Problem 1621: Number of Sets of K Non-Overlapping Line Segments

Problem Information

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

Problem Description

Given

n

points on a 1-D plane, where the

i

th

point (from

0

to

n-1

) is at

x = i

, find the number of ways we can draw

exactly

k

non-overlapping

line segments such that each segment covers two or more points. The endpoints of each segment must have

integral coordinates

. The

k

line segments

do not

have to cover all

n

points, and they are

allowed

to share endpoints.

Return

the number of ways we can draw

k

non-overlapping line segments

.

Since this number can be huge, return it modulo

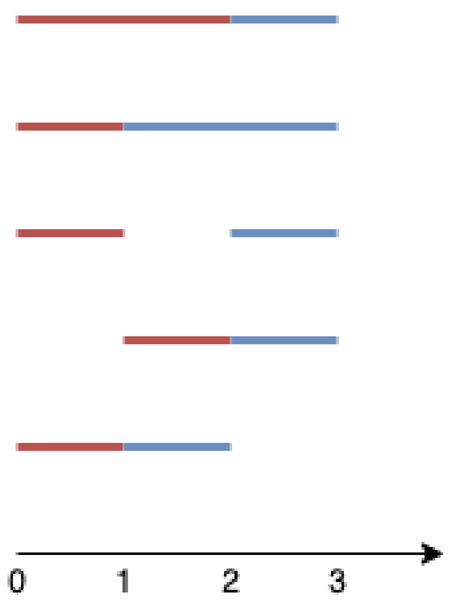
10

9

+ 7

.

Example 1:



Input:

n = 4, k = 2

Output:

5

Explanation:

The two line segments are shown in red and blue. The image above shows the 5 different ways {(0,2),(2,3)}, {(0,1),(1,3)}, {(0,1),(2,3)}, {(1,2),(2,3)}, {(0,1),(1,2)}.

Example 2:

Input:

n = 3, k = 1

Output:

3

Explanation:

The 3 ways are {(0,1)}, {(0,2)}, {(1,2)}.

Example 3:

Input:

n = 30, k = 7

Output:

796297179

Explanation:

The total number of possible ways to draw 7 line segments is 3796297200. Taking this number modulo 10

9

+ 7 gives us 796297179.

Constraints:

2 <= n <= 1000

1 <= k <= n-1

Code Snippets

C++:

```
class Solution {
public:
int numberOfSets(int n, int k) {

}
};
```

Java:

```
class Solution {
public int numberOfSets(int n, int k) {

}
}
```

Python3:

```
class Solution:
def numberOfSets(self, n: int, k: int) -> int:
```

Python:

```
class Solution(object):
def numberOfSets(self, n, k):
"""
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @return {number}
*/
var numberOfSets = function(n, k) {
};
```

TypeScript:

```
function numberOfSets(n: number, k: number): number {

};
```

C#:

```
public class Solution {
public int NumberOfSets(int n, int k) {

}
}
```

C:

```
int numberOfSets(int n, int k) {

}
```

Go:

```
func numberOfSets(n int, k int) int {

}
```

Kotlin:

```
class Solution {
fun numberOfSets(n: Int, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func numberOfSets(_ n: Int, _ k: Int) -> Int {
}
}
```

Rust:

```
impl Solution {
pub fn number_of_sets(n: i32, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def number_of_sets(n, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function numberOfSets($n, $k) {

}
}
```

Dart:

```
class Solution {
int numberOfSets(int n, int k) {

}
}
```

Scala:

```
object Solution {
def numberOfSets(n: Int, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_sets(n :: integer, k :: integer) :: integer
def number_of_sets(n, k) do

end
end
```

Erlang:

```
-spec number_of_sets(N :: integer(), K :: integer()) -> integer().
number_of_sets(N, K) ->
.
```

Racket:

```
(define/contract (number-of-sets n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Sets of K Non-Overlapping Line Segments
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
class Solution {
public:
int numberOfSets(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Sets of K Non-Overlapping Line Segments
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numberOfSets(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Sets of K Non-Overlapping Line Segments
Difficulty: Medium
Tags: dp, math

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def numberOfSets(self, n: int, k: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def numberOfSets(self, n, k):
"""
:type n: int
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Sets of K Non-Overlapping Line Segments
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @param {number} k
* @return {number}
*/
var numberOfSets = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Sets of K Non-Overlapping Line Segments
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numberOfSets(n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Sets of K Non-Overlapping Line Segments
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int NumberOfSets(int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Number of Sets of K Non-Overlapping Line Segments
* Difficulty: Medium
* Tags: dp, math
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int numberOfSets(int n, int k) {

}
```

Go Solution:

```
// Problem: Number of Sets of K Non-Overlapping Line Segments
// Difficulty: Medium
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func numberOfSets(n int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfSets(n: Int, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfSets(_ n: Int, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Sets of K Non-Overlapping Line Segments
// Difficulty: Medium
// Tags: dp, math
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn number_of_sets(n: i32, k: i32) -> i32 {



}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @return {Integer}
def number_of_sets(n, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @return Integer
*/
function numberOfSets($n, $k) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfSets(int n, int k) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfSets(n: Int, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_sets(n :: integer, k :: integer) :: integer
def number_of_sets(n, k) do

end
end
```

Erlang Solution:

```
-spec number_of_sets(N :: integer(), K :: integer()) -> integer().
number_of_sets(N, K) ->
.
```

Racket Solution:

```
(define/contract (number-of-sets n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```