L9 Combination Sum II

Tuesday, December 20, 2022 1:36 PM

Given a collection of candidate numbers (candidates) and a target number (target), find all unique combinations in candidates where the candidate numbers sum to target.
Each number in candidates may only be used **once** in the combination.

Note: The solution set must not contain duplicate combinations.

Example 1:
Input: candidates = [10,1,2,7,6,1,5], target = 8
Output:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]

→ Lexiographical and Sorted Order

↳ Basically in this problem Our task is print all the combination without Repetation and
→ Sorted Order
→ Lexiographical Order

Thought Process

arr = [1, 1, 1, 2, 2]
target = 4

[1, 1, 1, 2, 2]
✓ ✓ ✗ ✓ ✓
✓ ✗ ✓ ✓ ✓
_ _ _ ✓ ✓

Combinations
[1, 1, 2]
Not dublicate Appearence
[1, 1, 2] ✓
[2, 2] ✓
✓

[2, 1, 1] ✗
[1, 2, 1] ✗

element are Not in Sorted Order

→ Both are the Right Combination but According to this problem we only take that element that is in Sorted Order

Answer = 2

→ This Problem is the extended Version of previous problem that we Learned Combination Sum I

→ We Remember that in prevously we call the pick Recursion we can Not increase the index value because may be same other Combination exist

→ But here only one change is for unique Combination index+1 every time thats it.

myfunction (index, candidate, target - candidate[index], ans, ds); → In previously problem

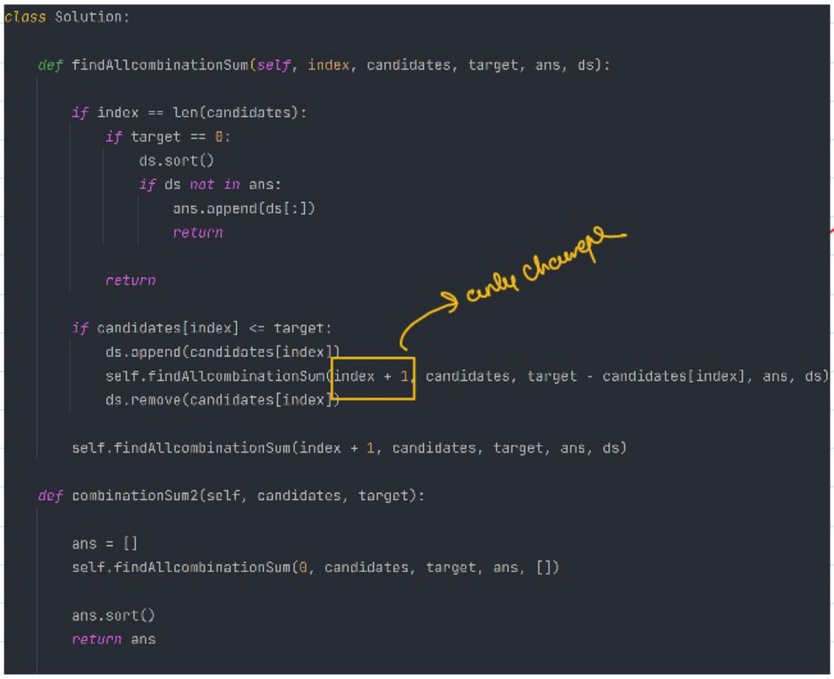
myfunction (index+1, candidate, target - candidate[index], ans, ds); → In this problem

→ used a set data structure for Avoide the dublicity
→ if we follow this Approach then the

$$\text{Time Complexity} = (2^T) \times K \times \log(f)$$

→ set size

→ for data structure put into the set data structure



```
class Solution:

    def findAllcombinationSum(self, index, candidates, target, ans, ds):

        if index == len(candidates):
            if target == 0:
                ds.sort()
                if ds not in ans:
                    ans.append(ds[:])
                    return

            return

        if candidates[index] <= target:
            ds.append(candidates[index])
            self.findAllcombinationSum(index + 1, candidates, target - candidates[index], ans, ds)
            ds.remove(candidates[index])

        self.findAllcombinationSum(index + 1, candidates, target, ans, ds)

    def combinationSum2(self, candidates, target):

        ans = []
        self.findAllcombinationSum(0, candidates, target, ans, [])

        ans.sort()
        return ans
```

→ Show TLE want to modified this Approach ↓

→ Now Interviewer do Not want to Extra log time into the Solution then we modify the our Solution (Modify the our Recursion)

Lets Understand

arr = [1, 1, 1, 2, 2] (indices 0 1 2 3 4)

target = 4

→ In this Approach In the place of **pick and Not pick** we fory the **pick the Subsequenced**

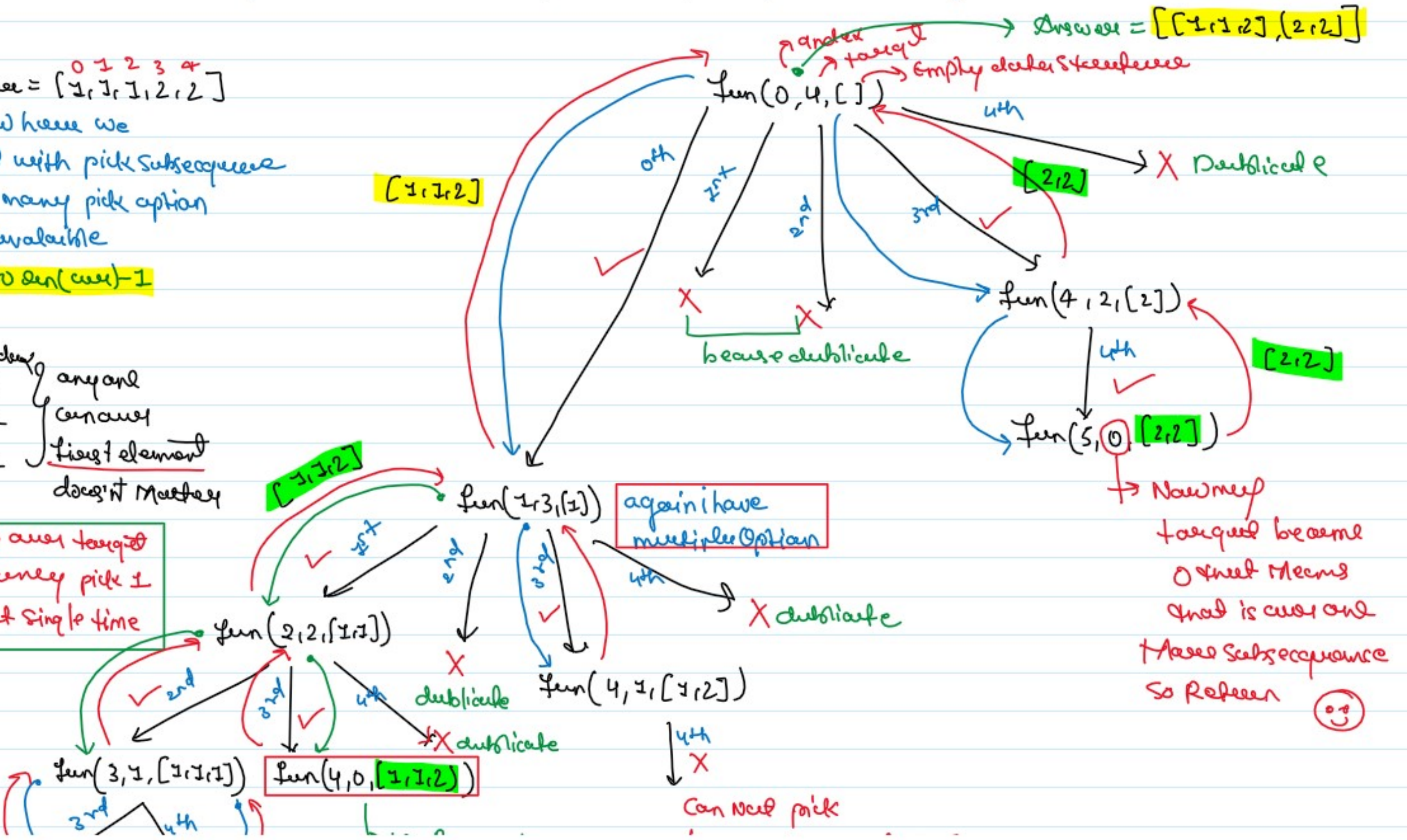
arr = [1, 1, 1, 2, 2] (indices 0 1 2 3 4)

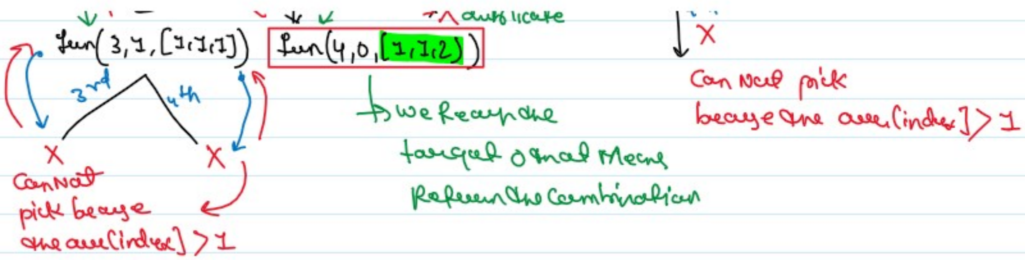
→ Now here we deal with pick Subsequence then many pick option are available

0 to len(arr)-1

→ Index 0, 1, 2 } anyone } answer } first element doesn't Matter

→ our target only pick 1 at single time




duplicate
fun(3,1,[1,1,1])
fun(4,0,[1,1,2])
3rd
4th
X
X
X
Can Not pick beaye the arr[index] > 1
we reach the target 0 that means Return the Combination
Can Not pick beaye the arr[index] > 1

→ that is the Approach that we follow for More Optimized that the over Recursion

```
myfunction(index, candidate, target, ans, ds){

    if(target == 0){
        ans.append(ds)
        return
    }

    for i in range (index, len(candidate)){
        if i > index and candidate[i] == candidate[i-1]{
            continue;
        }

        if candidate(i) > target{
            break;
        }

        ds.append(candidate(i));
        self.myfunction(i+1, candidate, target-candidate[i], ans, ds);
        ds.remove(candidate(i));

    }
}
```

→ Base case

Time Complexity = $2^n \times K$

→ Number of Combination

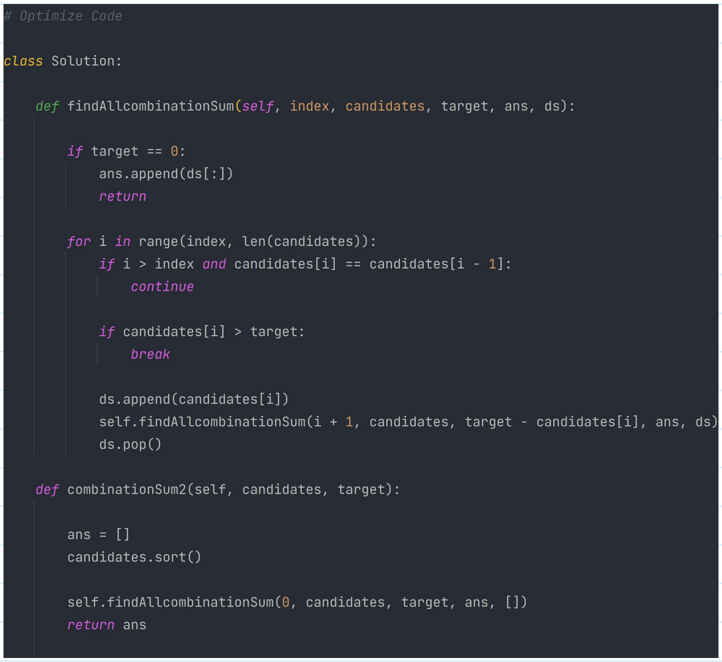
→ for put into the ds every time

Space Complexity = $K \times x \times N$

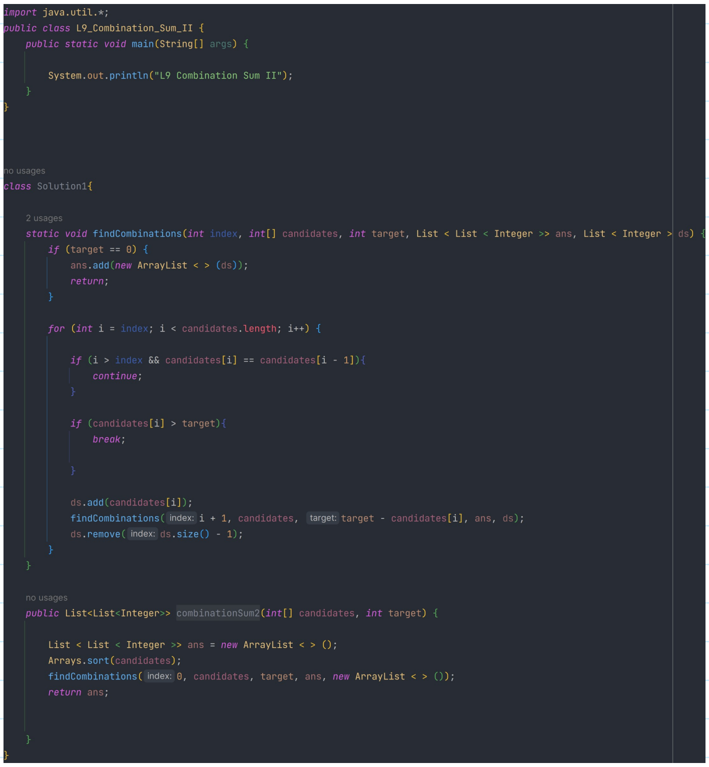
→ average length of the every Subsequence

→ for Combination

→ Recursion Stack Spaced



```python
# Optimize Code

class Solution:

    def findAllcombinationSum(self, index, candidates, target, ans, ds):

        if target == 0:
            ans.append(ds[:])
            return

        for i in range(index, len(candidates)):
            if i > index and candidates[i] == candidates[i - 1]:
                continue

            if candidates[i] > target:
                break

            ds.append(candidates[i])
            self.findAllcombinationSum(i + 1, candidates, target - candidates[i], ans, ds)
            ds.pop()

    def combinationSum2(self, candidates, target):

        ans = []
        candidates.sort()

        self.findAllcombinationSum(0, candidates, target, ans, [])
        return ans
```



```java
import java.util.*;
public class L9_Combination_Sum_II {
    public static void main(String[] args) {

        System.out.println("L9 Combination Sum II");
    }
}



no usages
class Solution1{

    2 usages
    static void findCombinations(int index, int[] candidates, int target, List < List < Integer >> ans, List < Integer > ds) {
        if (target == 0) {
            ans.add(new ArrayList < > (ds));
            return;
        }

        for (int i = index; i < candidates.length; i++) {

            if (i > index && candidates[i] == candidates[i - 1]){
                continue;
            }

            if (candidates[i] > target){
                break;

            }

            ds.add(candidates[i]);
            findCombinations( index: i + 1, candidates,  target: target - candidates[i], ans, ds);
            ds.remove( index: ds.size() - 1);
        }
    }

    no usages
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {

        List < List < Integer >> ans = new ArrayList < > ();
        Arrays.sort(candidates);
        findCombinations( index: 0, candidates, target, ans, new ArrayList < > ());
        return ans;


    }
}
```