Problem 1916: Count Ways to Build Rooms in an Ant Colony

Problem Information

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

Problem Description

You are an ant tasked with adding

n

new rooms numbered

0

to

n-1

to your colony. You are given the expansion plan as a

0-indexed

integer array of length

n

,

prevRoom

, where

prevRoom[i]

indicates that you must build room

prevRoom[i]

before building room

i

, and these two rooms must be connected

directly

. Room

0

is already built, so

prevRoom[0] = -1

. The expansion plan is given such that once all the rooms are built, every room will be reachable from room

0

.

You can only build

one room

at a time, and you can travel freely between rooms you have

already built

only if they are

connected

. You can choose to build

any room

as long as its

previous room

is already built.

Return

the

number of different orders

you can build all the rooms in

. Since the answer may be large, return it

modulo

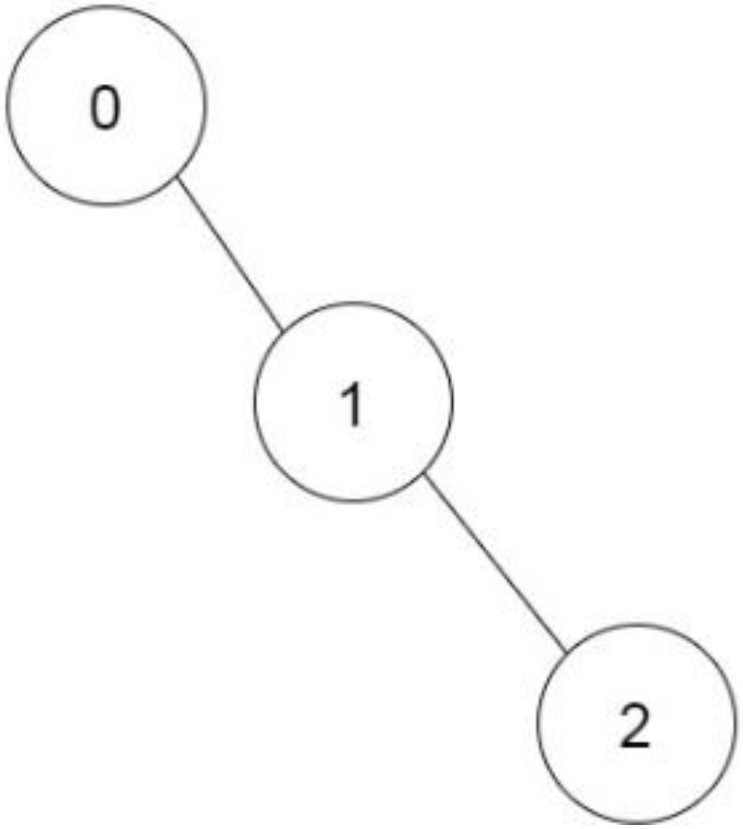
10

9

+ 7

.

Example 1:



Input:

prevRoom = [-1,0,1]

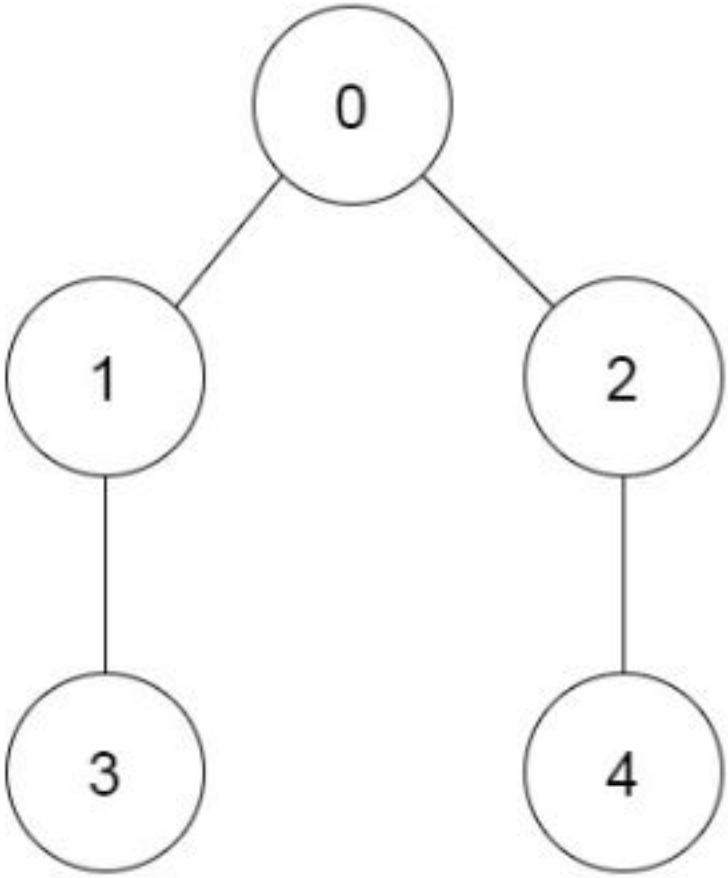
Output:

1

Explanation:

There is only one way to build the additional rooms: $0 \rightarrow 1 \rightarrow 2$

Example 2:



Input:

prevRoom = [-1,0,0,1,2]

Output:

6

Explanation:

The 6 ways are: 0 → 1 → 3 → 2 → 4 0 → 2 → 4 → 1 → 3 0 → 1 → 2 → 3 → 4 0 → 1 → 2 → 4 → 3 0 → 2 → 1 → 3 → 4 0 → 2 → 1 → 4 → 3

Constraints:

n == prevRoom.length

2 <= n <= 10

5

prevRoom[0] == -1

0 <= prevRoom[i] < n

for all

1 <= i < n

Every room is reachable from room

0

once all the rooms are built.

Code Snippets

C++:

```
class Solution {
public:
int waysToBuildRooms(vector<int>& prevRoom) {

}
};
```

Java:

```
class Solution {
public int waysToBuildRooms(int[] prevRoom) {

}
}
```

Python3:

```
class Solution:
def waysToBuildRooms(self, prevRoom: List[int]) -> int:
```

Python:

```
class Solution(object):
def waysToBuildRooms(self, prevRoom):
"""
:type prevRoom: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} prevRoom
* @return {number}
*/
var waysToBuildRooms = function(prevRoom) {

};
```

TypeScript:

```
function waysToBuildRooms(prevRoom: number[]): number {

};
```

C#:

```
public class Solution {
public int WaysToBuildRooms(int[] prevRoom) {

}
}
```

C:

```
int waysToBuildRooms(int* prevRoom, int prevRoomSize){

}
```

Go:

```
func waysToBuildRooms(prevRoom []int) int {
```

```
}
```

Kotlin:

```
class Solution {
fun waysToBuildRooms(prevRoom: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func waysToBuildRooms(_ prevRoom: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn ways_to_build_rooms(prev_room: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} prev_room
# @return {Integer}
def ways_to_build_rooms(prev_room)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $prevRoom
* @return Integer
*/
```

```
function waysToBuildRooms($prevRoom) {

}
}
```

Scala:

```
object Solution {
def waysToBuildRooms(prevRoom: Array[Int]): Int = {

}
}
```

Racket:

```
(define/contract (ways-to-build-rooms prevRoom)
(-> (listof exact-integer?) exact-integer?)

)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Ways to Build Rooms in an Ant Colony
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, sort
*
* 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 waysToBuildRooms(vector<int>& prevRoom) {

}
};
```

Java Solution:

```
/**
* Problem: Count Ways to Build Rooms in an Ant Colony
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, sort
*
* 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 waysToBuildRooms(int[] prevRoom) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Ways to Build Rooms in an Ant Colony
Difficulty: Hard
Tags: array, tree, graph, dp, math, sort

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 waysToBuildRooms(self, prevRoom: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def waysToBuildRooms(self, prevRoom):
"""
:type prevRoom: List[int]
:rtype: int
```

```
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Count Ways to Build Rooms in an Ant Colony
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, sort
*
* 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[]} prevRoom
* @return {number}
*/
var waysToBuildRooms = function(prevRoom) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Count Ways to Build Rooms in an Ant Colony
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, sort
*
* 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 waysToBuildRooms(prevRoom: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Ways to Build Rooms in an Ant Colony
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, sort
*
* 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 WaysToBuildRooms(int[] prevRoom) {

}
}
```

C Solution:

```
/*
* Problem: Count Ways to Build Rooms in an Ant Colony
* Difficulty: Hard
* Tags: array, tree, graph, dp, math, sort
*
* 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 waysToBuildRooms(int* prevRoom, int prevRoomSize){

}
```

Go Solution:

```
// Problem: Count Ways to Build Rooms in an Ant Colony
// Difficulty: Hard
// Tags: array, tree, graph, dp, math, sort
//
// 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 waysToBuildRooms(prevRoom []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun waysToBuildRooms(prevRoom: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func waysToBuildRooms(_ prevRoom: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Ways to Build Rooms in an Ant Colony
// Difficulty: Hard
// Tags: array, tree, graph, dp, math, sort
//
// 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_build_rooms(prev_room: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} prev_room
# @return {Integer}
def ways_to_build_rooms(prev_room)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $prevRoom
* @return Integer
*/
function waysToBuildRooms($prevRoom) {

}
}
```

Scala Solution:

```
object Solution {
def waysToBuildRooms(prevRoom: Array[Int]): Int = {

}
}
```

Racket Solution:

```
(define/contract (ways-to-build-rooms prevRoom)
(-> (listof exact-integer?) exact-integer?)

)
```