VMES Grinds

VMES Grinds

2580. Count Ways to Group Overlapping Ranges

2580. Count Ways to Group Overlapping Ranges

Solved

Medium | Topics | Companies | Hint

You are given a 2D integer array `ranges` where `ranges[i] = [start_i, end_i]` denotes that all integers between `start_i` and `end_i` (both **inclusive**) are contained in the `i^th` range.

You are to split `ranges` into **two** (possibly empty) groups such that:

- Each range belongs to exactly one group.
- Any two **overlapping** ranges must belong to the **same** group.

Two ranges are said to be **overlapping** if there exists at least **one** integer that is present in both ranges.

- For example, `[1, 3]` and `[2, 5]` are overlapping because `2` and `3` occur in both ranges.

Return *the **total number** of ways to split* `ranges` *into two groups*. Since the answer may be very large, return it **modulo** `10^9 + 7`.

Example 1:

> **Input:** ranges = [[6,10],[5,15]]
> **Output:** 2
> **Explanation:**
> The two ranges are overlapping, so they must be in the same group.
> Thus, there are two possible ways:
> - Put both the ranges together in group 1.
> - Put both the ranges together in group 2.

Example 2:

> **Input:** ranges = [[1,3],[10,20],[2,5],[4,8]]
> **Output:** 4
> **Explanation:**
> Ranges [1,3], and [2,5] are overlapping. So, they must be in the same group.
> Again, ranges [2,5] and [4,8] are also overlapping. So, they must also be in the same group.
> Thus, there are four possible ways to group them:
> - All the ranges in group 1.
> - All the ranges in group 2.
> - Ranges [1,3], [2,5], and [4,8] in group 1 and [10,20] in group 2.
> - Ranges [1,3], [2,5], and [4,8] in group 2 and [10,20] in group 1.

Constraints:

- $1 <= ranges.length <= 10^5$
- $ranges[i].length == 2$
- $0 <= start_i <= end_i <= 10^9$

Explanation

- Step 1: Sort the ranges by when they start (Like organizing tasks by start time)
- Step 2: Walk through them one by one:
  - Start a new "group" with the first range
  - If next range is in current group → add it to group
  - If next range doesn't → start a new group
- Step 3: Count how many groups
- Step 4: Answer = 2^(number of groups) (Each group can go to Group A or Group B)

Initial Approach

```python
class Solution(object):
    def countWays(self, ranges):
        MOD = 10**9 + 7
        ranges.sort()

        groups = 0

        i = 0
        n = len(ranges)
        while i < n:
            groups += 1
            end = ranges[i][1]
            i += 1
            while i < n and ranges[i][0] <= end:
                end = max(end, ranges[i][1])
                i += 1

        return pow(2, groups, MOD)
```

Optimized Approach

Same complexity (O(n log n)), but the for-loop + tuple unpacking cuts Python-level overhead and branches — lower constants => faster
Making it 31ms to 19ms

```python
class Solution:
    def countWays(self, ranges):
        MOD = 10**9 + 7
        ranges.sort()
        components = 0
        end = -1

        for start, finish in ranges:
            if start > end:
                components += 1
                end = finish
            else:
                end = max(end, finish)

        return pow(2, components, MOD)
```

VMES Grinds

Time and Space Complexity Analysis

```python
class Solution(object):
    def countWays(self, ranges):
        MOD = 10**9 + 7
        ranges.sort()

        groups = 0

        i = 0
        n = len(ranges)
        while i < n:
            groups += 1
            end = ranges[i][1]
            i += 1
            while i < n and ranges[i][0] <= end:
                end = max(end, ranges[i][1])
                i += 1

        return pow(2, groups, MOD)
```

Time - O(nlogn)

Time - O(n)

Space - O(1)

Time - O(n) + O(nlogn) = O(nlogn)

```python
class Solution:
    def countWays(self, ranges):
        MOD = 10**9 + 7
        ranges.sort()
        components = 0
        end = -1

        for start, finish in ranges:
            if start > end:
                components += 1
                end = finish
            else:
                end = max(end, finish)

        return pow(2, components, MOD)
```

Time - O(nlogn)

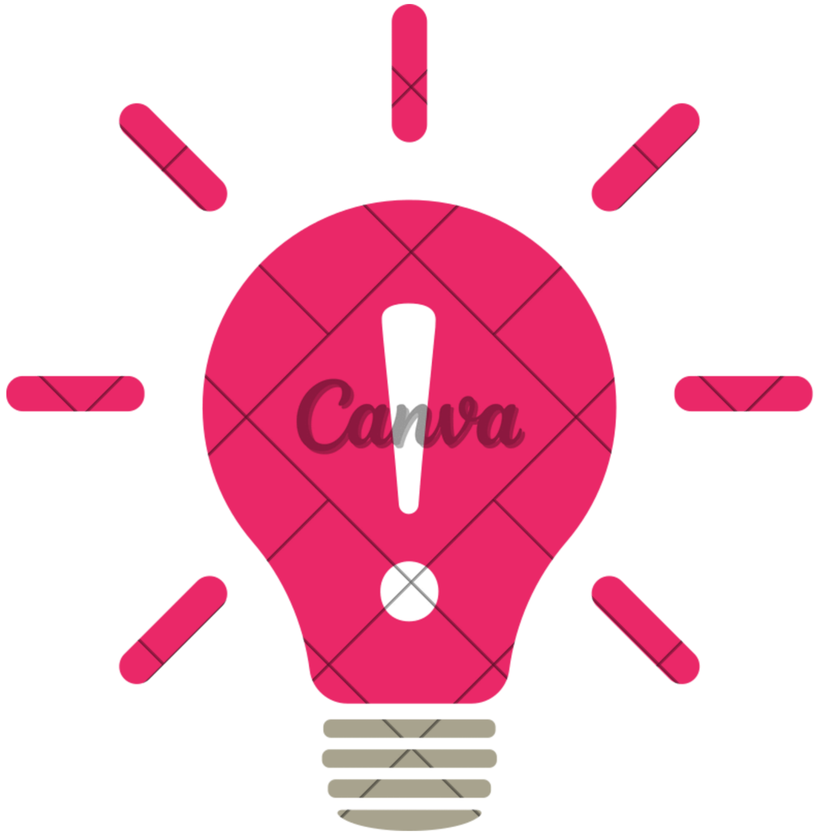
Time - O(n)

Space - O(1)

Time - O(n) + O(nlogn) = O(nlogn)

Key Takeaway

VMES Grinds



- Sort & Merge (Greedy): Sorting ranges and extending the current group’s end is a greedy strategy to efficiently form groups.
- Connected Components: Each non-overlapping cluster forms a separate group.
- Exponentiation: Total ways = 2^(number of groups) modulo 10^9 + 7