39. Combination Sum

01 February 2022 07:52 PM

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all* ***unique combinations*** *of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

It is **guaranteed** that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

Example 2:

```
Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]
```

Example 3:

```
Input: candidates = [2], target = 1
Output: []
```

arr[] = {2, 3, 6, 7} target = 7

Combination, subseq ⇒ Recursion.

arr[] = {2, 3, 6, 7} target = 7

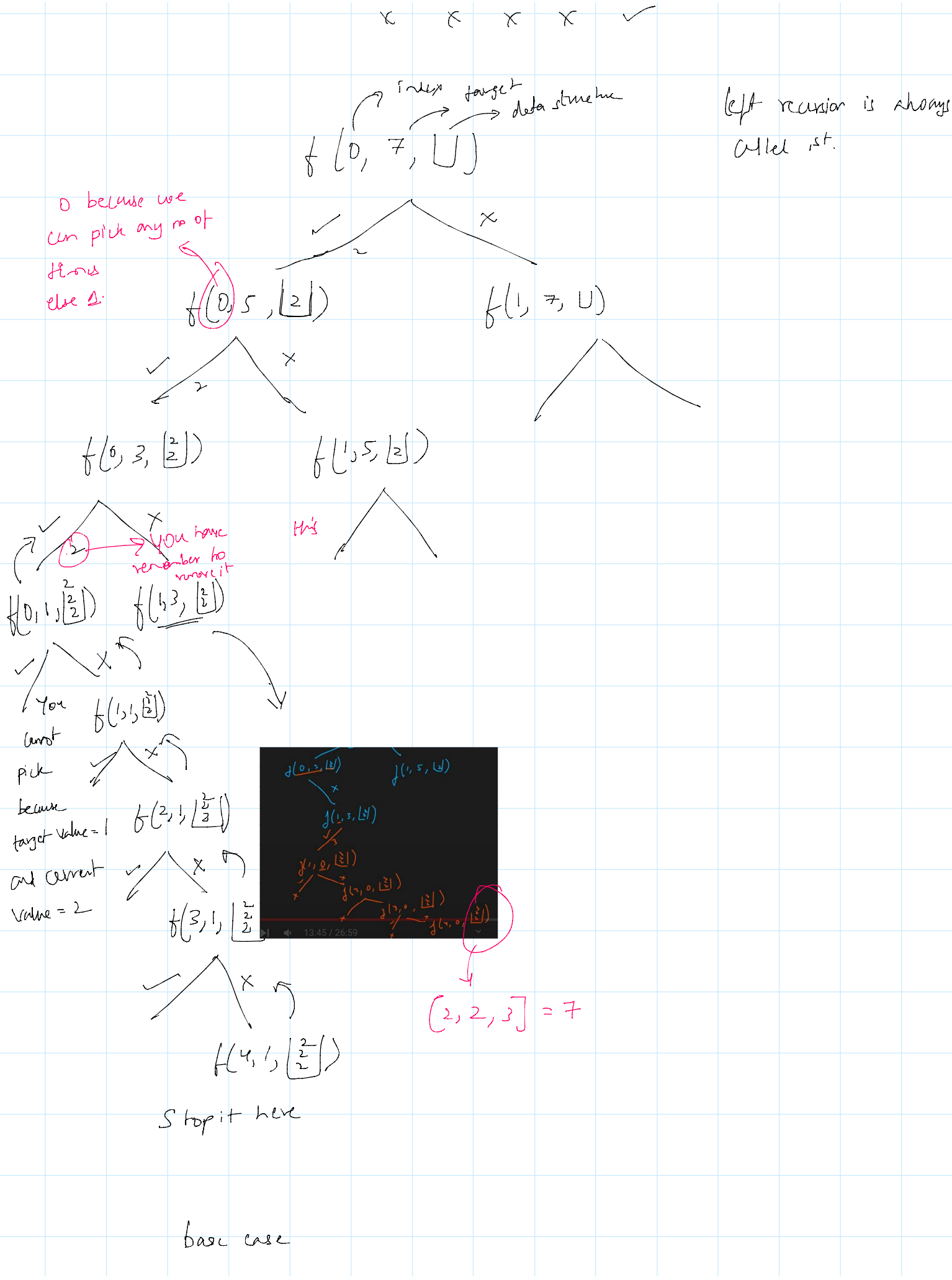
2, 2, 3 ⇒ 7

	PP	P	NP	NP
index →	0	1	2	3
	✓	✓	X	X

P → pick

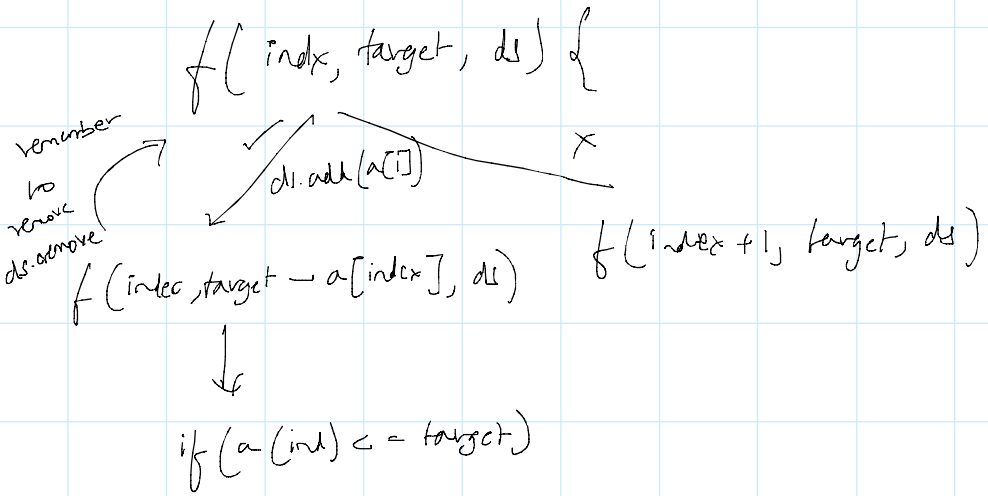
NP → not pick

	NP	NP	NP	NP	P
7	0	1	2	3	4 → index


index
target
data structure
f(0, 7, U)
left recursion is always called 1st.
0 because we can pick any no of items else 1.
f(0, 5, [2])
f(1, 7, U)
f(0, 3, [2 2])
f(1, 5, [2])
this
You have to remember to unmark it
f(0, 1, [2 2])
f(1, 3, [2 1])
You cant pick because target value = 1 and current value = 2
f(1, 1, [2 1])
f(2, 1, [2 2])
f(3, 1, [2 2 2])
f(4, 1, [2 2 2])
Stop it here
[2, 2, 3] = 7
basc case

ind == n

target == 0



Base

if (ind =)

if a + = 0)

else

ter ;

T. ⇒ 2^k

```
class Solution {

    private void findCombination(int ind, int[] arr, int target, List<List<Integer>>ans,
List<Integer>ds ){

        if(ind == arr.length){
            if(target == 0){
                ans.add(new ArrayList<>(ds));
            }
        return;
    }

        //pick condition
        if(arr[ind] <= target){
            ds.add(arr[ind]);
            //pick
            findCombination(ind, arr, target - arr[ind], ans, ds);
            ds.remove(ds.size() - 1);
        }
        // not pick
        findCombination(ind + 1, arr, target, ans, ds);
    }
    public List<List<Integer>> combinationSum(int[] candidates, int target) {
        List<List<Integer>> ans = new ArrayList<>();
        findCombination(0, candidates, target, ans, new ArrayList<>());
        return ans;
    }
}
```