Problem 2585: Number of Ways to Earn Points

Problem Information

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

Problem Description

There is a test that has

n

types of questions. You are given an integer

target

and a

0-indexed

2D integer array

types

where

types[i] = [count

i

, marks

i

]

indicates that there are

count

i

questions of the

i

th

type, and each one of them is worth

marks

i

points.

Return

the number of ways you can earn

exactly

target

points in the exam

. Since the answer may be too large, return it

modulo

10

9

+ 7

.

Note

that questions of the same type are indistinguishable.

For example, if there are

3

questions of the same type, then solving the

1

st

and

2

nd

questions is the same as solving the

1

st

and

3

rd

questions, or the

2

nd

and

3

rd

questions.

Example 1:

Input:

target = 6, types = [[6,1],[3,2],[2,3]]

Output:

7

Explanation:

You can earn 6 points in one of the seven ways: - Solve 6 questions of the 0

th

type: 1 + 1 + 1 + 1 + 1 + 1 = 6 - Solve 4 questions of the 0

th

type and 1 question of the 1

st

type: 1 + 1 + 1 + 1 + 2 = 6 - Solve 2 questions of the 0

th

type and 2 questions of the 1

st

type: 1 + 1 + 2 + 2 = 6 - Solve 3 questions of the 0

th

type and 1 question of the 2

nd

type: 1 + 1 + 1 + 3 = 6 - Solve 1 question of the 0

th

type, 1 question of the 1

st

type and 1 question of the 2

nd

type: 1 + 2 + 3 = 6 - Solve 3 questions of the 1

st

type: 2 + 2 + 2 = 6 - Solve 2 questions of the 2

nd

type: 3 + 3 = 6

Example 2:

Input:

target = 5, types = [[50,1],[50,2],[50,5]]

Output:

4

Explanation:

You can earn 5 points in one of the four ways: - Solve 5 questions of the 0

th

type: 1 + 1 + 1 + 1 + 1 = 5 - Solve 3 questions of the 0

th

type and 1 question of the 1

st

type: 1 + 1 + 1 + 2 = 5 - Solve 1 questions of the 0

th

type and 2 questions of the 1

st

type: 1 + 2 + 2 = 5 - Solve 1 question of the 2

nd

type: 5

Example 3:

Input:

target = 18, types = [[6,1],[3,2],[2,3]] Output:

1

Explanation:

You can only earn 18 points by answering all questions.

Constraints:

1 <= target <= 1000

n == types.length

1 <= n <= 50

types[i].length == 2

1 <= count

i

, marks

i

<= 50

Code Snippets

C++:

```
class Solution {
public:
int waysToReachTarget(int target, vector<vector<int>>& types) {

}
};
```

Java:

```
class Solution {
public int waysToReachTarget(int target, int[][] types) {

}
}
```

Python3:

```
class Solution:
def waysToReachTarget(self, target: int, types: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def waysToReachTarget(self, target, types):
"""
:type target: int
:type types: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} target
* @param {number[][]} types
* @return {number}
*/
var waysToReachTarget = function(target, types) {

};
```

TypeScript:

```
function waysToReachTarget(target: number, types: number[][]): number {

};
```

C#:

```
public class Solution {
public int WaysToReachTarget(int target, int[][] types) {

}
}
```

C:

```
int waysToReachTarget(int target, int** types, int typesSize, int*
typesColSize) {

}
```

Go:

```
func waysToReachTarget(target int, types [][]int) int {

}
```

Kotlin:

```
class Solution {
fun waysToReachTarget(target: Int, types: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func waysToReachTarget(_ target: Int, _ types: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn ways_to_reach_target(target: i32, types: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} target
# @param {Integer[][]} types
# @return {Integer}
def ways_to_reach_target(target, types)

end
```

PHP:

```
class Solution {

/**
* @param Integer $target
* @param Integer[][] $types
* @return Integer
*/
function waysToReachTarget($target, $types) {

}
}
```

Dart:

```
class Solution {
int waysToReachTarget(int target, List<List<int>> types) {

}
}
```

Scala:

```
object Solution {
def waysToReachTarget(target: Int, types: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec ways_to_reach_target(target :: integer, types :: [[integer]]) ::
```

```
integer
def ways_to_reach_target(target, types) do

end
end
```

Erlang:

```
-spec ways_to_reach_target(Target :: integer(), Types :: [[integer()]]) ->
integer().
ways_to_reach_target(Target, Types) ->
.
```

Racket:

```
(define/contract (ways-to-reach-target target types)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Earn Points
* Difficulty: Hard
* Tags: array, dp
*
* 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 waysToReachTarget(int target, vector<vector<int>>& types) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Earn Points
* Difficulty: Hard
* Tags: array, dp
*
* 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 waysToReachTarget(int target, int[][] types) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Earn Points
Difficulty: Hard
Tags: array, dp

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 waysToReachTarget(self, target: int, types: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def waysToReachTarget(self, target, types):
"""
:type target: int
:type types: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Ways to Earn Points
* Difficulty: Hard
* Tags: array, dp
*
* 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} target
* @param {number[][]} types
* @return {number}
*/
var waysToReachTarget = function(target, types) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Earn Points
* Difficulty: Hard
* Tags: array, dp
*
* 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 waysToReachTarget(target: number, types: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Earn Points
* Difficulty: Hard
* Tags: array, dp
*
* 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 WaysToReachTarget(int target, int[][] types) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Earn Points
* Difficulty: Hard
* Tags: array, dp
*
* 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 waysToReachTarget(int target, int** types, int typesSize, int*
typesColSize) {

}
```

Go Solution:

```
// Problem: Number of Ways to Earn Points
// Difficulty: Hard
// Tags: array, dp
//
// 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 waysToReachTarget(target int, types [][]int) int {
}
```

Kotlin Solution:

```
class Solution {
fun waysToReachTarget(target: Int, types: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func waysToReachTarget(_ target: Int, _ types: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Earn Points
// Difficulty: Hard
// Tags: array, dp
//
// 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 ways_to_reach_target(target: i32, types: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} target
# @param {Integer[][]} types
# @return {Integer}
def ways_to_reach_target(target, types)
```

```


end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $target
* @param Integer[][] $types
* @return Integer
*/
function waysToReachTarget($target, $types) {

}
}
```

Dart Solution:

```
class Solution {
int waysToReachTarget(int target, List<List<int>> types) {

}
}
```

Scala Solution:

```
object Solution {
def waysToReachTarget(target: Int, types: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec ways_to_reach_target(target :: integer, types :: [[integer]]) ::
integer
def ways_to_reach_target(target, types) do

end
```

```

end
```

Erlang Solution:

```
-spec ways_to_reach_target(Target :: integer(), Types :: [[integer()]]) ->
integer().
ways_to_reach_target(Target, Types) ->
.
```

Racket Solution:

```
(define/contract (ways-to-reach-target target types)
(-> exact-integer? (listof (listof exact-integer?)) exact-integer?)
)
```