Problem 1840: Maximum Building Height

Problem Information

Difficulty: Hard
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You want to build

n

new buildings in a city. The new buildings will be built in a line and are labeled from

1

to

n

.

However, there are city restrictions on the heights of the new buildings:

The height of each building must be a non-negative integer.

The height of the first building

must

be

0

.

The height difference between any two adjacent buildings

cannot exceed

1

.

Additionally, there are city restrictions on the maximum height of specific buildings. These restrictions are given as a 2D integer array

restrictions

where

restrictions[i] = [id

i

, maxHeight

i

]

indicates that building

id

i

must have a height

less than or equal to

maxHeight

i

.

It is guaranteed that each building will appear

at most once

in

restrictions

, and building

1

will

not

be in

restrictions

.

Return

the

maximum possible height

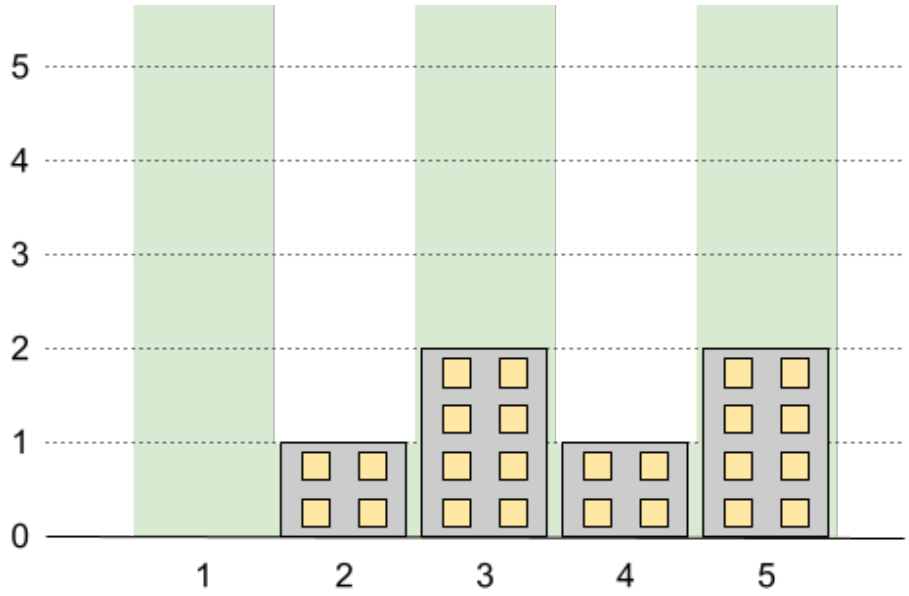
of the

tallest

building

.

Example 1:



Input:

n = 5, restrictions = [[2,1],[4,1]]

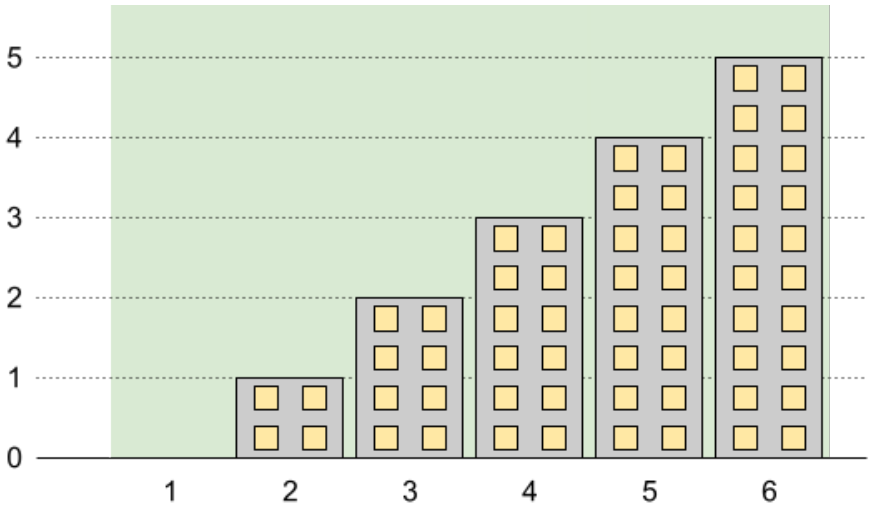
Output:

2

Explanation:

The green area in the image indicates the maximum allowed height for each building. We can build the buildings with heights [0,1,2,1,2], and the tallest building has a height of 2.

Example 2:



Input:

n = 6, restrictions = []

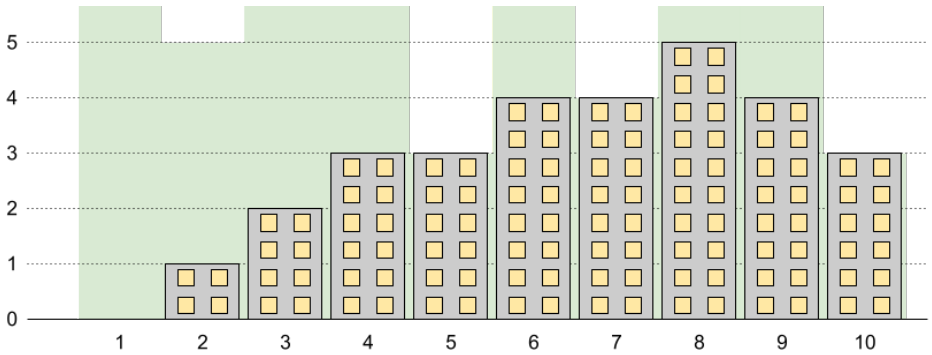
Output:

5

Explanation:

The green area in the image indicates the maximum allowed height for each building. We can build the buildings with heights [0,1,2,3,4,5], and the tallest building has a height of 5.

Example 3:



Input:

n = 10, restrictions = [[5,3],[2,5],[7,4],[10,3]]

Output:

5

Explanation:

The green area in the image indicates the maximum allowed height for each building. We can build the buildings with heights [0,1,2,3,3,4,4,5,4,3], and the tallest building has a height of 5.

Constraints:

2 <= n <= 10

9

0 <= restrictions.length <= min(n - 1, 10

5

)

2 <= id

i

<= n

id

i

is

unique

.

0 <= maxHeight

i

<= 10

9

Code Snippets

C++:

```
class Solution {
public:
int maxBuilding(int n, vector<vector<int>>& restrictions) {

}
};
```

Java:

```
class Solution {
public int maxBuilding(int n, int[][] restrictions) {

}
}
```

Python3:

```
class Solution:
def maxBuilding(self, n: int, restrictions: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def maxBuilding(self, n, restrictions):
"""
:type n: int
:type restrictions: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number[][]} restrictions
* @return {number}
*/
var maxBuilding = function(n, restrictions) {

};
```

TypeScript:

```
function maxBuilding(n: number, restrictions: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaxBuilding(int n, int[][] restrictions) {

}
}
```

C:

```
int maxBuilding(int n, int** restrictions, int restrictionsSize, int*
restrictionsColSize) {

}
```

Go:

```
func maxBuilding(n int, restrictions [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxBuilding(n: Int, restrictions: Array<IntArray>): Int {
```

```
}
}
```

Swift:

```
class Solution {
func maxBuilding(_ n: Int, _ restrictions: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_building(n: i32, restrictions: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer[][]} restrictions
# @return {Integer}
def max_building(n, restrictions)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $restrictions
* @return Integer
*/
function maxBuilding($n, $restrictions) {

}
}
```

Dart:

```
class Solution {
int maxBuilding(int n, List<List<int>> restrictions) {

}
}
```

Scala:

```
object Solution {
def maxBuilding(n: Int, restrictions: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_building(n :: integer, restrictions :: [[integer]]) :: integer
def max_building(n, restrictions) do

end
end
```

Erlang:

```
-spec max_building(N :: integer(), Restrictions :: [[integer()]]) ->
integer().
max_building(N, Restrictions) ->
.
```

Racket:

```
(define/contract (max-building n restrictions)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Building Height
* Difficulty: Hard
* Tags: array, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int maxBuilding(int n, vector<vector<int>>& restrictions) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Building Height
* Difficulty: Hard
* Tags: array, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int maxBuilding(int n, int[][] restrictions) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Building Height
Difficulty: Hard
Tags: array, math, sort
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maxBuilding(self, n: int, restrictions: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxBuilding(self, n, restrictions):
"""
:type n: int
:type restrictions: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Building Height
* Difficulty: Hard
* Tags: array, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} n
* @param {number[][]} restrictions
* @return {number}
*/
var maxBuilding = function(n, restrictions) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Building Height
* Difficulty: Hard
* Tags: array, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function maxBuilding(n: number, restrictions: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Building Height
* Difficulty: Hard
* Tags: array, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MaxBuilding(int n, int[][] restrictions) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Building Height
* Difficulty: Hard
* Tags: array, math, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
```

```
* Space Complexity: O(1) to O(n) depending on approach
*/

int maxBuilding(int n, int** restrictions, int restrictionsSize, int*
restrictionsColSize) {

}
```

Go Solution:

```
// Problem: Maximum Building Height
// Difficulty: Hard
// Tags: array, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maxBuilding(n int, restrictions [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxBuilding(n: Int, restrictions: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxBuilding(_ n: Int, _ restrictions: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Building Height
// Difficulty: Hard
// Tags: array, math, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn max_building(n: i32, restrictions: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer[][]} restrictions
# @return {Integer}
def max_building(n, restrictions)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer[][] $restrictions
* @return Integer
*/
function maxBuilding($n, $restrictions) {

}
}
```

Dart Solution:

```
class Solution {
int maxBuilding(int n, List<List<int>> restrictions) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def maxBuilding(n: Int, restrictions: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_building(n :: integer, restrictions :: [[integer]]) :: integer
def max_building(n, restrictions) do

end
end
```

Erlang Solution:

```
-spec max_building(N :: integer(), Restrictions :: [[integer()]]) ->
integer().
max_building(N, Restrictions) ->
.
```

Racket Solution:

```
(define/contract (max-building n restrictions)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```