Problem 3680: Generate Schedule

Problem Information

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

Problem Description

You are given an integer

n

representing

n

teams. You are asked to generate a schedule such that:

Each team plays every other team

exactly twice

: once at home and once away.

There is

exactly one

match per day; the schedule is a list of

consecutive

days and

schedule[i]

is the match on day

i

.

No team plays on

consecutive

days.

Return a 2D integer array

schedule

, where

schedule[i][0]

represents the home team and

schedule[i][1]

represents the away team. If multiple schedules meet the conditions, return

any

one of them.

If no schedule exists that meets the conditions, return an empty array.

Example 1:

Input:

n = 3

Output:

[]

Explanation:

Since each team plays every other team exactly twice, a total of 6 matches need to be played:

[0,1],[0,2],[1,2],[1,0],[2,0],[2,1]

.

It's not possible to create a schedule without at least one team playing consecutive days.

Example 2:

Input:

n = 5

Output:

[[0,1],[2,3],[0,4],[1,2],[3,4],[0,2],[1,3],[2,4],[0,3],[1,4],[2,0],[3,1],[4,0],[2,1],[4,3],[1,0],[3,2],[4,1],[3,0],[4,2]]

Explanation:

Since each team plays every other team exactly twice, a total of 20 matches need to be played.

The output shows one of the schedules that meet the conditions. No team plays on consecutive days.

Constraints:

2 <= n <= 50

Code Snippets

C++:

```cpp
class Solution {
public:
vector<vector<int>> generateSchedule(int n) {

}
};
```

Java:

```java
class Solution {
public int[][] generateSchedule(int n) {

}
}
```

Python3:

```python
class Solution:
def generateSchedule(self, n: int) -> List[List[int]]:
```

Python:

```python
class Solution(object):
def generateSchedule(self, n):
"""
:type n: int
:rtype: List[List[int]]
"""
```

JavaScript:

```javascript
/**
* @param {number} n
* @return {number[][]}
*/
var generateSchedule = function(n) {

};
```

TypeScript:

```
function generateSchedule(n: number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] GenerateSchedule(int n) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** generateSchedule(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func generateSchedule(n int) [][]int {

}
```

Kotlin:

```
class Solution {
fun generateSchedule(n: Int): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func generateSchedule(_ n: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn generate_schedule(n: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer[][]}
def generate_schedule(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer[][]
*/
function generateSchedule($n) {

}
}
```

Dart:

```
class Solution {
List<List<int>> generateSchedule(int n) {

}
}
```

Scala:

```
object Solution {
def generateSchedule(n: Int): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec generate_schedule(n :: integer) :: [[integer]]
def generate_schedule(n) do

end
end
```

Erlang:

```
-spec generate_schedule(N :: integer()) -> [[integer()]].
generate_schedule(N) ->
.
```

Racket:

```
(define/contract (generate-schedule n)
(-> exact-integer? (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Generate Schedule
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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:
vector<vector<int>> generateSchedule(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Generate Schedule
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 int[][] generateSchedule(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Generate Schedule
Difficulty: Medium
Tags: array, greedy, math

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 generateSchedule(self, n: int) -> List[List[int]]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Generate Schedule
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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} n
* @return {number[][]}
*/
var generateSchedule = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Generate Schedule
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 generateSchedule(n: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Generate Schedule
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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 int[][] GenerateSchedule(int n) {

}
}
```

C Solution:

```
/*
* Problem: Generate Schedule
* Difficulty: Medium
* Tags: array, greedy, math
*
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
```

```
*/
int** generateSchedule(int n, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Generate Schedule
// Difficulty: Medium
// Tags: array, greedy, math
//
// 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 generateSchedule(n int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun generateSchedule(n: Int): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func generateSchedule(_ n: Int) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Generate Schedule
// Difficulty: Medium
// Tags: array, greedy, math
//
```

```
// 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 generate_schedule(n: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer[][]}
def generate_schedule(n)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @return Integer[][]
*/
function generateSchedule($n) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> generateSchedule(int n) {

}
}
```

Scala Solution:

```
object Solution {
def generateSchedule(n: Int): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec generate_schedule(n :: integer) :: [[integer]]
def generate_schedule(n) do

end
end
```

Erlang Solution:

```
-spec generate_schedule(N :: integer()) -> [[integer()]].
generate_schedule(N) ->
.
```

Racket Solution:

```
(define/contract (generate-schedule n)
(-> exact-integer? (listof (listof exact-integer?)))
)
```