L10 Combination Sum III

Tuesday, December 20, 2022 8:12 PM

Find all valid combinations of k numbers that sum up to n such that the following conditions are true:

- Only numbers 1 through 9 are used.
- Each number is used at most once.

Return *a list of all possible valid combinations*. The list must not contain the same combination twice, and the combinations may be returned in any order.

Example 1:
Input: k = 3, n = 7
Output: [[1,2,4]]
Explanation:
1 + 2 + 4 = 7
There are no other valid combinations.

↳ Basically this problem is the extended version of the printing All the combination that we learned previously

→ In Printing All the combination we learned print all possible unique combination, same concept used here only with some steps of modification

→ In this problem only change is

↳ we print only that combination that sum == N (target) and that length == K (every length) that's it only that change apply and Rest code will be same

↳ and we create (visit) Array and one element use only one time in a particular combination

Let's Understand

N = 3
target = 7
arr = [1,2,3,4,5,6,7,8,9]

Pick and Not Pick

1	2	3	4	5	6	7	8	9	
✓	✓	X	✓	X	X	X	X	X	= [1,2,4] ✓ Sum == target and len == n ✓
X	✓	X	X	✓	X	X	X	X	= [2,5]
✓	X	X	X	X	✓	X	X	X	= [1,6]
X	X	✓	X	X	X	✓	X	X	= [7]
X	✓	X	X	✓	X	✓	X	X	= [2,5]
X	X	✓	✓	X	X	X	X	X	= [3,4]

X because the sum is == target ✓

↳ Even the sum is not equal to N X

→ So this particular question arr answer is [1,2,4] only

```
myfunction(index, candidate, target, ans, ds, k) {
    if(index == len(candidate)) {
        if(target == 0 and sum(ds) == k) {
            ans.append(ds)
            return;
        }
        return;
    }
    ds.append(candidate[index]);
    myfunction(index+1, candidate, target - candidate[index], ans, ds, k);
    ds.removed(candidate[index]);
    myfunction(index+1, candidate, target - candidate[index], ans, ds, k);
```

(what is the only change)

(Base case)

(Pick case)

(Not pick)

case

ds.removed (candidate[index]);

Not pick case ← recursion (index+1, candidate, target - candidate[index], ans, ds, k);

}

→ Recursion Tree, and Recursion code working are same as we learn in previous chats only we not discuss again here

Time Complexity = ?

→ for every recursion we have tuple of option

take or Not take = 2^n (Exponential)

$O(2^n)$

Space complexity = $O(N) + O(N) \times O(K) \approx O(N + N \times K)$

→ for every to store the element

→ because we store the every element that is our subsequences

→ for putting the one data structure into another data structure

K → Size of every combination

```python
class Solution:

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

        if index == len(candidates):
            if target == 0 and len(ds) == k:
                ans.append(ds[:])
                return
            return

        ds.append(candidates[index])
        self.findAllcombinationSum(index + 1, candidates, target - candidates[index], ans, ds, k)
        ds.remove(candidates[index])

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

    def combinationSum3(self, k: int, target: int) -> List[List[int]]:

        candidates = [1, 2, 3, 4, 5, 6, 7, 8, 9]
        ans = []
        self.findAllcombinationSum(0, candidates, target, ans, [], k)

        return ans
```

```java
import java.util.*;
public class L10_Combination_Sum_III {
    public static void main(String[] args) { System.out.println("Combination Sum III"); }
}


class Solution2{
    3 usages
    private void findCombinations(int index, int[] candidates, int target, List < List < Integer >> ans, List < Integer > ds,int k) {
        if (index == candidates.length) {
            if (target == 0 && ds.size() == k) {
                ans.add(new ArrayList <> (ds));
            }
            return;
        }

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

    no usages
    public List<List<Integer>> combinationSum3(int k, int target) {

        List < List < Integer >> ans = new ArrayList < > ();
        int[] candidates = {1,2,3,4,5,6,7,8,9};
        findCombinations( index: 0, candidates, target, ans, new ArrayList < > (),k);

        return ans;
    }
}
```