

# Problem 1349: Maximum Students Taking Exam

## Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

## Problem Description

Given a

$m \times n$

matrix

seats

that represent seats distributions in a classroom. If a seat is broken, it is denoted by

'#'

character otherwise it is denoted by a

'.'

character.

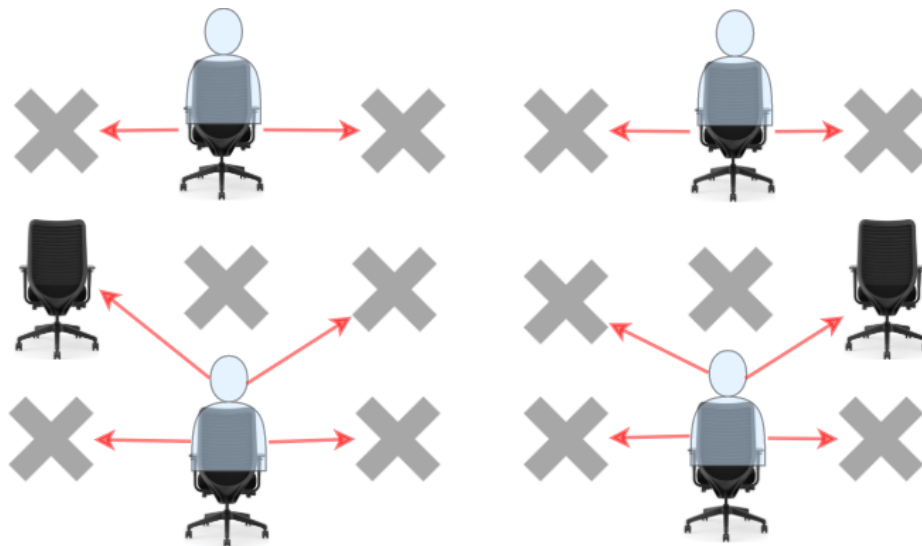
Students can see the answers of those sitting next to the left, right, upper left and upper right, but he cannot see the answers of the student sitting directly in front or behind him. Return the

maximum

number of students that can take the exam together without any cheating being possible.

Students must be placed in seats in good condition.

Example 1:



Input:

```
seats = [["#",".","#","#",".","#"], [".","#","#","#","#","."], ["#",".","#","#",".","#"]]
```

Output:

4

Explanation:

Teacher can place 4 students in available seats so they don't cheat on the exam.

Example 2:

Input:

```
seats = [".","#"], ["#","#"], ["#","."], ["#","#"], [".","#"]]
```

Output:

3

Explanation:

Place all students in available seats.

Example 3:

Input:

```
seats = [{"#", ".", "
```

```
."
```

```
", ".", "#"], ["
```

```
."
```

```
", "#", "
```

```
."
```

```
", "#", "
```

```
."
```

```
"], ["
```

```
."
```

```
"", ".", "#", ".", "
```

```
."
```

```
"], ["
```

```
."
```

```
", "#", "
```

```
."
```

```
", "#", "
```

.

"], [{"#", ".", "

.

", ".", "#"]]

Output:

10

Explanation:

Place students in available seats in column 1, 3 and 5.

Constraints:

seats

contains only characters

'.'

and

'#'.

m == seats.length

n == seats[i].length

1 <= m <= 8

1 <= n <= 8

## Code Snippets

### C++:

```
class Solution {
public:
    int maxStudents(vector<vector<char>>& seats) {

    }
};
```

### Java:

```
class Solution {
    public int maxStudents(char[][] seats) {

    }
}
```

### Python3:

```
class Solution:
    def maxStudents(self, seats: List[List[str]]) -> int:
```

### Python:

```
class Solution(object):
    def maxStudents(self, seats):
        """
        :type seats: List[List[str]]
        :rtype: int
        """
```

### JavaScript:

```
/**
 * @param {character[][]} seats
 * @return {number}
 */
var maxStudents = function(seats) {

};
```

### TypeScript:

```
function maxStudents(seats: string[][]): number {  
  
};
```

### C#:

```
public class Solution {  
    public int MaxStudents(char[][] seats) {  
  
    }  
}
```

### C:

```
int maxStudents(char** seats, int seatsSize, int* seatsColSize) {  
  
}
```

### Go:

```
func maxStudents(seats [][]byte) int {  
  
}
```

### Kotlin:

```
class Solution {  
    fun maxStudents(seats: Array<CharArray>): Int {  
  
    }  
}
```

### Swift:

```
class Solution {  
    func maxStudents(_ seats: [[Character]]) -> Int {  
  
    }  
}
```

### Rust:

```

impl Solution {
  pub fn max_students(seats: Vec<Vec<char>>) -> i32 {

  }
}

```

## Ruby:

```

# @param {Character[][]} seats
# @return {Integer}
def max_students(seats)

end

```

## PHP:

```

class Solution {

  /**
   * @param String[][] $seats
   * @return Integer
   */
  function maxStudents($seats) {

  }
}

```

## Dart:

```

class Solution {
  int maxStudents(List<List<String>> seats) {

  }
}

```

## Scala:

```

object Solution {
  def maxStudents(seats: Array[Array[Char]]): Int = {

  }
}

```

### Elixir:

```
defmodule Solution do
  @spec max_students(seats :: [[char]]) :: integer
  def max_students(seats) do

  end

end
```

### Erlang:

```
-spec max_students(Seats :: [[char()]]) -> integer().
max_students(Seats) ->
.
```

### Racket:

```
(define/contract (max-students seats)
  (-> (listof (listof char?)) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
 * Problem: Maximum Students Taking Exam
 * Difficulty: Hard
 * Tags: array, dp
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

class Solution {
public:
    int maxStudents(vector<vector<char>>& seats) {

    }

};
```

### Java Solution:

```
/**
 * Problem: Maximum Students Taking Exam
 * Difficulty: Hard
 * Tags: array, dp
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

class Solution {
public int maxStudents(char[][] seats) {

}

}
```

### Python3 Solution:

```
"""
Problem: Maximum Students Taking Exam
Difficulty: Hard
Tags: array, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxStudents(self, seats: List[List[str]]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def maxStudents(self, seats):
"""
:type seats: List[List[str]]
:rtype: int
```

```
"""
```

### JavaScript Solution:

```
/**
 * Problem: Maximum Students Taking Exam
 * Difficulty: Hard
 * Tags: array, dp
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

/**
 * @param {character[][]} seats
 * @return {number}
 */
var maxStudents = function(seats) {

};
```

### TypeScript Solution:

```
/**
 * Problem: Maximum Students Taking Exam
 * Difficulty: Hard
 * Tags: array, dp
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

function maxStudents(seats: string[][]): number {

};
```

### C# Solution:

```

/*
 * Problem: Maximum Students Taking Exam
 * Difficulty: Hard
 * Tags: array, dp
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

public class Solution {
    public int MaxStudents(char[][] seats) {

    }
}

```

### C Solution:

```

/*
 * Problem: Maximum Students Taking Exam
 * Difficulty: Hard
 * Tags: array, dp
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) or O(n * m) for DP table
 */

int maxStudents(char** seats, int seatsSize, int* seatsColSize) {

}

```

### Go Solution:

```

// Problem: Maximum Students Taking Exam
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

```

```

func maxStudents(seats [][]byte) int {

}

```

### Kotlin Solution:

```

class Solution {
    fun maxStudents(seats: Array<CharArray>): Int {

    }
}

```

### Swift Solution:

```

class Solution {
    func maxStudents(_ seats: [[Character]]) -> Int {

    }
}

```

### Rust Solution:

```

// Problem: Maximum Students Taking Exam
// Difficulty: Hard
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
    pub fn max_students(seats: Vec<Vec<char>>) -> i32 {

    }
}

```

### Ruby Solution:

```

# @param {Character[][]} seats
# @return {Integer}
def max_students(seats)

```

```
end
```

### PHP Solution:

```
class Solution {  
  
    /**  
     * @param String[][] $seats  
     * @return Integer  
     */  
    function maxStudents($seats) {  
  
    }  
}
```

### Dart Solution:

```
class Solution {  
    int maxStudents(List<List<String>> seats) {  
  
    }  
}
```

### Scala Solution:

```
object Solution {  
    def maxStudents(seats: Array[Array[Char]]): Int = {  
  
    }  
}
```

### Elixir Solution:

```
defmodule Solution do  
    @spec max_students(seats :: [[char]]) :: integer  
    def max_students(seats) do  
  
    end  
end
```

### Erlang Solution:

```
-spec max_students(Seats :: [[char()]]) -> integer().  
max_students(Seats) ->  
.
```

### Racket Solution:

```
(define/contract (max-students seats)  
  (-> (listof (listof char?)) exact-integer?)  
)
```