Examination 1

*You can choose Java or C++

– "Orienteering" –

Abstract

We are planning an orienteering game.
The aim of this game is to arrive at the goal (G) from the start (S) with the shortest distance.
However, the players have to pass all the checkpoints (@) on the map.

An orienteering map is to be given in the following format.

```
########
#@....G#
##.##@##
#..@..S#
#@.....#
########
```

In this problem, an orienteering map is to be given.
Calculate the minimum distance from the start to the goal with passing all the checkpoints.

Specification

* A map consists of 5 characters as following.
You can assume that the map does not contain any invalid characters and
the map has exactly one start symbol 'S' and exactly one goal symbol 'G'.

- 'S' means the orienteering start.
- 'G' means the orienteering goal.
- '@' means an orienteering checkpoint.
- '.' means an opened-block that players can pass.
- '#' means a closed-block that players cannot pass.

* It is allowed to move only by one step vertically or horizontally (up, down, left, or right) to the next block.
Other types of movements, such as moving diagonally (left up, right up, left down and right down) and skipping one or more blocks, are NOT permitted.
* You MUST NOT get out of the map.
* Distance is to be defined as the number of movements to the different blocks.
* You CAN pass opened-blocks, checkpoints, the start, and the goal more than once if necessary.
* You can assume that parameters satisfy following conditions.

- 1 <= width <= 100
- 1 <= height <= 100
- The maximum number of checkpoints is 18.

* Return -1 if given arguments do not satisfy specifications, or players cannot arrive at the goal from the start by passing all the checkpoints.

Input

The input is to be given in the following format, from the standard input.

```
W H
Row1
Row2
...
RowH
```

The first row is to describe the width and the height of the orienteering map, sectioned by a space.
From the second row, map information is to be given for exactly the same number of rows as the height.
Each row contains exactly the same number of letters as the width.
See “Specification” for details of the symbols contained in the map.

Output

Output into the standard output, and put a return.

<u>DO NOT forget to delete debug outputs before submission** .</u>**

Implementation

<Java>

Write the "jp.co.worksap.global.Orienteering" class, which has executable "main" method.

If other fields, methods, and/or classes are necessary, you can add them to the skeleton, or implement them on other files.

<Skeleton Code of Java>

```
package jp.co.worksap.global;

public class Orienteering {
    public static void main(String[] args) {
        // TODO: Implement your program
    }
}
```

<C++>

Write the "main" function of the "Orienteering" class, included in given skeleton.

If some C++ standard libraries and members are necessary, add them to the skeleton.
You can implement them on other files, but describe them not to cause compile errors with given compile option.

<Skeleton Code of C++>

```
class Orienteering {
public:
    void main();
};

void Orienteering::main() {
  // TODO: Implement this function
}

int main(int argc, char* argv[]) {
  Orienteering o;
  o.main();
  return 0;
}
```

<Compile Options>

The codes you have submitted are to be compiled by the following command.

```
g++ -std=c++11 -O2 -o a.out orienteering.cpp
```

Evaluation Criteria

This problem is to be evaluated by following criteria.

*** Being able to solve the problem correctly.**
*** Being able to get the answer fast.**

Examples

<Notes>

In these examples, the origin of coordinates is left-upper corner, and is described as (0, 0).
The direction of positive X-Axis is to right and positive Y-Axis is to down.

Example1

<Input>

```
5 4
#####
#...#
#S#G#
#####
```

<Output>

```
4
```

<Java Test Example>

```
# current directory has your build.
java -cp . jp.co.worksap.global.Orienteering < example1.txt
```

<C++ Test Example>

```
# current directory has your source code.
g++ -std=c++11 -O2 -o a.out orienteering.cpp
./a.out < example1.txt
```

Result path (x, y): (1, 2) => (1, 1) => (2, 1) => (3, 1) => (3, 2).
The result distance is 4, because the player moves 4 times.

Example2

<Input>

```
5 5
#####
#.@@#
#S###
#..G#
#####
```

<Output>

```
9
```

Result path (x, y): (1, 2) => (1, 1) => (2, 1) => (3, 1) => (2, 1) => (1, 1) => (1, 2) => (1, 3) => (2, 3) => (3, 3).

The result distance is 9, because the player moves 9 times.
Start (1, 2), opened-block (1, 1), and checkpoint (2, 1) appear twice each on the result path, but it is correct because the problem specifications allow to pass the same block for multiple times.

Example3

<Input>

```
5 5
#####
#S..#
##G##
#..@#
#####
```

<Output>

```
6
```

Result path (x, y): (1, 1) => (2, 1) => (2, 2) => (2, 3) => (3, 3) => (2, 3) => (2, 2).

The result distance is 6, because the player moves 6 times.
Opened-block (2, 3) and goal (2, 2) appear twice each on the result path, but it is correct because of the problem specifications.