# Problem 3288: Length of the Longest Increasing Path

## Problem Information

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

## Problem Description

You are given a 2D array of integers

coordinates

of length

n

and an integer

k

, where

0 <= k < n

.

coordinates[i] = [x

i

, y

i

]

indicates the point

(x

i

, y

i

)

in a 2D plane.

An

increasing path

of length

m

is defined as a list of points

(x

1

, y

1

)

,

(x

2

, y

2

)

,

(x

3

, y

3

)

, ...,

(x

m

, y

m

)

such that:

x

i

< x

i + 1

and

y

i

< y

i + 1

for all

i

where

1 <= i < m

.

(x

i

, y

i

)

is in the given coordinates for all

i

where

1 <= i <= m

.

Return the

maximum

length of an

increasing path

that contains

coordinates[k]

.

Example 1:

Input:

coordinates = [[3,1],[2,2],[4,1],[0,0],[5,3]], k = 1

Output:

3

Explanation:

(0, 0)

,

(2, 2)

,

(5, 3)

is the longest increasing path that contains

(2, 2)

.

Example 2:

Input:

coordinates = [[2,1],[7,0],[5,6]], k = 2

Output:

2

Explanation:

(2, 1)

,

(5, 6)

is the longest increasing path that contains

(5, 6)

.

Constraints:

1 <= n == coordinates.length <= 10

5

coordinates[i].length == 2

0 <= coordinates[i][0], coordinates[i][1] <= 10

9

All elements in

coordinates

are

distinct

.

0 <= k <= n - 1

## Code Snippets

**C++:**

```
class Solution {
public:
int maxPathLength(vector<vector<int>>& coordinates, int k) {

}
};
```

**Java:**

```
class Solution {
public int maxPathLength(int[][] coordinates, int k) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

```
/**
* @param {number[][]} coordinates
* @param {number} k
* @return {number}
*/
var maxPathLength = function(coordinates, k) {

};
```

**TypeScript:**

```
function maxPathLength(coordinates: number[][], k: number): number {

};
```

**C#:**

```
public class Solution {
public int MaxPathLength(int[][] coordinates, int k) {

}
}
```

**C:**

```
int maxPathLength(int** coordinates, int coordinatesSize, int*
coordinatesColSize, int k) {
```

```
}
```

**Go:**

```
func maxPathLength(coordinates [][]int, k int) int {

}
```

**Kotlin:**

```
class Solution {
fun maxPathLength(coordinates: Array<IntArray>, k: Int): Int {

}
}
```

**Swift:**

```
class Solution {
func maxPathLength(_ coordinates: [[Int]], _ k: Int) -> Int {

}
}
```

**Rust:**

```
impl Solution {
pub fn max_path_length(coordinates: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

**Ruby:**

```
# @param {Integer[][]} coordinates
# @param {Integer} k
# @return {Integer}
def max_path_length(coordinates, k)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $coordinates
* @param Integer $k
* @return Integer
*/
function maxPathLength($coordinates, $k) {

}
}
```

**Dart:**

```
class Solution {
int maxPathLength(List<List<int>> coordinates, int k) {

}
}
```

**Scala:**

```
object Solution {
def maxPathLength(coordinates: Array[Array[Int]], k: Int): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec max_path_length(coordinates :: [[integer]], k :: integer) :: integer
def max_path_length(coordinates, k) do

end
end
```

**Erlang:**

```
-spec max_path_length(Coordinates :: [[integer()]], K :: integer()) ->
integer().
max_path_length(Coordinates, K) ->
.
```

### Racket:

```
(define/contract (max-path-length coordinates k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Length of the Longest Increasing Path
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 maxPathLength(vector<vector<int>>& coordinates, int k) {

}
};
```

### Java Solution:

```
/**
* Problem: Length of the Longest Increasing Path
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 maxPathLength(int[][] coordinates, int k) {
```

```
}
}
```

### Python3 Solution:

```
"""
Problem: Length of the Longest Increasing Path
Difficulty: Hard
Tags: array, sort, search

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Length of the Longest Increasing Path
* Difficulty: Hard
* Tags: array, sort, search
*
* 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
* @param {number} k
* @return {number}
*/
var maxPathLength = function(coordinates, k) {

};
```

### TypeScript Solution:

```
/**
* Problem: Length of the Longest Increasing Path
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 maxPathLength(coordinates: number[][], k: number): number {

};
```

### C# Solution:

```
/*
* Problem: Length of the Longest Increasing Path
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 MaxPathLength(int[][] coordinates, int k) {
```

```
}
}
```

### C Solution:

```
/*
* Problem: Length of the Longest Increasing Path
* Difficulty: Hard
* Tags: array, sort, search
*
* 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 maxPathLength(int** coordinates, int coordinatesSize, int*
coordinatesColSize, int k) {

}
```

### Go Solution:

```
// Problem: Length of the Longest Increasing Path
// Difficulty: Hard
// Tags: array, sort, search
//
// 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 maxPathLength(coordinates [][]int, k int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun maxPathLength(coordinates: Array<IntArray>, k: Int): Int {

}
```

```
}
```

### Swift Solution:

```
class Solution {
func maxPathLength(_ coordinates: [[Int]], _ k: Int) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Length of the Longest Increasing Path
// Difficulty: Hard
// Tags: array, sort, search
//
// 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_path_length(coordinates: Vec<Vec<i32>>, k: i32) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} coordinates
# @param {Integer} k
# @return {Integer}
def max_path_length(coordinates, k)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[][] $coordinates
```

```
* @param Integer $k
* @return Integer
*/
function maxPathLength($coordinates, $k) {

}
}
```

**Dart Solution:**

```
class Solution {
int maxPathLength(List<List<int>> coordinates, int k) {

}
}
```

**Scala Solution:**

```
object Solution {
def maxPathLength(coordinates: Array[Array[Int]], k: Int): Int = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec max_path_length(coordinates :: [[integer]], k :: integer) :: integer
def max_path_length(coordinates, k) do

end
end
```

**Erlang Solution:**

```
-spec max_path_length(Coordinates :: [[integer()]], K :: integer()) ->
integer().
max_path_length(Coordinates, K) ->
.
```

**Racket Solution:**

```
(define/contract (max-path-length coordinates k)
(-> (listof (listof exact-integer?)) exact-integer? exact-integer?)
)
```