Problem 841: Keys and Rooms

Problem Information

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

Problem Description

There are

n

rooms labeled from

0

to

n - 1

and all the rooms are locked except for room

0

. Your goal is to visit all the rooms. However, you cannot enter a locked room without having its key.

When you visit a room, you may find a set of

distinct keys

in it. Each key has a number on it, denoting which room it unlocks, and you can take all of them with you to unlock the other rooms.

Given an array

rooms

where

rooms[i]

is the set of keys that you can obtain if you visited room

i

, return

true

if you can visit

all

the rooms, or

false

otherwise

.

Example 1:

Input:

rooms = [[1],[2],[3],[]]

Output:

true

Explanation:

We visit room 0 and pick up key 1. We then visit room 1 and pick up key 2. We then visit room 2 and pick up key 3. We then visit room 3. Since we were able to visit every room, we return true.

Example 2:

Input:

rooms = [[1,3],[3,0,1],[2],[0]]

Output:

false

Explanation:

We can not enter room number 2 since the only key that unlocks it is in that room.

Constraints:

n == rooms.length

2 <= n <= 1000

0 <= rooms[i].length <= 1000

1 <= sum(rooms[i].length) <= 3000

0 <= rooms[i][j] < n

All the values of

rooms[i]

are

unique

.

Code Snippets

C++:

```
class Solution {
public:
bool canVisitAllRooms(vector<vector<int>>& rooms) {

}
};
```

Java:

```
class Solution {
public boolean canVisitAllRooms(List<List<Integer>> rooms) {

}
}
```

Python3:

```
class Solution:
def canVisitAllRooms(self, rooms: List[List[int]]) -> bool:
```

Python:

```
class Solution(object):
def canVisitAllRooms(self, rooms):
"""
:type rooms: List[List[int]]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {number[][]} rooms
* @return {boolean}
*/
var canVisitAllRooms = function(rooms) {
```

```
};
```

TypeScript:

```
function canVisitAllRooms(rooms: number[][]): boolean {

};
```

C#:

```
public class Solution {
public bool CanVisitAllRooms(IList<IList<int>> rooms) {

}
}
```

C:

```
bool canVisitAllRooms(int** rooms, int roomsSize, int* roomsColSize) {

}
```

Go:

```
func canVisitAllRooms(rooms [][]int) bool {

}
```

Kotlin:

```
class Solution {
fun canVisitAllRooms(rooms: List<List<Int>>): Boolean {

}
}
```

Swift:

```
class Solution {
func canVisitAllRooms(_ rooms: [[Int]]) -> Bool {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn can_visit_all_rooms(rooms: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby:

```
# @param {Integer[][]} rooms
# @return {Boolean}
def can_visit_all_rooms(rooms)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $rooms
* @return Boolean
*/
function canVisitAllRooms($rooms) {

}
}
```

Dart:

```
class Solution {
bool canVisitAllRooms(List<List<int>> rooms) {

}
}
```

Scala:

```
object Solution {
def canVisitAllRooms(rooms: List[List[Int]]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_visit_all_rooms(rooms :: [[integer]]) :: boolean
def can_visit_all_rooms(rooms) do

end
end
```

Erlang:

```
-spec can_visit_all_rooms(Rooms :: [[integer()]]) -> boolean().
can_visit_all_rooms(Rooms) ->
.
```

Racket:

```
(define/contract (can-visit-all-rooms rooms)
(-> (listof (listof exact-integer?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Keys and Rooms
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
bool canVisitAllRooms(vector<vector<int>>& rooms) {

}
};
```

Java Solution:

```
/**
* Problem: Keys and Rooms
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean canVisitAllRooms(List<List<Integer>> rooms) {

}
}
```

Python3 Solution:

```
"""
Problem: Keys and Rooms
Difficulty: Medium
Tags: array, graph, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def canVisitAllRooms(self, rooms: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def canVisitAllRooms(self, rooms):
"""
:type rooms: List[List[int]]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Keys and Rooms
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} rooms
* @return {boolean}
*/
var canVisitAllRooms = function(rooms) {

};
```

TypeScript Solution:

```
/**
* Problem: Keys and Rooms
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function canVisitAllRooms(rooms: number[][]): boolean {
```

```
};
```

C# Solution:

```
/*
* Problem: Keys and Rooms
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool CanVisitAllRooms(IList<IList<int>> rooms) {

}
}
```

C Solution:

```
/*
* Problem: Keys and Rooms
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

bool canVisitAllRooms(int** rooms, int roomsSize, int* roomsColSize) {

}
```

Go Solution:

```
// Problem: Keys and Rooms
// Difficulty: Medium
```

```
// Tags: array, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func canVisitAllRooms(rooms [][]int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun canVisitAllRooms(rooms: List<List<Int>>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canVisitAllRooms(_ rooms: [[Int]]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Keys and Rooms
// Difficulty: Medium
// Tags: array, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn can_visit_all_rooms(rooms: Vec<Vec<i32>>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} rooms
# @return {Boolean}
def can_visit_all_rooms(rooms)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $rooms
* @return Boolean
*/
function canVisitAllRooms($rooms) {

}
}
```

Dart Solution:

```
class Solution {
bool canVisitAllRooms(List<List<int>> rooms) {

}
}
```

Scala Solution:

```
object Solution {
def canVisitAllRooms(rooms: List[List[Int]]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_visit_all_rooms(rooms :: [[integer]]) :: boolean
def can_visit_all_rooms(rooms) do
```

```
end
end
```

Erlang Solution:

```
-spec can_visit_all_rooms(Rooms :: [[integer()]]) -> boolean().
can_visit_all_rooms(Rooms) ->
.
```

Racket Solution:

```
(define/contract (can-visit-all-rooms rooms)
(-> (listof (listof exact-integer?)) boolean?)
)
```