Problem 3450: Maximum Students on a Single Bench

Problem Information

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

Problem Description

You are given a 2D integer array of student data

students

, where

students[i] = [student_id, bench_id]

represents that student

student_id

is sitting on the bench

bench_id

.

Return the

maximum

number of

unique

students sitting on any single bench. If no students are present, return 0.

Note

: A student can appear multiple times on the same bench in the input, but they should be counted only once per bench.

Example 1:

Input:

students = [[1,2],[2,2],[3,3],[1,3],[2,3]]

Output:

3

Explanation:

Bench 2 has two unique students:

[1, 2]

.

Bench 3 has three unique students:

[1, 2, 3]

.

The maximum number of unique students on a single bench is 3.

Example 2:

Input:

students = [[1,1],[2,1],[3,1],[4,2],[5,2]] Output:

3

Explanation:

Bench 1 has three unique students:

[1, 2, 3]

.

Bench 2 has two unique students:

[4, 5]

.

The maximum number of unique students on a single bench is 3.

Example 3:

Input:

students = [[1,1],[1,1]]

Output:

1

Explanation:

The maximum number of unique students on a single bench is 1.

Example 4:

Input:

students = []

Output:

0

Explanation:

Since no students are present, the output is 0.

Constraints:

0 <= students.length <= 100

students[i] = [student_id, bench_id]

1 <= student_id <= 100

1 <= bench_id <= 100

Code Snippets

C++:

```
class Solution {
public:
int maxStudentsOnBench(vector<vector<int>>& students) {

}
};
```

Java:

```
class Solution {
public int maxStudentsOnBench(int[][] students) {

}
}
```

Python3:

```
class Solution:
def maxStudentsOnBench(self, students: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def maxStudentsOnBench(self, students):
"""
:type students: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} students
* @return {number}
*/
var maxStudentsOnBench = function(students) {

};
```

TypeScript:

```
function maxStudentsOnBench(students: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaxStudentsOnBench(int[][] students) {

}
}
```

C:

```
int maxStudentsOnBench(int** students, int studentsSize, int*
studentsColSize) {
```

```
}
```

Go:

```
func maxStudentsOnBench(students [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxStudentsOnBench(students: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maxStudentsOnBench(_ students: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_students_on_bench(students: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} students
# @return {Integer}
def max_students_on_bench(students)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $students
* @return Integer
*/
function maxStudentsOnBench($students) {

}
}
```

Dart:

```
class Solution {
int maxStudentsOnBench(List<List<int>> students) {

}
}
```

Scala:

```
object Solution {
def maxStudentsOnBench(students: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_students_on_bench(students :: [[integer]]) :: integer
def max_students_on_bench(students) do

end
end
```

Erlang:

```
-spec max_students_on_bench(Students :: [[integer()]]) -> integer().
max_students_on_bench(Students) ->
.
```

Racket:

```
(define/contract (max-students-on-bench students)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Students on a Single Bench
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int maxStudentsOnBench(vector<vector<int>>& students) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Students on a Single Bench
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int maxStudentsOnBench(int[][] students) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Maximum Students on a Single Bench
Difficulty: Easy
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

```
class Solution(object):
def maxStudentsOnBench(self, students):
"""
:type students: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Students on a Single Bench
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
```

```
* @param {number[][]} students
* @return {number}
*/
var maxStudentsOnBench = function(students) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Students on a Single Bench
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function maxStudentsOnBench(students: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Students on a Single Bench
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int MaxStudentsOnBench(int[][] students) {

}
}
```

C Solution:

```c
/*
* Problem: Maximum Students on a Single Bench
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int maxStudentsOnBench(int** students, int studentsSize, int*
studentsColSize) {

}
```

Go Solution:

```go
// Problem: Maximum Students on a Single Bench
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func maxStudentsOnBench(students [][]int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun maxStudentsOnBench(students: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxStudentsOnBench(_ students: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Students on a Single Bench
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn max_students_on_bench(students: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} students
# @return {Integer}
def max_students_on_bench(students)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $students
* @return Integer
*/
function maxStudentsOnBench($students) {

}
}
```

Dart Solution:

```
class Solution {
int maxStudentsOnBench(List<List<int>> students) {

}
}
```

Scala Solution:

```
object Solution {
def maxStudentsOnBench(students: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_students_on_bench(students :: [[integer]]) :: integer
def max_students_on_bench(students) do

end
end
```

Erlang Solution:

```
-spec max_students_on_bench(Students :: [[integer()]]) -> integer().
max_students_on_bench(Students) ->
.
```

Racket Solution:

```
(define/contract (max-students-on-bench students)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```