces if it ends its move in the cell that the piece occupies.

- The knight cannot jump out of the board.

- A chessboard has 8 rows and 8 columns.

Constraints

- knight, rook and queen will all be distinct.

- knight, rook and queen will be formatted as "cr", where c is a lowercase character between 'a' and 'h', inclusive, and r is a digit between '1' and '8', inclusive.

Examples

0)

"a1"

"b3"

"c5"

Returns: 2

From "a1", the knight can move directly to "b3" and capture the rook. From there, the knight can move directly to "c5" and capture the queen.

1)

"b1"

"c3"

"a3"

Returns: 3

The rook and the queen are both 1 move away from the knight. Once the knight captures one of them (it doesn't matter which one), it can return to its starting location, and capture the other piece in one more move.

2)

"a1"

"a2"

"b2"

Returns: 6

The rook and the queen are close, but it takes 6 moves to capture them.

3)

"a5"

"b7"

"e4"

Returns: 3

4)

"h8"

"e2"

"d2"

Returns: 6

Problem Statement

Note: This problem statement includes images that may not appear if you are using a plugin. For best results, use the Arena editor.

In one mode of the grafix software package, the user blocks off portions of a masking layer using opaque rectangles. The bitmap used as the masking layer is 400 pixels tall and 600 pixels wide. Once the rectangles have been blocked off, the user can perform painting actions through the remaining areas of the masking layer, known as holes. To be precise, each hole is a maximal collection of contiguous pixels that are not covered by any of the opaque rectangles. Two pixels are contiguous if they share an edge, and contiguity is transitive.

You are given a String[] named rectangles, the elements of which specify the rectangles that have been blocked off in the masking layer. Each String in rectangles consists of four integers separated by single spaces, with no additional spaces in the string. The first two integers are the window coordinates of the top left pixel in the given rectangle, and the last two integers are the window coordinates of its bottom right pixel. The window coordinates of a pixel are a pair of integers specifying the row number and column number of the pixel, in that order. Rows are numbered from top to bottom, starting with 0 and ending with 399. Columns are numbered from left to right, starting with 0 and ending with 599. Every pixel within and along the border of the rectangle defined by these opposing corners is blocked off.

Return a int[] containing the area, in pixels, of every hole in the resulting masking area, sorted from smallest area to greatest.

Definition

Class: grafixMask

Method: sortedAreas

Parameters: String[]

Returns: int[]

Method signature: int[] sortedAreas(String[] rectangles)

(be sure your method is public)

Notes

- Window coordinates are not the same as Cartesian coordinates. Follow the definition given in the second paragraph of the problem statement.

Constraints

- rectangles contains between 1 and 50 elements, inclusive

- each element of rectangles has the form "ROW COL ROW COL", where: "ROW" is a placeholder for a non-zero-padded integer between 0 and 399, inclusive; "COL" is a placeholder for a non-zero-padded integer between 0 and 599, inclusive; the first row number is no greater than the second row number; the first column number is no greater than the second column number

Examples

0)

{"0 292 399 307"}

Returns: { 116800, 116800 }



The masking layer is depicted below in a 1:4 scale diagram.

1)

{"48 192 351 207", "48 392 351 407", "120 52 135 547", "260 52 275 547"}

Returns: { 22816, 192608 }



2)

{"0 192 399 207", "0 392 399 407", "120 0 135 599", "260 0 275 599"}

Returns: { 22080, 22816, 22816, 23040, 23040, 23808, 23808, 23808, 23808 }



3)

{"50 300 199 300", "201 300 350 300", "200 50 200 299", "200 301 200 550"}

Returns: { 1, 239199 }

4)

{"0 20 399 20", "0 44 399 44", "0 68 399 68", "0 92 399 92",

"0 116 399 116", "0 140 399 140", "0 164 399 164", "0 188 399 188",

"0 212 399 212", "0 236 399 236", "0 260 399 260", "0 284 399 284",

"0 308 399 308", "0 332 399 332", "0 356 399 356", "0 380 399 380",

"0 404 399 404", "0 428 399 428", "0 452 399 452", "0 476 399 476",

"0 500 399 500", "0 524 399 524", "0 548 399 548", "0 572 399 572",

"0 596 399 596", "5 0 5 599", "21 0 21 599", "37 0 37 599",

"53 0 53 599", "69 0 69 599", "85 0 85 599", "101 0 101 599",

"117 0 117 599", "133 0 133 599", "149 0 149 599", "165 0 165 599",

"181 0 181 599", "197 0 197 599", "213 0 213 599", "229 0 229 599",

"245 0 245 599", "261 0 261 599", "277 0 277 599", "293 0 293 599",

"309 0 309 599", "325 0 325 599", "341 0 341 599", "357 0 357 599",

"373 0 373 599", "389 0 389 599"}

Returns:

{ 15, 30, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 45, 100, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 115, 200, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 230, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 300, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345, 345 }

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

This problem statement is the exclusive and proprietary property of TopCoder, Inc. Any unauthorized use or reproduction of this information without the prior written consent of TopCoder, Inc. is strictly prohibited. (c)2010, TopCoder, Inc. All rights reserved.