Problem 1692: Count Ways to Distribute Candies

Problem Information

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

Problem Description

There are

n

unique

candies (labeled

1

through

n

) and

k

bags. You are asked to distribute

all

the candies into the bags such that every bag has

at least

one candy.

There can be multiple ways to distribute the candies. Two ways are considered

different

if the candies in one bag in the first way are not all in the same bag in the second way. The order of the bags and the order of the candies within each bag do not matter.

For example,

(1), (2,3)

and

(2), (1,3)

are considered different because candies

2

and

3

in the bag

(2,3)

in the first way are not in the same bag in the second way (they are split between the bags

(

2

)

and

(1,

3

)

). However,

(1), (2,3)

and

(3,2), (1)

are considered the same because the candies in each bag are all in the same bags in both ways.

Given two integers,

n

and

k

, return

the

number

of different ways to distribute the candies

. As the answer may be too large, return it

modulo

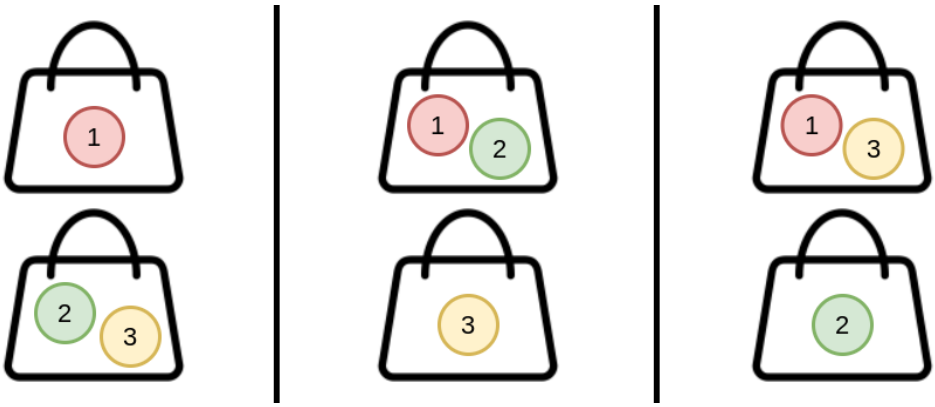
10

9

+ 7

.

Example 1:



Input:

n = 3, k = 2

Output:

3

Explanation:

You can distribute 3 candies into 2 bags in 3 ways: (1), (2,3) (1,2), (3) (1,3), (2)

Example 2:

Input:

n = 4, k = 2

Output:

7

Explanation:

You can distribute 4 candies into 2 bags in 7 ways: (1), (2,3,4) (1,2), (3,4) (1,3), (2,4) (1,4), (2,3) (1,2,3), (4) (1,2,4), (3) (1,3,4), (2)

Example 3:

Input:

n = 20, k = 5

Output:

206085257

Explanation:

You can distribute 20 candies into 5 bags in 1881780996 ways. 1881780996 modulo 10

9

+ 7 = 206085257.

Constraints:

1 <= k <= n <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (ways-to-distribute n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Ways to Distribute Candies
* Difficulty: Hard
* Tags: dp
*
* 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 waysToDistribute(int n, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Count Ways to Distribute Candies
```

```
* Difficulty: Hard
* Tags: dp
*
* 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 waysToDistribute(int n, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Ways to Distribute Candies
Difficulty: Hard
Tags: dp

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 waysToDistribute(self, n: int, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Ways to Distribute Candies
* Difficulty: Hard
* Tags: dp
*
* 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 waysToDistribute = function(n, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Ways to Distribute Candies
* Difficulty: Hard
* Tags: dp
*
* 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 waysToDistribute(n: number, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Ways to Distribute Candies
* Difficulty: Hard
* Tags: dp
```

```
*
* 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 WaysToDistribute(int n, int k) {

}
}
```

C Solution:

```
/*
* Problem: Count Ways to Distribute Candies
* Difficulty: Hard
* Tags: dp
*
* 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 waysToDistribute(int n, int k) {

}
```

Go Solution:

```
// Problem: Count Ways to Distribute Candies
// Difficulty: Hard
// Tags: dp
//
// 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 waysToDistribute(n int, k int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Ways to Distribute Candies
// Difficulty: Hard
// Tags: dp
//
// 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 ways_to_distribute(n: i32, k: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (ways-to-distribute n k)
(-> exact-integer? exact-integer? exact-integer?)
)
```