Problem 1420: Build Array Where You Can Find The Maximum Exactly K Comparisons

Problem Information

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

Problem Description

You are given three integers

n

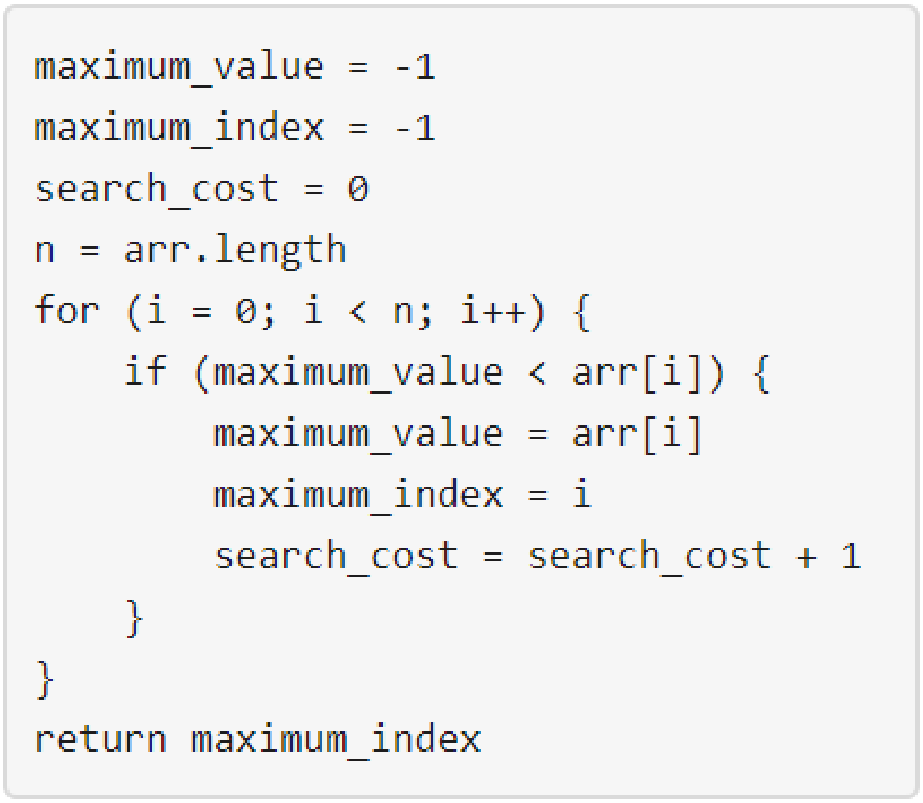
,

m

and

k

. Consider the following algorithm to find the maximum element of an array of positive integers:



```
maximum_value = -1
maximum_index = -1
search_cost = 0
n = arr.length
for (i = 0; i < n; i++) {
    if (maximum_value < arr[i]) {
        maximum_value = arr[i]
        maximum_index = i
        search_cost = search_cost + 1
    }
}
return maximum_index
```

You should build the array arr which has the following properties:

arr

has exactly

n

integers.

1 <= arr[i] <= m

where

(0 <= i < n)

.

After applying the mentioned algorithm to arr

, the value

search_cost

is equal to

k

.

Return

the number of ways

to build the array

arr

under the mentioned conditions. As the answer may grow large, the answer

must be

computed modulo

10

9

+ 7

.

Example 1:

Input:

n = 2, m = 3, k = 1

Output:

6

Explanation:

The possible arrays are [1, 1], [2, 1], [2, 2], [3, 1], [3, 2] [3, 3]

Example 2:

Input:

n = 5, m = 2, k = 3

Output:

0

Explanation:

There are no possible arrays that satisfy the mentioned conditions.

Example 3:

Input:

n = 9, m = 1, k = 1

Output:

1

Explanation:

The only possible array is [1, 1, 1, 1, 1, 1, 1, 1, 1]

Constraints:

1 <= n <= 50

1 <= m <= 100

0 <= k <= n

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int numOfArrays(int n, int m, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numOfArrays(n: number, m: number, k: number): number {

};
```

C#:

```
public class Solution {
public int NumOfArrays(int n, int m, int k) {

}
}
```

C:

```
int numOfArrays(int n, int m, int k) {

}
```

Go:

```
func numOfArrays(n int, m int, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int numOfArrays(int n, int m, int k) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (num-of-arrays n m k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Build Array Where You Can Find The Maximum Exactly K Comparisons
* Difficulty: Hard
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int numOfArrays(int n, int m, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Build Array Where You Can Find The Maximum Exactly K Comparisons
* Difficulty: Hard
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numOfArrays(int n, int m, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Build Array Where You Can Find The Maximum Exactly K Comparisons
```

```
Difficulty: Hard
Tags: array, dp, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Build Array Where You Can Find The Maximum Exactly K Comparisons
* Difficulty: Hard
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @param {number} m
* @param {number} k
* @return {number}
```

```
*/
var numOfArrays = function(n, m, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Build Array Where You Can Find The Maximum Exactly K Comparisons
* Difficulty: Hard
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numOfArrays(n: number, m: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Build Array Where You Can Find The Maximum Exactly K Comparisons
* Difficulty: Hard
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int NumOfArrays(int n, int m, int k) {

}
}
```

C Solution:

```
/*
* Problem: Build Array Where You Can Find The Maximum Exactly K Comparisons
* Difficulty: Hard
* Tags: array, dp, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int numOfArrays(int n, int m, int k) {

}
```

Go Solution:

```
// Problem: Build Array Where You Can Find The Maximum Exactly K Comparisons
// Difficulty: Hard
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numOfArrays(n int, m int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Build Array Where You Can Find The Maximum Exactly K Comparisons
// Difficulty: Hard
// Tags: array, dp, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int numOfArrays(int n, int m, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-of-arrays n m k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```