Description | Editorial | Solutions (3.7K) | Submissions

47. Permutations II

Medium | 7.7K | 134

Companies

Given a collection of numbers, `nums`, that might contain duplicates, return *all possible unique permutations* ***in any order***.

Example 1:

```
Input: nums = [1,1,2]
Output:
[[1,1,2],
 [1,2,1],
 [2,1,1]]
```

Example 2:

```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

Constraints:

- `1 <= nums.length <= 8`
- `-10 <= nums[i] <= 10`

Accepted **801K** | Submissions **1.4M** | Acceptance Rate **57.7%**

Seen this question in a real interview before? 1/4

Yes No

Java | Auto

```java
class Solution {
    public List<List<Integer>> permuteUnique(int[] nums) {
        List<List<Integer>> matrix = new ArrayList<>();
        List<Integer> row = new ArrayList<>();
        List<Integer> list = new ArrayList<>();
        for (int num : nums) {
            list.add(num);
        }
        permutelist(list, row, matrix);
        return matrix;
    }

    public static void permutelist(List<Integer> list, List<Integer> row, List<List<Integer>> matrix)
{
        if (list.size() == 0) {

            if(matrix.contains(row)){
                return;
            }
            matrix.add(new ArrayList<>(row));
```

Testcase | Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

nums =
[1,1,2]

Output

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

Expected

Console | Run | Submit

Description | Editorial | Solutions (522) | Submissions

2551. Put Marbles in Bags

Hint

Hard | 1.5K | 59

Companies

You have `k` bags. You are given a **0-indexed** integer array `weights` where `weights[i]` is the weight of the i^{th} marble. You are also given the integer `k`.

Divide the marbles into the `k` bags according to the following rules:

- No bag is empty.
- If the i^{th} marble and j^{th} marble are in a bag, then all marbles with an index between the i^{th} and j^{th} indices should also be in that same bag.
- If a bag consists of all the marbles with an index from `i` to `j` inclusively, then the cost of the bag is `weights[i] + weights[j]`.

The **score** after distributing the marbles is the sum of the costs of all the `k` bags.

Return *the **difference** between the **maximum** and **minimum** scores among marble distributions.*

Example 1:

```
Input: weights = [1,3,5,1], k = 2
Output: 4
Explanation:
The distribution [1],[3,5,1] results in the minimal score of (1+1) + (3+1) = 6.
The distribution [1,3],[5,1], results in the maximal score of (1+3) + (5+1) = 10.
Thus, we return their difference 10 - 6 = 4.
```

Example 2:

```
Input: weights = [1, 3], k = 2
```

Java | Auto

```java
class Solution {
    public long putMarbles(int[] weights, int k) {
        int n = weights.length;
        int[] pairs = new int[n - 1];
        for (int i = 1; i < n; i++) {
            pairs[i - 1] = weights[i] + weights[i - 1];
        }
        Arrays.sort(pairs);
        long minScore = 0;
        long maxScore = 0;
        for (int i = 0; i < k - 1; i++) {
            minScore += pairs[i];
            maxScore += pairs[n - i - 2];
        }
        return maxScore - minScore;
    }
}
```

Testcase | Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

weights =
[1,3,5,1]

k =
2

Output

Console | Run | Submit