## Problem D. asteroids

This contest is open for practice. You can try every problem as many times as you like, though we won't keep track of which problems you solve. Read the Quick-Start Guide to get started.

| | |
|---|---|
| small<br>5 points<br>*2 minute timeout* | The contest is finished. |
| large<br>25 points<br>*10 minute timeout* | The contest is finished. |

Problem

# Asteroids

You are playing a game of DCJAsteroids in your old arcade machine. The game is played on a screen that displays a matrix of square cells. The matrix has a fixed width and infinite height. Each cell may contain your spaceship, an asteroid, or some amount of helium. The goal of the game is to collect helium without colliding with any asteroids. Your spaceship always stays in the bottom row of the screen, and it can only move horizontally.

The game is turn-based. On a turn, you can remain where you are, move to the cell on your left (unless you are in the left-most column), or move to the cell on your right (unless you are in the right-most column). After you move (or not) your spaceship, every other item in the screen moves to the cell directly below it. The cells in the bottom row fall off the bottom of the screen and disappear.

For instance, suppose this is a current arrangement, with X representing your spaceship, # representing a cell occupied by an asteroid, and . representing a cell not occupied by an asteroid (to keep it simple, we do not depict the amount of helium in this example).

```
#.##.
#..#.
..X..
```

If you decide to stay at your current position, then the resulting state after the turn is resolved will be:

```
#.##.
#.X#.
```

If you decide to move left, then the result after the turn is resolved will be:

```
#.##.
#X.#.
```

If you ever move the spaceship into a cell occupied by an asteroid or an asteroid moves into the cell occupied by your spaceship, the spaceship is destroyed. In the example above, if you decided to move right, after the turn is resolved, your spaceship and an asteroid would occupy the same cell, so the spaceship would be destroyed. If you decided to move left two turns in a row, the second move would make your spaceship move into an asteroid, so it would be destroyed as well.

Each cell not occupied by an asteroid may have a different amount of helium. We assign a digit 0 through 9 to each non-asteroid cell (instead of .); the number in a cell represents the number of points you earn from the helium in that cell. Each time your spaceship moves into a cell containing helium or a cell containing helium moves into your spaceship's position, you collect the helium there and are awarded the points. You can only collect the helium in each cell once; after that, your spaceship can still be in that cell, but you collect no additional helium and receive no additional points for it.

You will be given a matrix of characters representing the asteroids and helium that are approaching. You may assume that all rows below and above the input matrix are full of worthless helium (i.e., they are rows of 0s). You may choose the position you are in when the bottom row in the input matrix falls into your current row.

If there is no way to navigate through the entire input matrix without the spaceship being destroyed, output -1. Otherwise, output the maximum number of points you can accumulate while navigating through the entire input matrix. You are considered done navigating only when every row in the input matrix has already disappeared from the screen.

Let us add point values to the empty cells in the example above (assume your current position has 0 value). If you start by going left and then stay for two turns, you would go through these states and emerge without being destroyed:

```
#1##9    00000    00000     00000
#23#9 -> #1##9 -> 00000  -> 00000
66X15    #X3#9    #X##9     0X000
```

In this case you collect 6+2 points in the first turn, 1 in the second turn and 0 in the third, yielding a total of 9 points. If instead you stay for the first turn, then move left, and then stay again, this would happen:

```
#1##9    00000    00000     00000
#23#9 -> #1##9 -> 00000  -> 00000
66X15    #2X#9    #X##9     0X000
```

In this case you collect 3 points in the first turn, 2+1 points in the second and 0 in the third, for a total of 6 points. Any other sequence of moves gets the spaceship destroyed.

### Input

The input library is called "asteroids"; see the sample inputs below for examples in your language. It defines three methods:

- **GetHeight()**:
  - Takes no argument.
  - Returns a 64-bit integer: the height of the input matrix.
  - Expect each call to take 0.05 microseconds.
- **GetWidth()**:
  - Takes no argument.
  - Returns a 64-bit integer: the width of both the screen and the input matrix.
  - Expect each call to take 0.05 microseconds.
- **GetPosition(i, j)**:
  - Takes two 64-bit integers in the ranges $0 \le i <$ GetHeight(), $0 \le j <$ GetWidth().
  - Returns a character: the contents of the cell at row i and column j of the input matrix. The character is # if the cell contains an asteroid and a digit if not, representing the point value of the helium you can collect from that cell, as explained above. Rows are numbered from bottom to top (so your spaceship will enter the rows in increasing order). Columns are numbered from left to right, as explained above.
  - Expect each call to take 0.05 microseconds.

### Output

Output a single line with a single integer: the maximum number of points you can accumulate in your journey through the asteroids without being destroyed, or -1 if it is impossible to avoid destruction.

### Limits

Time limit: 5 seconds.
Memory limit per node: 128 MB.
Maximum number of messages a single node can send: 1000.
Maximum total size of messages a single node can send: 8 MB.
GetPosition(i, j) is a digit or #, for all i and j.

### Small dataset

Number of nodes: 10.
$1 \le$ GetHeight() $\le 1000$.
$1 \le$ GetWidth() $\le 1000$.

### Large dataset

Number of nodes: 100.
$1 \le$ GetHeight() $\le 30{,}000$.
$1 \le$ GetWidth() $\le 30{,}000$.

### Sample

| Input | Output |
| --- | --- |
| See input files below. | For sample input 1:<br>10<br>For sample input 2:<br>22<br>For sample input 3:<br>-1 |

For ease of reading, these are the input matrices in the samples:

```
8##123
#999#1
21##11
52#11#

1#78
0011
#2#9
0136
0#8#
21#9

0##
000
##0
```

Sample input libraries:
Sample input for test 1: asteroids.h [CPP] asteroids.java [Java]
Sample input for test 2: asteroids.h [CPP] asteroids.java [Java]
Sample input for test 3: asteroids.h [CPP] asteroids.java [Java]