47. Permutations II

Medium | Topics | 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:

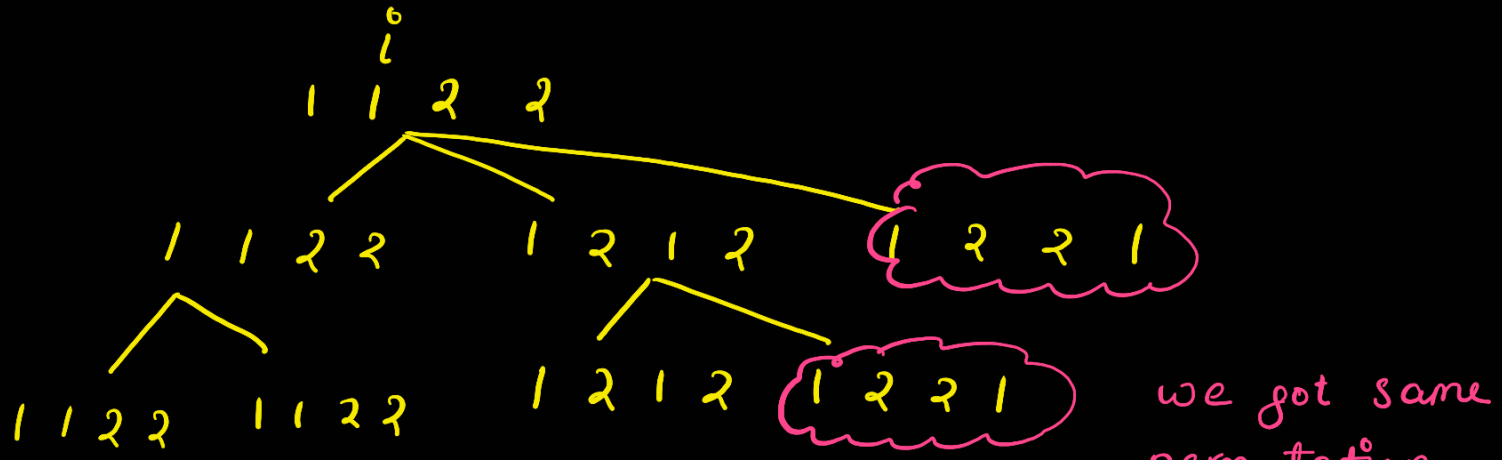
- `1 <= nums.length <= 8`
- `-10 <= nums[i] <= 10`

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

Yes | No

Accepted 857.1K | Submissions 1.5M | Acceptance Rate 58.3%

If you go by swapping approach, we may get duplicates because two different states can lead to same state in future after some swaps when there are duplicate elements.



permutation and there is no way to avoid this phenomenon in future with swapping approach

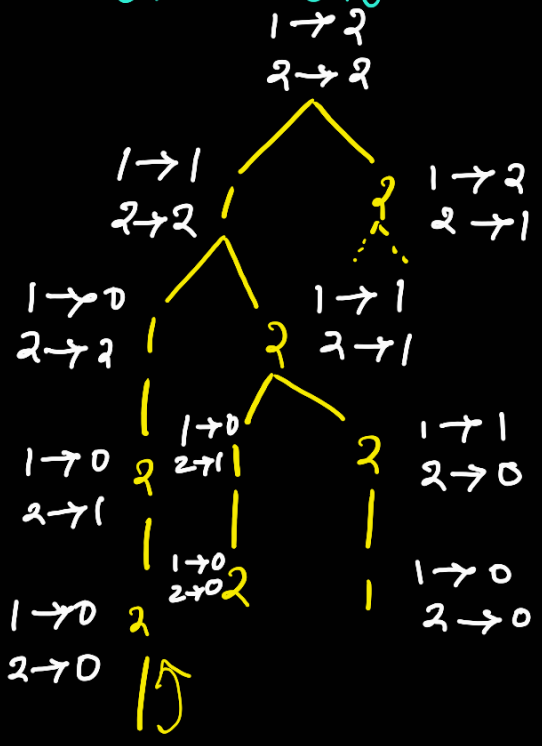
So instead of swapping approach, we use frequency approach. note: frequency approach can also be used when all elements are unique

element freq

1 → 2

2 → 2

at each level we consider only one occurence of an element




```
class Solution {
public:
    void find(unordered_map<int,int> &m,int n,vector<int> &currPer,vector<vector<int>> &ans){
        if(currPer.size() == n){
            ans.push_back(currPer);
            return;
        }

        for(auto it = m.begin();it != m.end();it++){
```

note: Range based for loop

```
        if(it->second != 0){
            it->second--;
            currPer.push_back(it->first);
            find(m,n,currPer,ans);
            it->second++;
            currPer.pop_back();
        }
    }
}

vector<vector<int>> permuteUnique(vector<int>& nums) {
    vector<vector<int>> ans;
    unordered_map<int,int> m;
    vector<int> currPer;

    for(auto i : nums)
        m[i]++;

    find(m,nums.size(),currPer,ans);

    return ans;
}
};
```

Note: Range based for loop cannot be used because arithmetic operations can't be Performed on iterator values when using range based "for" loops.

$T(n)$: $O(n \times n!)$

↓

for pushing vector into ans vector.