L15 Permutations II

Saturday, December 24, 2022 7:26 PM

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]]

↳ In this Problem duplicates element exist then we can Not apply previously concept directly

→ In previously problem our given array is Unique so No need to considered any type of Special case for duplicates permutation

↳ Because In the unique array Not possible any duplicate Combinations and permutations

→ But In this problem the given Array contains duplicate element So duplicate permutations are also possible

↳ And our task is only print unique permutations, So How we do It ?

Thought Process

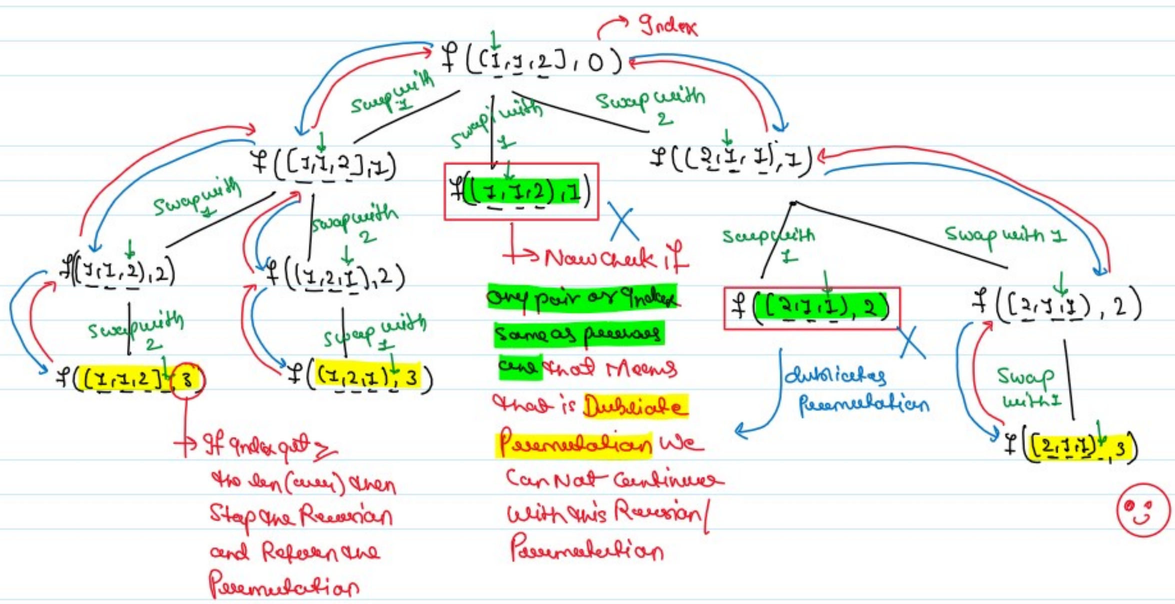
→ Basically We used the previous Concept here that is Swaping concept that we learn in Permutation I

↳ But only change the Some edge case Condition that is If
or
↳ used the Set data Structure for Unique permutation

→ Let's Understand Using the Recursive Tree and understand How the our Concept Work

arr = [1,1,2]
0 1 2

n=3

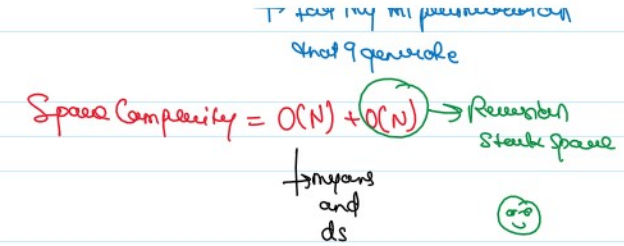


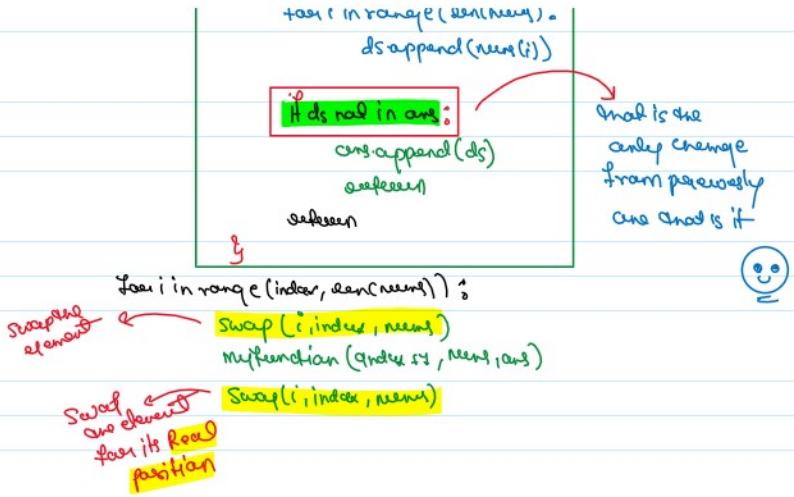
→ that is the Way How to Swaping Concept are Work

Time Complexity = (N!) x (N) → for loop 0 to N every time
↳ for my All permutation that I generate

```
myfunction(index, nums, ans){

  if index == len(nums){          → Base case
     ds = []
     for i in range(len(nums)):
         ds.append(nums[i])
```






```python
class Solution:

    def swap(self, i, index, nums):
        temp = nums[i]
        nums[i] = nums[index]
        nums[index] = temp

    def findAllPermutations(self, index, nums, ans):

        if index == len(nums):
            ds = []
            for i in range(len(nums)):
                ds.append(nums[i])

            if ds not in ans:
                ans.append(ds)
                return
            return

        for i in range(index, len(nums)):
            self.swap(i, index, nums)
            self.findAllPermutations(index + 1, nums, ans)
            self.swap(i, index, nums)

    def permuteUnique(self, nums: List[int]) -> List[List[int]]:

        ans = []
        self.findAllPermutations(0, nums, ans)

        print(ans)
        return ans

# more optimize code using set Data Structure
# Same function template for python

class Solution:
    def uniquePerms(self, nums, n):

        results = []

        def backtrack(pointer):
            if pointer == len(nums):
                results.append(nums[:])
                return

            seen = set()  # generate an empty set everytime the backtrack function is called
            for i in range(pointer, len(nums)):

                if nums[i] not in seen:  # go below > if nums[i] is not already in the seen

                    seen.add(nums[i])  # add nums[i] to the seen so if there is a duplicate, if condition not enter

                    nums[i], nums[pointer] = nums[pointer], nums[i]

                    backtrack(pointer + 1)

                    nums[i], nums[pointer] = nums[pointer], nums[i]

        backtrack(0)
        results.sort()
        return results
```

only change

```java
import java.util.*;
public class L15_Permutations_II {
    public static void main(String[] args) {
        System.out.println("L15 Permutations II");
    }
}

no usages
class Solution8{

    2 usages
    public void swap(int i, int index, int[] nums) {
        int temp = nums[i];
        nums[i] = nums[index];
        nums[index] = temp;
    }

    2 usages
    public void findAllPermutations(int index, List<List<Integer>> ans, int[] nums) {

        if (index == nums.length) {
            List<Integer> ds = new ArrayList<>();
            for (int i = 0; i < nums.length; i++) {
                ds.add(nums[i]);
            }
            if(!ans.contains(ds)){
                ans.add(new ArrayList<>(ds));
                return;
            }
//            ans.add(new ArrayList<>(ds));
            return;
        }

        for (int i = index; i < nums.length; i++) {
            swap(i, index, nums);
            findAllPermutations( index: index + 1, ans, nums);
            swap(i, index, nums);
        }
    }
    no usages
    public List<List<Integer>> permuteUnique(int[] nums) {
        List<List<Integer>> ans = new ArrayList<>();
        findAllPermutations( index: 0, ans, nums);
        return ans;

    }
}
```

only change