L14 Permutations I

Saturday, December 24, 2022 11:19 AM

Given an array nums of **distinct integers**, return ***all the possible permutations***. You can return the answer in **any order.**

Example 1:
Input: nums = [1,2,3]

Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]

↳ In this problem every element is unique

→ If these problem comes in the interview then triple of solution possible of this problem and all the solution are Recursive

Thought Process

→ lets understand the first Approach which take extra space and the Optimized that Approach

Important Point
The total Permutation of the any Given Array or String is n!

→ Where (N) is the length of the Array

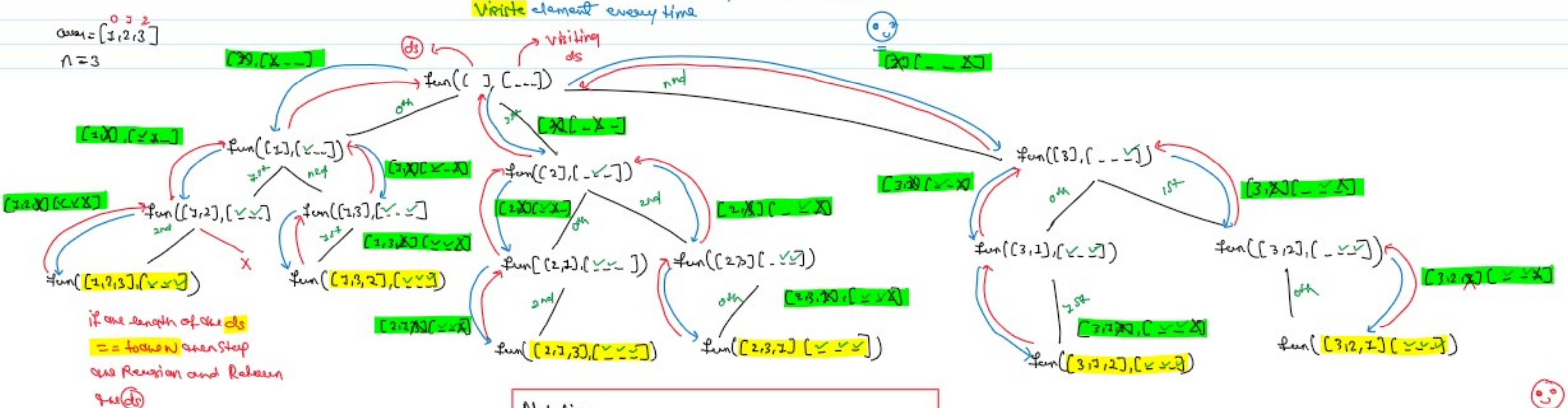
arr = [1,2,3]
n = 3
total permutation = n!
= 3!
= (6) possible permutation

→ Now the basically Draw the Recursion Tree that we do every time

→ We contain the empty ds for storing the All the permutation that are possible

→ Again we used the concept of pick and Not pick

→ One more data structure carry for marking the Visited element every time



Notation
🟨 → our Permutation Answer
🟩 → Remove the data for further Recursion

Important point
if the Recursion Achived the base case then Return and every Return pop element from ds and pop the element from the marked ds also

→ Because we want to Empty data structure for further Next Recursion that is why we every time pop the element from the both data structure

```
myfunction(nums, ans, visitedMap, ds){

    if len(ds) == len(nums){
        ans.append(ds)
        return;
    }                                   → Base case

    for i in range(len(nums)):
        if visitedMap[i] == false:
            visitedMap[i] = True        → marked visited
  pick ←    ds.add(nums[i])
            myfunction(nums, ans, visitedMap, ds);
  not pick ← ds.remove(nums[i])
            visitedMap[i] = False       → unmark for Next Recursion
}
```

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

Space Complexity = O(N) + O(N) + O(N)
→ O(N) for ans and ds
→ O(N) for Map Array
→ O(N) Recursion Stack Space

```python
class Solution:

    def findAllPermutations(self, nums, ans, visiteMap, ds):

        if len(ds) == len(nums):
            ans.append(ds[:])
            return
```

```java
import java.util.*;
public class L14_Permutations_I {
    public static void main(String[] args) {
        System.out.println("Permutations I");
    }
}

class Solution {
```

```python
def findAllPermutations(self, nums, ans, visiteMap, ds):

    if len(ds) == len(nums):
        ans.append(ds[:])
        return

    for i in range(len(nums)):

        if visiteMap[i] == False:
            visiteMap[i] = True
            ds.append(nums[i])
            self.findAllPermutations(nums, ans, visiteMap, ds)
            ds.remove(nums[i])
            visiteMap[i] = False

    print(visiteMap)

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

    ans = []
    visiteMap = [False] * len(nums)
    ds = []

    self.findAllPermutations(nums, ans, visiteMap, ds)

    return ans
```

```java
        System.out.println("Permutations I");
    }
}

class Solution6 {
    public List<List<Integer>> permute(int[] nums) {
        List<List<Integer>> ans = new ArrayList<>();
        List<Integer> ds = new ArrayList<>();
        boolean[] mapVisit = new boolean[nums.length];
        findAllPermutations(nums,ans,mapVisit,ds);
        return ans;

    }

    private void findAllPermutations(int[] nums, List<List<Integer>> ans, boolean[] mapVisit, List<Integer> ds) {

        if(ds.size() == nums.length){
            ans.add(new ArrayList<>(ds));
        }

        for(int i = 0;i < nums.length;i++){
            if(!mapVisit[i]){
                mapVisit[i] = true;
                ds.add(nums[i]);
                findAllPermutations(nums,ans,mapVisit,ds);
                ds.remove( index: ds.size()-1);
                mapVisit[i] = false;
            }
        }
    }
}
```

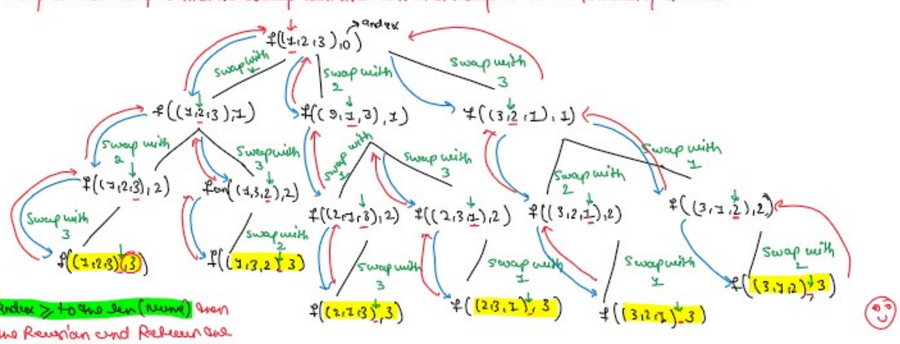
Approach 2

As we learn previously we used a extra data structure for storing the visiting element, In this Approach try to Reduced extra space and using the concept of swapping element

arr = [1, 2, 3]

n = 3

→ every element try to find the every combination and swap it to the Remaining element



Now index ≥ to the len(Nums) then stop the Recursion and Return the answer/any permutation

→ that is the way How to Swapping concept are work

Time Complexity = (N!) × (N) → for loop 0 to N every time

↳ for my All permutation that I generate

Space Complexity = O(N) + O(N) → Recursion Stack Space

↳ my ans and ds

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

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

        ans.add(ds)
        return
    }

    for i in range(index, len(nums)):
        swap(i, index, nums)            ← swapping element
        myfunction(index+1, nums, ans)
        swap(i, index, nums)            ← swap the element for its Real position
```

```python
# todo Optimized Aprroch Using Swap Concept
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])

            ans.append(ds)
            return
```

```java
//    todo Optimized Aprroch Using Swap Concept

class Solution7 {

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

    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;
            }
```

```python
            for i in range(len(nums)):
                ds.append(nums[i])

            ans.append(ds)
            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 permute(self, nums: List[int]) -> List[List[int]]:

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

        return ans
```

```java
                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>> permute(int[] nums) {
        List<List<Integer>> ans = new ArrayList<>();
        findAllPermutations( index: 0, ans, nums);
        return ans;

    }
```