Problem 920: Number of Music Playlists

Problem Information

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

Problem Description

Your music player contains

n

different songs. You want to listen to

goal

songs (not necessarily different) during your trip. To avoid boredom, you will create a playlist so that:

Every song is played

at least once

.

A song can only be played again only if

k

other songs have been played.

Given

n

,

goal

, and

k

, return

the number of possible playlists that you can create

. Since the answer can be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

n = 3, goal = 3, k = 1

Output:

6

Explanation:

There are 6 possible playlists: [1, 2, 3], [1, 3, 2], [2, 1, 3], [2, 3, 1], [3, 1, 2], and [3, 2, 1].

Example 2:

Input:

n = 2, goal = 3, k = 0

Output:

6

Explanation:

There are 6 possible playlists: [1, 1, 2], [1, 2, 1], [2, 1, 1], [2, 2, 1], [2, 1, 2], and [1, 2, 2].

Example 3:

Input:

n = 2, goal = 3, k = 1

Output:

2

Explanation:

There are 2 possible playlists: [1, 2, 1] and [2, 1, 2].

Constraints:

0 <= k < n <= goal <= 100

Code Snippets

C++:

```
class Solution {
public:
```

```
int numMusicPlaylists(int n, int goal, int k) {

}
};
```

Java:

```
class Solution {
public int numMusicPlaylists(int n, int goal, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function numMusicPlaylists(n: number, goal: number, k: number): number {

};
```

C#:

```
public class Solution {
public int NumMusicPlaylists(int n, int goal, int k) {

}
}
```

C:

```
int numMusicPlaylists(int n, int goal, int k) {

}
```

Go:

```
func numMusicPlaylists(n int, goal int, k int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int numMusicPlaylists(int n, int goal, int k) {

}
}
```

Scala:

```
object Solution {
def numMusicPlaylists(n: Int, goal: Int, k: Int): Int = {
}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (num-music-playlists n goal k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Music Playlists
* Difficulty: Hard
* 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 numMusicPlaylists(int n, int goal, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Music Playlists
* Difficulty: Hard
* 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 numMusicPlaylists(int n, int goal, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Music Playlists
Difficulty: Hard
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 numMusicPlaylists(self, n: int, goal: int, k: int) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Music Playlists
* Difficulty: Hard
* 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} goal
* @param {number} k
* @return {number}
*/
var numMusicPlaylists = function(n, goal, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Music Playlists
* Difficulty: Hard
* 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 numMusicPlaylists(n: number, goal: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Music Playlists
* Difficulty: Hard
* 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 NumMusicPlaylists(int n, int goal, int k) {

}
}
```

C Solution:

```
/*
* Problem: Number of Music Playlists
* Difficulty: Hard
* 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 numMusicPlaylists(int n, int goal, int k) {
}
```

Go Solution:

```
// Problem: Number of Music Playlists
// Difficulty: Hard
// 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 numMusicPlaylists(n int, goal int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Music Playlists
// Difficulty: Hard
// 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 num_music_playlists(n: i32, goal: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int numMusicPlaylists(int n, int goal, int k) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-music-playlists n goal k)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```