Problem 2327: Number of People Aware of a Secret

Problem Information

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

Problem Description

On day

1

, one person discovers a secret.

You are given an integer

delay

, which means that each person will

share

the secret with a new person

every day

, starting from

delay

days after discovering the secret. You are also given an integer

forget

, which means that each person will

forget

the secret

forget

days after discovering it. A person

cannot

share the secret on the same day they forgot it, or on any day afterwards.

Given an integer

n

, return

the number of people who know the secret at the end of day

n

. Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

n = 6, delay = 2, forget = 4

Output:

5

Explanation:

Day 1: Suppose the first person is named A. (1 person) Day 2: A is the only person who knows the secret. (1 person) Day 3: A shares the secret with a new person, B. (2 people) Day 4: A shares the secret with a new person, C. (3 people) Day 5: A forgets the secret, and B shares the secret with a new person, D. (3 people) Day 6: B shares the secret with E, and C shares the secret with F. (5 people)

Example 2:

Input:

n = 4, delay = 1, forget = 3

Output:

6

Explanation:

Day 1: The first person is named A. (1 person) Day 2: A shares the secret with B. (2 people) Day 3: A and B share the secret with 2 new people, C and D. (4 people) Day 4: A forgets the secret. B, C, and D share the secret with 3 new people. (6 people)

Constraints:

2 <= n <= 1000

1 <= delay < forget <= n

Code Snippets

C++:

```
class Solution {
public:
int peopleAwareOfSecret(int n, int delay, int forget) {

}
};
```

Java:

```
class Solution {
public int peopleAwareOfSecret(int n, int delay, int forget) {

}
}
```

Python3:

```
class Solution:
def peopleAwareOfSecret(self, n: int, delay: int, forget: int) -> int:
```

Python:

```
class Solution(object):
def peopleAwareOfSecret(self, n, delay, forget):
"""
:type n: int
:type delay: int
:type forget: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} delay
* @param {number} forget
* @return {number}
*/
```

```
var peopleAwareOfSecret = function(n, delay, forget) {

};
```

TypeScript:

```
function peopleAwareOfSecret(n: number, delay: number, forget: number):
number {

};
```

C#:

```
public class Solution {
public int PeopleAwareOfSecret(int n, int delay, int forget) {

}
}
```

C:

```
int peopleAwareOfSecret(int n, int delay, int forget) {

}
```

Go:

```
func peopleAwareOfSecret(n int, delay int, forget int) int {

}
```

Kotlin:

```
class Solution {
fun peopleAwareOfSecret(n: Int, delay: Int, forget: Int): Int {

}
}
```

Swift:

```
class Solution {
func peopleAwareOfSecret(_ n: Int, _ delay: Int, _ forget: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn people_aware_of_secret(n: i32, delay: i32, forget: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} delay
# @param {Integer} forget
# @return {Integer}
def people_aware_of_secret(n, delay, forget)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $delay
* @param Integer $forget
* @return Integer
*/
function peopleAwareOfSecret($n, $delay, $forget) {

}
}
```

Dart:

```
class Solution {
int peopleAwareOfSecret(int n, int delay, int forget) {
```

```
}
}
```

Scala:

```
object Solution {
def peopleAwareOfSecret(n: Int, delay: Int, forget: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec people_aware_of_secret(n :: integer, delay :: integer, forget ::
integer) :: integer
def people_aware_of_secret(n, delay, forget) do

end
end
```

Erlang:

```
-spec people_aware_of_secret(N :: integer(), Delay :: integer(), Forget ::
integer()) -> integer().
people_aware_of_secret(N, Delay, Forget) ->
.
```

Racket:

```
(define/contract (people-aware-of-secret n delay forget)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of People Aware of a Secret
* Difficulty: Medium
* Tags: dp, queue
*
* 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 peopleAwareOfSecret(int n, int delay, int forget) {

}
};
```

Java Solution:

```
/**
* Problem: Number of People Aware of a Secret
* Difficulty: Medium
* Tags: dp, queue
*
* 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 peopleAwareOfSecret(int n, int delay, int forget) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of People Aware of a Secret
Difficulty: Medium
Tags: dp, queue
```

```
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 peopleAwareOfSecret(self, n: int, delay: int, forget: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def peopleAwareOfSecret(self, n, delay, forget):
"""
:type n: int
:type delay: int
:type forget: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of People Aware of a Secret
* Difficulty: Medium
* Tags: dp, queue
*
* 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} delay
* @param {number} forget
* @return {number}
*/
var peopleAwareOfSecret = function(n, delay, forget) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Number of People Aware of a Secret
* Difficulty: Medium
* Tags: dp, queue
*
* 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 peopleAwareOfSecret(n: number, delay: number, forget: number):
number {

};
```

C# Solution:

```
/*
* Problem: Number of People Aware of a Secret
* Difficulty: Medium
* Tags: dp, queue
*
* 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 PeopleAwareOfSecret(int n, int delay, int forget) {

}
}
```

C Solution:

```
/*
* Problem: Number of People Aware of a Secret
```

```
* Difficulty: Medium
* Tags: dp, queue
*
* 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 peopleAwareOfSecret(int n, int delay, int forget) {

}
```

Go Solution:

```
// Problem: Number of People Aware of a Secret
// Difficulty: Medium
// Tags: dp, queue
//
// 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 peopleAwareOfSecret(n int, delay int, forget int) int {

}
```

Kotlin Solution:

```
class Solution {
fun peopleAwareOfSecret(n: Int, delay: Int, forget: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func peopleAwareOfSecret(_ n: Int, _ delay: Int, _ forget: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of People Aware of a Secret
// Difficulty: Medium
// Tags: dp, queue
//
// 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 people_aware_of_secret(n: i32, delay: i32, forget: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} delay
# @param {Integer} forget
# @return {Integer}
def people_aware_of_secret(n, delay, forget)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $delay
* @param Integer $forget
* @return Integer
*/
function peopleAwareOfSecret($n, $delay, $forget) {

}
}
```

Dart Solution:

```
class Solution {
int peopleAwareOfSecret(int n, int delay, int forget) {

}
}
```

Scala Solution:

```
object Solution {
def peopleAwareOfSecret(n: Int, delay: Int, forget: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec people_aware_of_secret(n :: integer, delay :: integer, forget ::
integer) :: integer
def people_aware_of_secret(n, delay, forget) do

end
end
```

Erlang Solution:

```
-spec people_aware_of_secret(N :: integer(), Delay :: integer(), Forget ::
integer()) -> integer().
people_aware_of_secret(N, Delay, Forget) ->
.
```

Racket Solution:

```
(define/contract (people-aware-of-secret n delay forget)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```