11. LC Subset Sum II

90. Subsets II

Medium 5687 164 Add to List Share

Given an integer array `nums` that may contain duplicates, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

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

① Brute force :

[1, 2, 2, 2, 3, 3]

In every index, we make decision to pick or not pick that element at that index. This will help us in generating all possible combination but doesn't take care of the duplicates. Hence we will use a set to store all the combinations that will discard the duplicates.

2^n → Set (To avoid duplicates) → list<list>

Set size will be $(n \times \log n)$

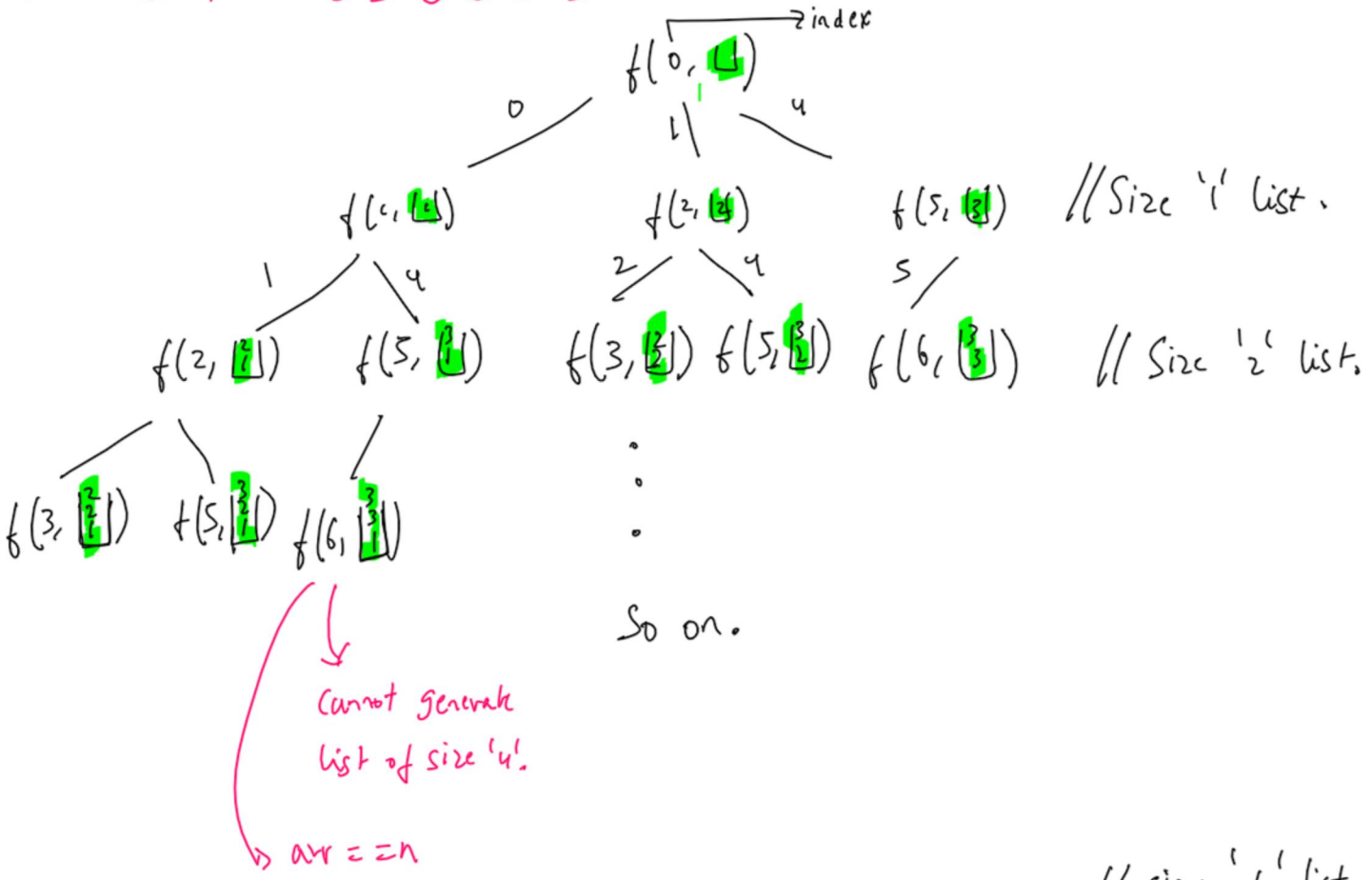
2^n.

② Optimal Approach :

0 1 2 3 4 5

[1, 2, 2, 2, 3, 3]

o/p → [], [1], [2] [3] [1,2] [1,3] [2,2] [2,3] [3,3] [1,2,2] [1,2,3] [1,3,3]



// size '6' list

↓

because arr size = 6.

S → Size

f(ind, ds)

ds.add(arr[i]) / i = ind − (n − 1)

f(i+1, ds)

↳ (S+1)

T.C ⇒ O(2^n)

T.C ⇒ $O(2^n)$ → to put in ans.

S.C ⇒ $O(2^n)\,O(k)$

Auxiliary Space ⇒ $O(n)$

Steps:

(1) Sort the i/p array

(2) Make a recursive function that take the i/p array, the current subset, current index and a list of list for Java, vector of vector for C++ to contain the answer.

(3) Make a subset of size 'n' during the n^{th} recursion call and consider elements from every index while generating the combinations. Only pick up elements that are appearing for the 1st time during a recursion call to avoid duplicates.

(4) Once the element is picked up, move to the next index, The recursion will terminate when the end of array is reached. While returning backtrack by removing the last element that was inserted.

Java • Autocomplete

```java
class Solution {
    public void findSubset(int ind, int[] nums, List<Integer> ds, List<List<Integer>>ans){
        ans.add(new ArrayList<>(ds));
        for(int i = ind; i < nums.length; i++){
            if(i != ind && nums[i] == nums[i-1]) continue;  // if it not the first and duplicate check
            ds.add(nums[i]);
            findSubset(i + 1, nums, ds, ans);
            ds.remove(ds.size() - 1);
        }
    }
```

```
    public List<List<Integer>> subsetsWithDup(int[] nums) {
        Arrays.sort(nums);  // in order to have duplicates side by side you need to sort it
        List<List<Integer>> ans = new ArrayList<>();
        findSubset(0, nums, new ArrayList<>(), ans);
        return ans;
    }
}
```

C++ • Autocomplete

```
class Solution {
private:
    void findSubset(int ind, vector<int> &nums, vector<int> &ds, vector<vector<int>> &ans){
        ans.push_back(ds);
         for(int i = ind; i < nums.size(); i++){
            if(i != ind && nums[i] == nums[i - 1]) continue;
            ds.push_back(nums[i]);
            findSubset(i + 1, nums, ds, ans);
            ds.pop_back();
         }
    }
public:
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        vector<vector<int>> ans;
        vector<int> ds;
        sort(nums.begin(), nums.end());
        findSubset(0, nums, ds, ans);
        return ans;
    }
};
```