

Problem 1359: Count All Valid Pickup and Delivery Options

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given

n

orders, each order consists of a pickup and a delivery service.

Count all valid pickup/delivery possible sequences such that delivery(i) is always after of pickup(i).

Since the answer may be too large, return it modulo $10^9 + 7$.

Example 1:

Input:

$n = 1$

Output:

1

Explanation:

Unique order (P1, D1), Delivery 1 always is after of Pickup 1.

Example 2:

Input:

n = 2

Output:

6

Explanation:

All possible orders: (P1,P2,D1,D2), (P1,P2,D2,D1), (P1,D1,P2,D2), (P2,P1,D1,D2), (P2,P1,D2,D1) and (P2,D2,P1,D1). This is an invalid order (P1,D2,P2,D1) because Pickup 2 is after of Delivery 2.

Example 3:

Input:

n = 3

Output:

90

Constraints:

$1 \leq n \leq 500$

Code Snippets

C++:

```
class Solution {  
public:  
    int countOrders(int n) {  
  
    }  
}
```

```
};
```

Java:

```
class Solution {  
    public int countOrders(int n) {  
  
    }  
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function countOrders(n: number): number {  
  
};
```

C#:

```
public class Solution {  
    public int CountOrders(int n) {  
        }  
        }  
}
```

C:

```
int countOrders(int n) {  
    }  
}
```

Go:

```
func countOrders(n int) int {  
    }  
}
```

Kotlin:

```
class Solution {  
    fun countOrders(n: Int): Int {  
        }  
        }  
}
```

Swift:

```
class Solution {  
    func countOrders(_ n: Int) -> Int {  
        }  
        }  
}
```

Rust:

```
impl Solution {  
    pub fn count_orders(n: i32) -> i32 {  
        }  
        }  
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def count_orders(n)

end
```

PHP:

```
class Solution {

    /**
     * @param Integer $n
     * @return Integer
     */
    function countOrders($n) {

    }
}
```

Dart:

```
class Solution {
int countOrders(int n) {

}
```

Scala:

```
object Solution {
def countOrders(n: Int): Int = {

}
```

Elixir:

```
defmodule Solution do
@spec count_orders(n :: integer) :: integer
def count_orders(n) do

end
end
```

Erlang:

```
-spec count_orders(N :: integer()) -> integer().  
count_orders(N) ->  
.
```

Racket:

```
(define/contract (count-orders n)  
  (-> exact-integer? exact-integer?)  
)
```

Solutions

C++ Solution:

```
/*  
 * Problem: Count All Valid Pickup and Delivery Options  
 * 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 countOrders(int n) {  
  
    }  
};
```

Java Solution:

```
/**  
 * Problem: Count All Valid Pickup and Delivery Options  
 * 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 countOrders(int n) {

}
}

```

Python3 Solution:

```

"""
Problem: Count All Valid Pickup and Delivery Options
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 countOrders(self, n: int) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

class Solution(object):
    def countOrders(self, n):
        """
        :type n: int
        :rtype: int
        """

```

JavaScript Solution:

```

/**
 * Problem: Count All Valid Pickup and Delivery Options
 * 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
* @return {number}
*/
var countOrders = function(n) {
}

```

TypeScript Solution:

```

/**
* Problem: Count All Valid Pickup and Delivery Options
* 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 countOrders(n: number): number {
}

```

C# Solution:

```

/*
* Problem: Count All Valid Pickup and Delivery Options
* 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

```

```
*/\n\npublic class Solution {\n    public int CountOrders(int n) {\n\n    }\n}\n\n}
```

C Solution:

```
/*\n * Problem: Count All Valid Pickup and Delivery Options\n * Difficulty: Hard\n * Tags: dp, math\n *\n * Approach: Dynamic programming with memoization or tabulation\n * Time Complexity: O(n * m) where n and m are problem dimensions\n * Space Complexity: O(n) or O(n * m) for DP table\n */\n\nint countOrders(int n) {\n\n}
```

Go Solution:

```
// Problem: Count All Valid Pickup and Delivery Options\n// Difficulty: Hard\n// Tags: dp, math\n//\n// Approach: Dynamic programming with memoization or tabulation\n// Time Complexity: O(n * m) where n and m are problem dimensions\n// Space Complexity: O(n) or O(n * m) for DP table\n\nfunc countOrders(n int) int {\n\n}
```

Kotlin Solution:

```
class Solution {  
    fun countOrders(n: Int): Int {  
        }  
        }  
}
```

Swift Solution:

```
class Solution {  
    func countOrders(_ n: Int) -> Int {  
        }  
        }  
}
```

Rust Solution:

```
// Problem: Count All Valid Pickup and Delivery Options  
// 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 count_orders(n: i32) -> i32 {  
        }  
        }  
}
```

Ruby Solution:

```
# @param {Integer} n  
# @return {Integer}  
def count_orders(n)  
  
end
```

PHP Solution:

```
class Solution {
```

```
/**  
 * @param Integer $n  
 * @return Integer  
 */  
function countOrders($n) {  
  
}  
}
```

Dart Solution:

```
class Solution {  
int countOrders(int n) {  
  
}  
}
```

Scala Solution:

```
object Solution {  
def countOrders(n: Int): Int = {  
  
}  
}
```

Elixir Solution:

```
defmodule Solution do  
@spec count_orders(n :: integer) :: integer  
def count_orders(n) do  
  
end  
end
```

Erlang Solution:

```
-spec count_orders(N :: integer()) -> integer().  
count_orders(N) ->  
.
```

Racket Solution:

```
(define/contract (count-orders n)
  (-> exact-integer? exact-integer?))
```