Problem 1232: Check If It Is a Straight Line

Problem Information

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

Problem Description

You are given an integer array

coordinates

,

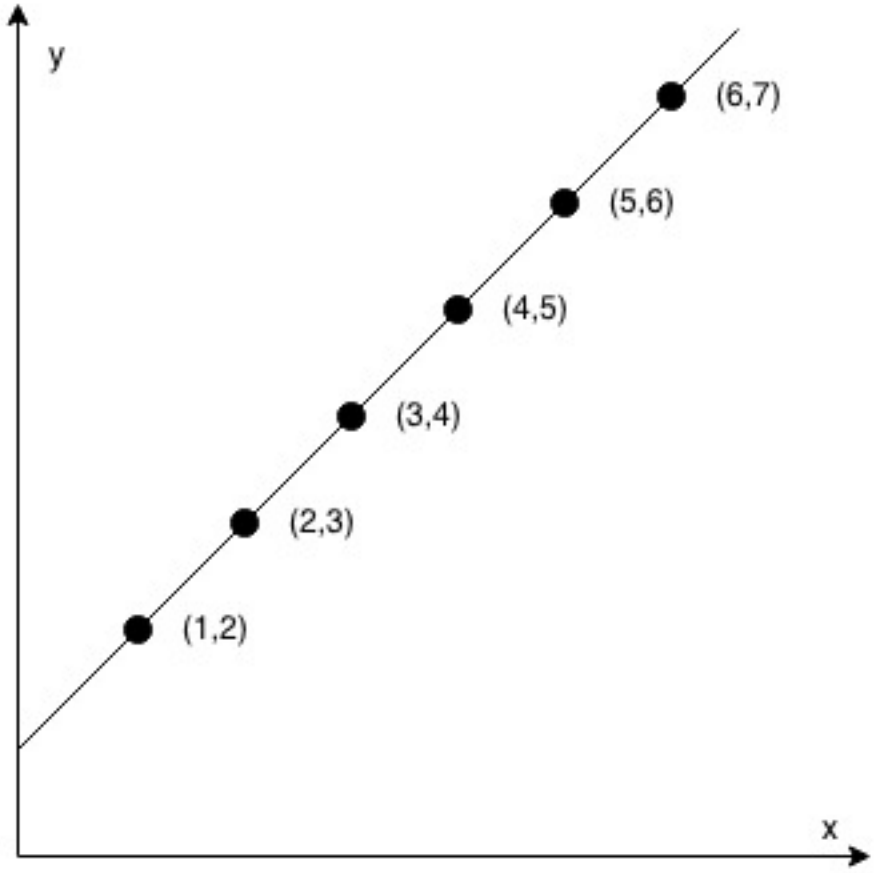
coordinates[i] = [x, y]

, where

[x, y]

represents the coordinate of a point. Check if these points make a straight line in the XY plane.

Example 1:



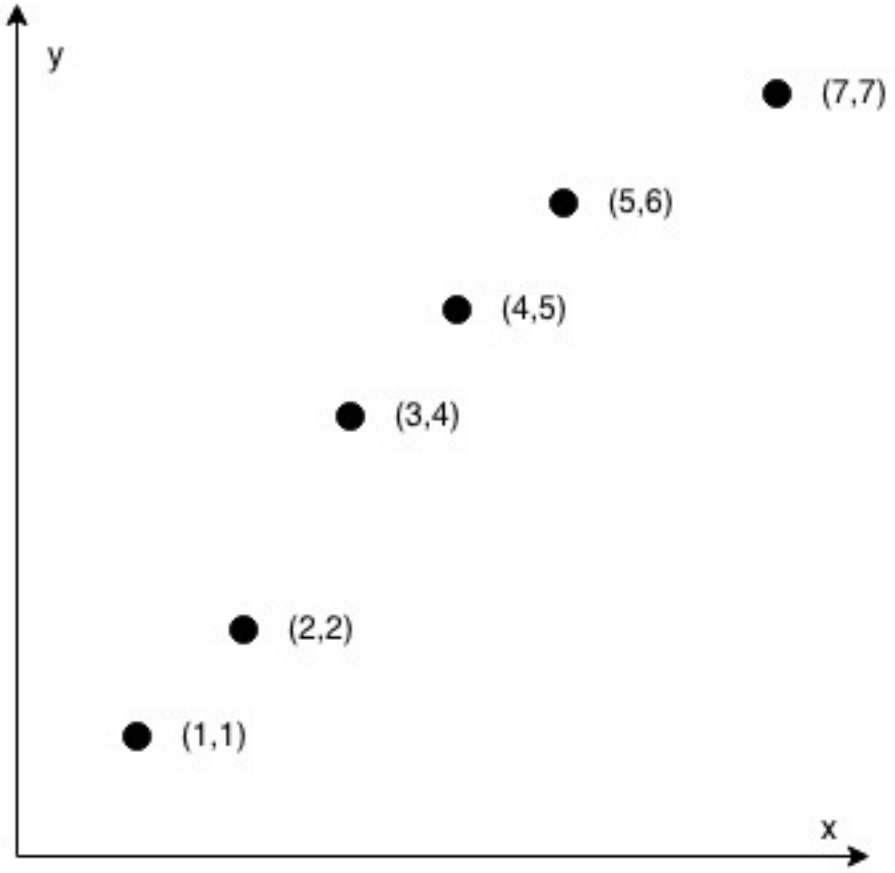
Input:

coordinates = [[1,2],[2,3],[3,4],[4,5],[5,6],[6,7]]

Output:

true

Example 2:



Input:

coordinates = [[1,1],[2,2],[3,4],[4,5],[5,6],[7,7]]

Output:

false

Constraints:

2 <= coordinates.length <= 1000

coordinates[i].length == 2

-10^4 <= coordinates[i][0], coordinates[i][1] <= 10^4

coordinates

contains no duplicate point.

Code Snippets

C++:

```
class Solution {
public:
bool checkStraightLine(vector<vector<int>>& coordinates) {

}
};
```

Java:

```
class Solution {
public boolean checkStraightLine(int[][] coordinates) {

}
}
```

Python3:

```
class Solution:
def checkStraightLine(self, coordinates: List[List[int]]) -> bool:
```

Python:

```
class Solution(object):
def checkStraightLine(self, coordinates):
"""
:type coordinates: List[List[int]]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[][]} coordinates
* @return {boolean}
*/
var checkStraightLine = function(coordinates) {
```

```
};
```

TypeScript:

```
function checkStraightLine(coordinates: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool CheckStraightLine(int[][] coordinates) {

}
}
```

C:

```
bool checkStraightLine(int** coordinates, int coordinatesSize, int*
coordinatesColSize) {

}
```

Go:

```
func checkStraightLine(coordinates [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun checkStraightLine(coordinates: Array<IntArray>): Boolean {

}
}
```

Swift:

```
class Solution {
func checkStraightLine(_ coordinates: [[Int]]) -> Bool {
```

```
}
}
```

Rust:

```
impl Solution {
pub fn check_straight_line(coordinates: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[][]} coordinates
# @return {Boolean}
def check_straight_line(coordinates)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $coordinates
* @return Boolean
*/
function checkStraightLine($coordinates) {

}
}
```

Dart:

```
class Solution {
bool checkStraightLine(List<List<int>> coordinates) {

}
}
```

Scala:

```
object Solution {
def checkStraightLine(coordinates: Array[Array[Int]]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec check_straight_line(coordinates :: [[integer]]) :: boolean
def check_straight_line(coordinates) do

end
end
```

Erlang:

```
-spec check_straight_line(Coordinates :: [[integer()]]) -> boolean().
check_straight_line(Coordinates) ->
.
```

Racket:

```
(define/contract (check-straight-line coordinates)
(-> (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check If It Is a Straight Line
* Difficulty: Easy
* Tags: array, math
*
* 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:
bool checkStraightLine(vector<vector<int>>& coordinates) {

}
};
```

Java Solution:

```
/**
* Problem: Check If It Is a Straight Line
* Difficulty: Easy
* Tags: array, math
*
* 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 boolean checkStraightLine(int[][] coordinates) {

}
}
```

Python3 Solution:

```
"""
Problem: Check If It Is a Straight Line
Difficulty: Easy
Tags: array, math

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 checkStraightLine(self, coordinates: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def checkStraightLine(self, coordinates):
"""
:type coordinates: List[List[int]]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Check If It Is a Straight Line
* Difficulty: Easy
* Tags: array, math
*
* 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[][]} coordinates
* @return {boolean}
*/
var checkStraightLine = function(coordinates) {

};
```

TypeScript Solution:

```
/**
* Problem: Check If It Is a Straight Line
* Difficulty: Easy
* Tags: array, math
*
* 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 checkStraightLine(coordinates: number[][]): boolean {
```

```
};
```

C# Solution:

```
/*
* Problem: Check If It Is a Straight Line
* Difficulty: Easy
* Tags: array, math
*
* 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 bool CheckStraightLine(int[][] coordinates) {

}
}
```

C Solution:

```
/*
* Problem: Check If It Is a Straight Line
* Difficulty: Easy
* Tags: array, math
*
* 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
*/

bool checkStraightLine(int** coordinates, int coordinatesSize, int*
coordinatesColSize) {

}
```

Go Solution:

```
// Problem: Check If It Is a Straight Line
// Difficulty: Easy
// Tags: array, math
//
// 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 checkStraightLine(coordinates [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun checkStraightLine(coordinates: Array<IntArray>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func checkStraightLine(_ coordinates: [[Int]]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Check If It Is a Straight Line
// Difficulty: Easy
// Tags: array, math
//
// 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 check_straight_line(coordinates: Vec<Vec<i32>>) -> bool {

}
```

```
}
```

Ruby Solution:

```
# @param {Integer[][]} coordinates
# @return {Boolean}
def check_straight_line(coordinates)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $coordinates
* @return Boolean
*/
function checkStraightLine($coordinates) {

}
}
```

Dart Solution:

```
class Solution {
bool checkStraightLine(List<List<int>> coordinates) {

}
}
```

Scala Solution:

```
object Solution {
def checkStraightLine(coordinates: Array[Array[Int]]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec check_straight_line(coordinates :: [[integer]]) :: boolean
def check_straight_line(coordinates) do

end
end
```

Erlang Solution:

```
-spec check_straight_line(Coordinates :: [[integer()]]) -> boolean().
check_straight_line(Coordinates) ->
.
```

Racket Solution:

```
(define/contract (check-straight-line coordinates)
(-> (listof (listof exact-integer?)) boolean?)
)
```