Problem 1496: Path Crossing

Problem Information

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

Problem Description

Given a string

path

, where

path[i] = 'N'

,

'S'

,

'E'

or

'W'

, each representing moving one unit north, south, east, or west, respectively. You start at the origin

(0, 0)

on a 2D plane and walk on the path specified by path

.

Return

true

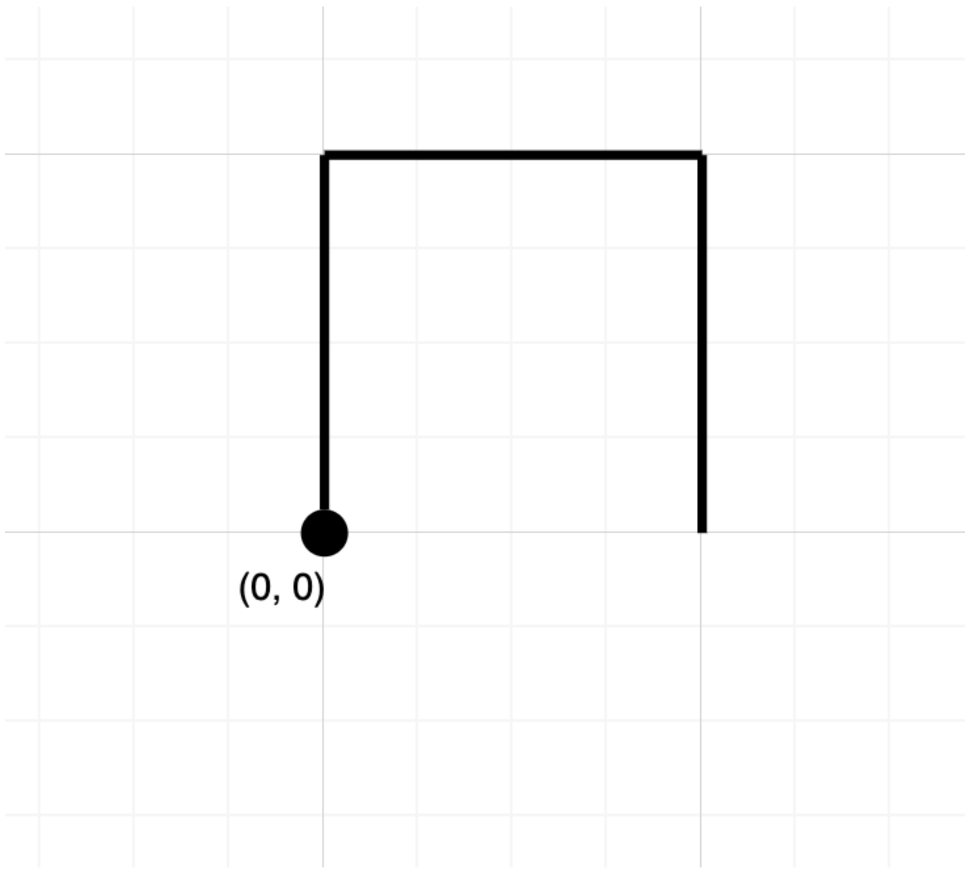
if the path crosses itself at any point, that is, if at any time you are on a location you have previously visited

. Return

false

otherwise.

Example 1:



Input:

path = "NES"

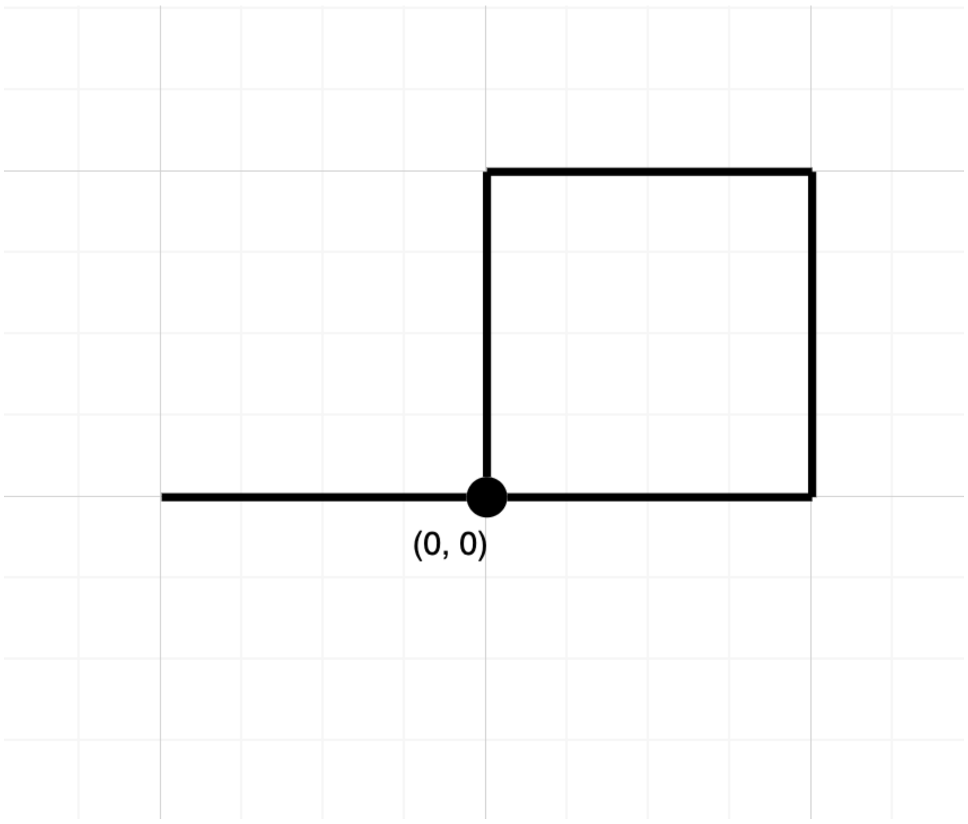
Output:

false

Explanation:

Notice that the path doesn't cross any point more than once.

Example 2:



Input:

path = "NESWW"

Output:

true

Explanation:

Notice that the path visits the origin twice.

Constraints:

1 <= path.length <= 10

4

path[i]

is either

'N'

,

'S'

,

'E'

, or

'W'

.

Code Snippets

C++:

```
class Solution {
public:
bool isPathCrossing(string path) {
}
};
```

Java:

```
class Solution {
public boolean isPathCrossing(String path) {

}
}
```

Python3:

```
class Solution:
def isPathCrossing(self, path: str) -> bool:
```

Python:

```
class Solution(object):
def isPathCrossing(self, path):
"""
:type path: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} path
* @return {boolean}
*/
var isPathCrossing = function(path) {

};
```

TypeScript:

```
function isPathCrossing(path: string): boolean {

};
```

C#:

}

```
public class Solution {
public bool IsPathCrossing(string path) {

}
}
```

C:

```
bool isPathCrossing(char* path) {

}
```

Go:

```
func isPathCrossing(path string) bool {

}
```

Kotlin:

```
class Solution {
fun isPathCrossing(path: String): Boolean {

}
}
```

Swift:

```
class Solution {
func isPathCrossing(_ path: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn is_path_crossing(path: String) -> bool {

}
}
```

Ruby:

```
# @param {String} path
# @return {Boolean}
def is_path_crossing(path)

end
```

PHP:

```
class Solution {

/**
* @param String $path
* @return Boolean
*/
function isPathCrossing($path) {

}
}
```

Dart:

```
class Solution {
bool isPathCrossing(String path) {

}
}
```

Scala:

```
object Solution {
def isPathCrossing(path: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec is_path_crossing(path :: String.t) :: boolean
def is_path_crossing(path) do

end
end
```

Erlang:

```
-spec is_path_crossing(Path :: unicode:unicode_binary()) -> boolean().
is_path_crossing(Path) ->
.
```

Racket:

```
(define/contract (is-path-crossing path)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Path Crossing
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
bool isPathCrossing(string path) {

}
};
```

Java Solution:

```
/**
* Problem: Path Crossing
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean isPathCrossing(String path) {

}
}
```

Python3 Solution:

```
"""
Problem: Path Crossing
Difficulty: Easy
Tags: string, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def isPathCrossing(self, path: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def isPathCrossing(self, path):
"""
:type path: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Path Crossing
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} path
* @return {boolean}
*/
var isPathCrossing = function(path) {

};
```

TypeScript Solution:

```
/**
* Problem: Path Crossing
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function isPathCrossing(path: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Path Crossing
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

public class Solution {
public bool IsPathCrossing(string path) {

}
}
```

C Solution:

```
/*
* Problem: Path Crossing
* Difficulty: Easy
* Tags: string, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool isPathCrossing(char* path) {

}
```

Go Solution:

```
// Problem: Path Crossing
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func isPathCrossing(path string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun isPathCrossing(path: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func isPathCrossing(_ path: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Path Crossing
// Difficulty: Easy
// Tags: string, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn is_path_crossing(path: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} path
# @return {Boolean}
def is_path_crossing(path)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String $path
* @return Boolean
*/
function isPathCrossing($path) {

}
}
```

Dart Solution:

```
class Solution {
bool isPathCrossing(String path) {

}
}
```

Scala Solution:

```
object Solution {
def isPathCrossing(path: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec is_path_crossing(path :: String.t) :: boolean
def is_path_crossing(path) do

end
end
```

Erlang Solution:

```
-spec is_path_crossing(Path :: unicode:unicode_binary()) -> boolean().
is_path_crossing(Path) ->
.
```

Racket Solution:

```
(define/contract (is-path-crossing path)
(-> string? boolean?)
)
```