<https://leetcode.com/problems/combination-sum/>

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.

The test cases are generated such 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: []

**Constraints:**

1 <= candidates.length <= 30

2 <= candidates[i] <= 40

All elements of candidates are **distinct**.

1 <= target <= 40

**Attempt 1: 2022-10-14**

**Solution 1: Backtracking style 1 (10min)**

class Solution {

    public List<List<Integer>> combinationSum(int[] candidates, int target) {

        List<List<Integer>> result = new ArrayList<List<Integer>>();

        helper(target, candidates, result, new ArrayList<Integer>(), 0);

        return result;

    }

    private void helper(int target, int[] candidates, List<List<Integer>> result, List<Integer> tmp, int index) {

        if(target == 0) {

            result.add(new ArrayList<Integer>(tmp));

            return;

        }

        if(target < 0) {

            return;

        }

        // Use local variable 'i' start from 'index' to traverse

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

            // Not pick --> no need "No pick" branch in for loop style, why ?

            //helper(target, candidates, result, tmp, i);

            // Pick

            tmp.add(candidates[i]);

            // Since we can use same number unlimited times, next recursion

            // still start with 'i'

            helper(target - candidates[i], candidates, result, tmp, i);

            tmp.remove(tmp.size() - 1);

        }

    }

}

Time Complexity: O(2^k) where k is the sum of target/candidate[i] from i = 0 to size of candidate - 1

Space Complexity: O(length\_of\_longest\_combination)

**Progress of Backtracking style 1**

target=7,candidates={2,3,6,7},result={},tmp={},index=0

--------------------------------------

Round 1:

helper(7,{2,3,6,7},{},{},0)

target=7

i=index=0

tmp.add(candidates[0])={2}

--------------------------------------

Round 2:

helper(7-2,{2,3,6,7},{},{2},0)

target=5

i=index=0

tmp.add(candidates[0])={2,2}

--------------------------------------

Round 3:

helper(5-2,{2,3,6,7},{},{2,2},0)

target=3

i=index=0

tmp.add(candidates[0])={2,2,2}

--------------------------------------

Round 4:

helper(3-2,{2,3,6,7},{},{2,2,2},0)

target=1

i=index=0

tmp.add(candidates[0])={2,2,2,2}

--------------------------------------

Round 5:

helper(1-2,{2,3,6,7},{},{2,2,2,2},0)

target=-1 < 0 return to Round 4 statistics

tmp.remove(4-1)={2,2,2}

for loop i++=1

tmp.add(candidates[1])={2,2,2,3}

--------------------------------------

Round 6:

helper(1-3,{2,3,6,7},{},{2,2,2,3},1)

target=-2 < 0 return to Round 4 statistics

tmp.remove(4-1)={2,2,2}

for loop i++=2

tmp.add(candidates[2])={2,2,2,6}

--------------------------------------

Round 7:

helper(1-6,{2,3,6,7},{},{2,2,2,6},2)

target=-5 < 0 return to Round 4 statistics

tmp.remove(4-1)={2,2,2}

for loop i++=3

tmp.add(candidates[1])={2,2,2,7}

--------------------------------------

Round 8:

helper(1-7,{2,3,6,7},{},{2,2,2,7},3)

target=-6 < 0 return to Round 4 statistics

tmp.remove(4-1)={2,2,2}

for loop i++=4 -> for loop ending in Round 4

tmp.remove(3-1)={2,2}

for loop i++=1

tmp.add(candidates[1])={2,2,3}

--------------------------------------

Round 9:

helper(3-3,{2,3,6,7},{},{2,2,3},1)

target=0 == 0 -> result={{2,2,3}}

return to Round 8 statistics

tmp.remove(3-1)={2,2}

for loop i++=2

tmp.add(candidates[2])={2,2,6}

--------------------------------------

Round 10:

helper(3-6,{2,3,6,7},{{2,2,3}},{2,2,6},2)

target=-3 < 0 return to Round 8 statistics

tmp.remove(3-1)={2,2}

for loop i++=3

tmp.add(candidates[3])={2,2,7}

--------------------------------------

Round 11:

helper(3-7,{2,3,6,7},{{2,2,3}},{2,2,7},3)

target=-4 < 0 return to Round 8 statistics

tmp.remove(3-1)={2,2}

for loop i++=4 -> for loop ending in Round 4

tmp.remove(2-1)={2}

for loop i++=1

tmp.add(candidates[1])={2,3}

--------------------------------------

Round 12:

helper(5-3,{2,3,6,7},{{2,2,3}},{2,3},1)

tmp.add(candidates[1])={2,3,3}

--------------------------------------

Round 13:

helper(2-3,{2,3,6,7},{{2,2,3}},{2,3,3},1)

target=-1 < 0 return to Round 8 statistics

tmp.remove(3-1)={2,3}

for loop i++=2

tmp.add(candidates[2])={2,3,6}

--------------------------------------

Round 14:

helper(2-6,{2,3,6,7},{{2,2,3}},{2,3,6},2)

target=-4 < 0 return to Round 8 statistics

tmp.remove(3-1)={2,3}

for loop i++=3

tmp.add(candidates[2])={2,3,7}

--------------------------------------

Round 15:

helper(2-7,{2,3,6,7},{{2,2,3}},{2,3,7},3)

target=-5 < 0 return to Round 8 statistics

tmp.remove(3-1)={2,3}

for loop i++=4 -> for loop ending in Round 4

tmp.remove(2-1)={2}

for loop i++=2

tmp.add(candidates[2])={2,6}

--------------------------------------

Round 16:

helper(5-6,{2,3,6,7},{{2,2,3}},{2,6},2)

target=-1 < 0 return to Round 4 statistics

tmp.remove(2-1)={2}

for loop i++=3

tmp.add(candidates[3])={2,7}

--------------------------------------

Round 17:

helper(5-7,{2,3,6,7},{{2,2,3}},{2,7},3)

target=-2 < 0 return to Round 4 statistics

tmp.remove(2-1)={2}

for loop i++=4 -> for loop ending in Round 4

tmp.remove(1-1)={}

for loop i++=1

tmp.add(candidates[1])={3}

--------------------------------------

Round 18:

helper(7-3,{2,3,6,7},{{2,2,3}},{3},1)

i=index=1

tmp.add(candidates[1])={3,3}

--------------------------------------

Round 19:

helper(4-3,{2,3,6,7},{{2,2,3}},{3,3},1)

i=index=1

tmp.add(candidates[1])={3,3,3}

--------------------------------------

Round 20:

helper(1-3,{2,3,6,7},{{2,2,3}},{3,3,3},2)

return to Round 19 statistics

tmp.remove(3-1)={3,3}

for loop i++=2

tmp.add(candidates[2])={3,3,6}

--------------------------------------

Round 21:

helper(1-6,{2,3,6,7},{{2,2,3}},{3,3,6},2)

target=-5 < 0 return to Round 18 statistics

tmp.remove(3-1)={3,3}

for loop i++=3

tmp.add(candidates[3])={3,3,7}

--------------------------------------

Round 22:

helper(1-7,{2,3,6,7},{{2,2,3}},{3,3,7},3)

target=-6 < 0 return to Round 18 statistics

tmp.remove(3-1)={3,3}

for loop i++=4 -> for loop ending in Round 18

tmp.remove(2-1)={3}

for loop i++=2

tmp.add(candidates[2])={3,6}

--------------------------------------

Round 23:

helper(4-6,{2,3,6,7},{{2,2,3}},{3,6},2)

target=-2 < 0 return to Round 18 statistics

tmp.remove(2-1)={3}

for loop i++=3

tmp.add(candidates[3])={3,7}

--------------------------------------

Round 24:

helper(4-7,{2,3,6,7},{{2,2,3}},{3,7},3)

target=-3 < 0 return to Round 18 statistics

tmp.remove(2-1)={3}

for loop i++=4 -> for loop ending in Round 4

tmp.remove(1-1)={}

for loop i++=2

tmp.add(candidates[2])={6}

--------------------------------------

Round 25:

helper(7-6,{2,3,6,7},{{2,2,3}},{6},2)

i=index=2

tmp.add(candidates[2])={6,6}

--------------------------------------

Round 26:

helper(1-6,{2,3,6,7},{{2,2,3}},{6,6},2)

target=-5 < 0 return to Round 25 statistics

tmp.remove(2-1)={6}

for loop i++=3

tmp.add(candidates[3])={6,7}

--------------------------------------

Round 27:

helper(1-7,{2,3,6,7},{{2,2,3}},{6,7},3)

target=-6 < 0 return to Round 25 statistics

tmp.remove(2-1)={6}

for loop i++=4 -> for loop ending in Round 25

tmp.remove(1-1)={}

for loop i++=3

tmp.add(candidates[3])={7}

--------------------------------------

Round 28:

helper(7-7,{2,3,6,7},{{2,2,3}},{7},3)

target=0 == 0 -> result={{2,2,3}{7}}

return

tmp.remove(1-1)={}

for loop i++=4 -> for loop ending in Round 28

======================================

End

======================================

Result time elapsed statistics:

result={{2,2,3}}

result={{2,2,3}{7}}

**Backtracking style 1 recursion step by step**

candidates={2,3,6,7} target=7

                                                    {}                            all combination of length=0

                                          /                        |        |        \

                                      {2}                        {3}      {6}      {7}          length=1

        /                      |        |    \          /  |  \      |  \ sum=target

              {2,2}                  {2,3}    {2,6} {2,7}  {3,3}{3,6}{3,7} {6,6}{6,7}              length=2

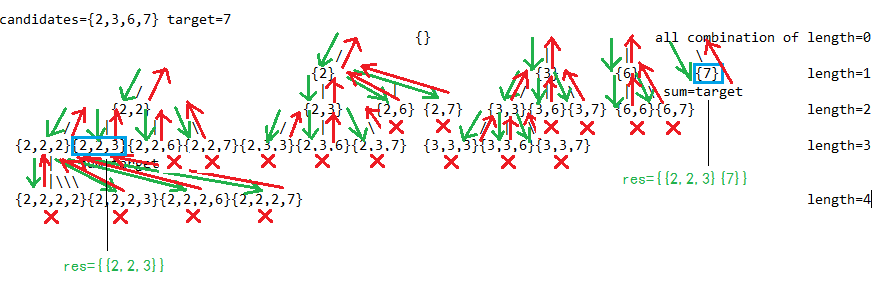
        /    |    |    \          /    |    \            /  |  \

  {2,2,2}{2,2,3}{2,2,6}{2,2,7}{2,3,3}{2,3,6}{2,3,7}  {3,3,3}{3,3,6}{3,3,7}                          length=3

      |  sum=target

      |\\\

  {2,2,2,2}{2,2,2,3}{2,2,2,6}{2,2,2,7}                                                              length=4



**Refer to**

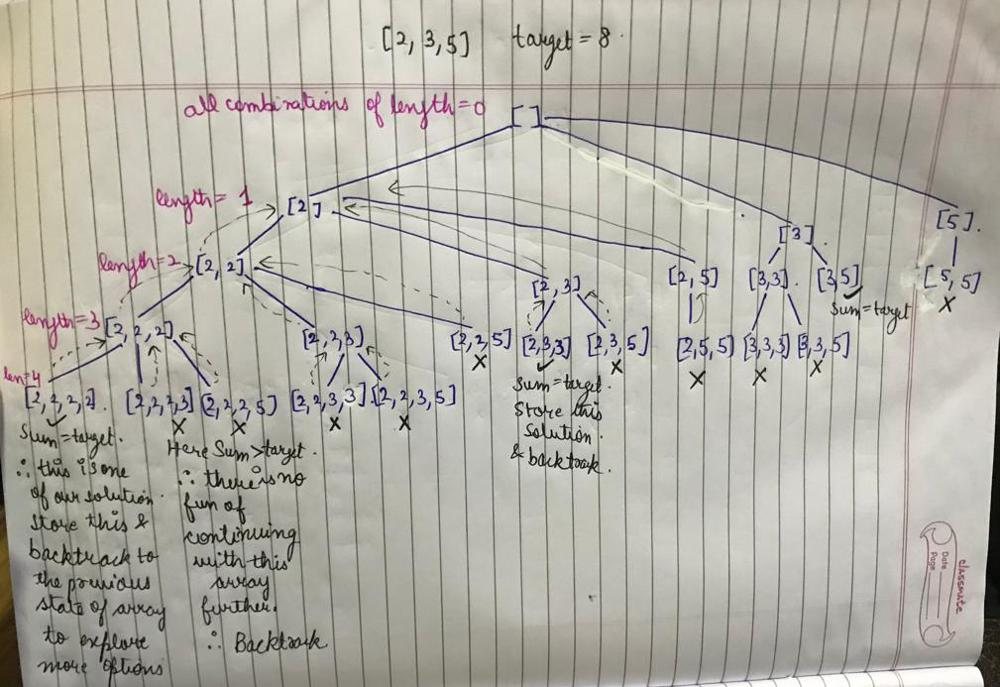
<https://leetcode.com/problems/combination-sum/discuss/1777569/FULL-EXPLANATION-WITH-STATE-SPACE-TREE-oror-Recursion-and-Backtracking-oror-Well-Explained-oror-C%2B%2B>

Reading the question we understand that we need every possible unique combination such that sum of the combination is equal to target.

For such questions where we have to find **"every possible"** we generally use **backtracking**.

Let's understand how can backtracking help us achieve what we want here. Below is an example with a **state space tree** for better understanding of how backtracking will be used in the solution.

**Example (State Space Tree):**



As you might have noticed that we are not considering the whole array as possible options at every level, because we want **unique combinations**.

e.g. at the node **[2, 2, 3]**, the *possible options* for the next level are only *[3, 5]*. Why?? because if we consider the whole array as possible options, then we will end up with [2, 2, 3, 2] (with 2 as a possible option), which has already been checked as [2, 2, 2, 3] (see the tree). So to *make the solution unique* we are only considering the *part of the array from current last element to the end element* (like in this example).

**Algorithm:**

1. If the sum of the current combination is greater than target, then even if we move forward with this combination, the sum will only increase, so there is no fun to moving further with such a combination as we can never achieve the target sum from this. So backtrack from this.

2. If the sum of the current combination is equal to the target, then we have a solution, so store this combination in the answers. Now moving forward with this combnation also will only increase the sum and we can't achieve the target sum again from this ever. So backtrack from here.

3. if we are here then that means the sum of the combination is still less that the target sum, and we have a scope of finding a combination whose sum can be equal to the target.

i) Now consider all possible options into this combination, one at a time.

ii) Go check if considering the current option can give us the solution.

iii) Now when this option backtracks to this place again, now remove this option and try another option. e.g. at [2, 2, \_ ] we have 3 options to fill the 3rd place i.e. [2, 3, 5].

So firstly we will go on with [2, 2, 2]. Then when this backtracks to this place again, remove the last 2 and try the next option which is 3 that means [2, 2, 3].

When this also backtracks remove 3 to try 5, that means [2, 2, 5].

Now as all the options are exhausted for [2, 2, \_ ], now backtrack to its previous state which is [2, 2], and so on...

**Code:**

class Solution {

        if(currSum>target) return; //backtrack

        if(currSum==target){

            ans.push\_back(currComb); //store the solution and backtrack

            return;

        }

        for(int i=currIndex; i<candidates.size(); i++){ //try all possible options for the next level

            currComb.push\_back(candidates[i]); //put 1 option into the combination

            currSum+=candidates[i];

            combination(candidates, target, currComb, currSum, i, ans); //try with this combination, whether it gives a solution or not.

            currComb.pop\_back(); //when this option backtrack to here, remove this and go on to the next option.

            currSum-=candidates[i];

        }

    }

public:

    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {

        vector<vector<int>> ans;

        vector<int> currComb;

        combination(candidates, target, currComb, 0, 0, ans);

        return ans;

    }

};

**Complexity analysis:**

<https://leetcode.com/problems/combination-sum/discuss/1755084/Detailed-Time-and-Space-Complecity-analysisc++javabacktracking>

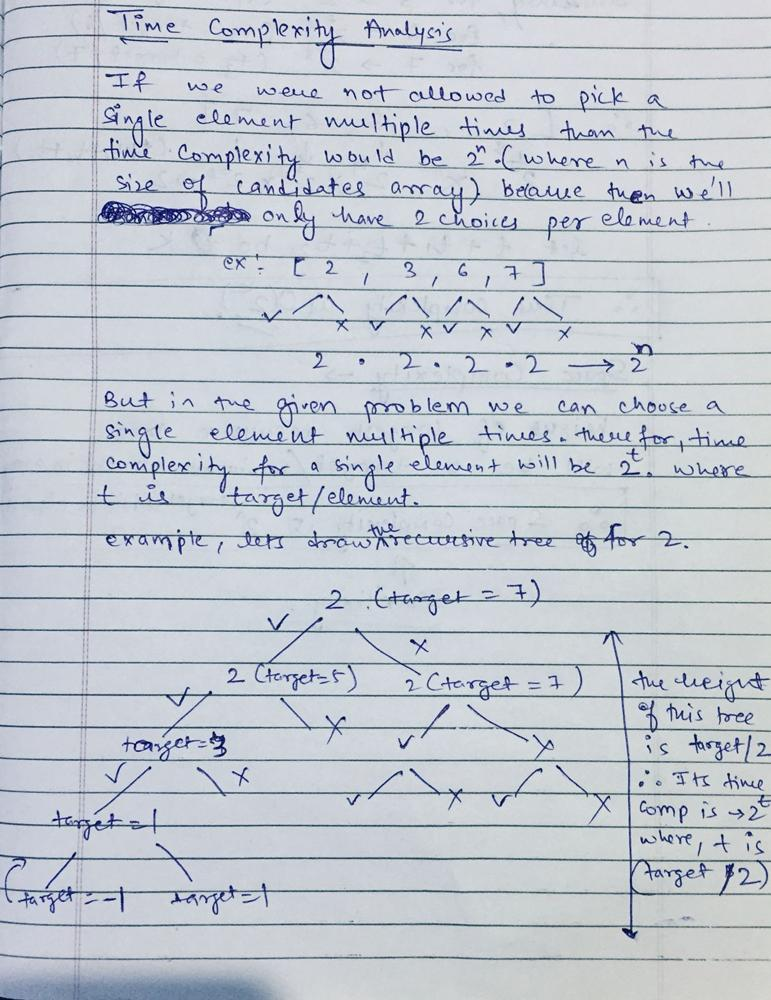
**Time Complexity** - O(2^k) where k is the sum of target/candidate[i] from i = 0 to size of candidate - 1.

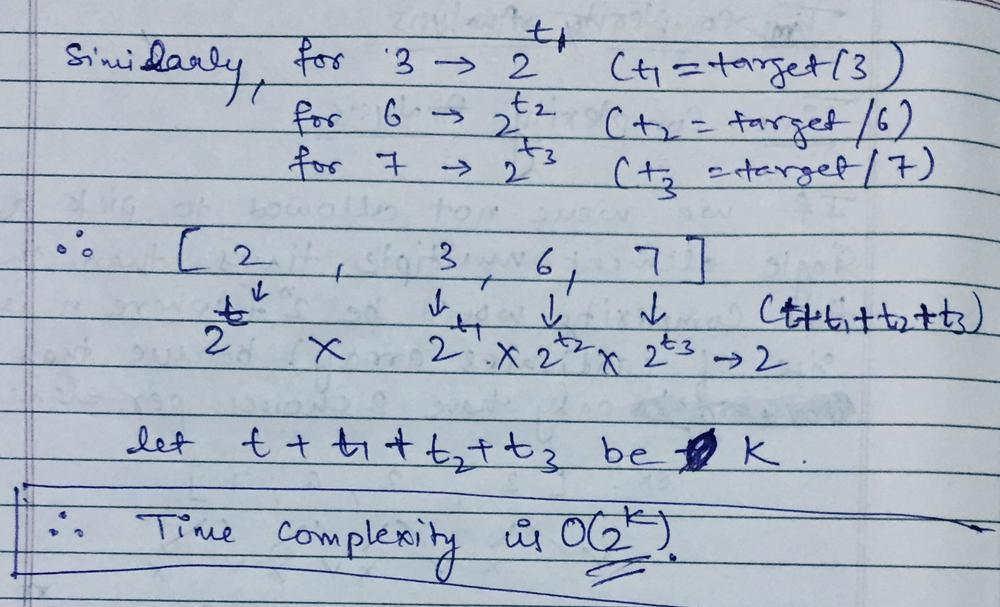
**Note-** *the time complexity of pushing cur into res is len\_of\_cur, so we can say TC - O( len\_of\_cur×2^k)*

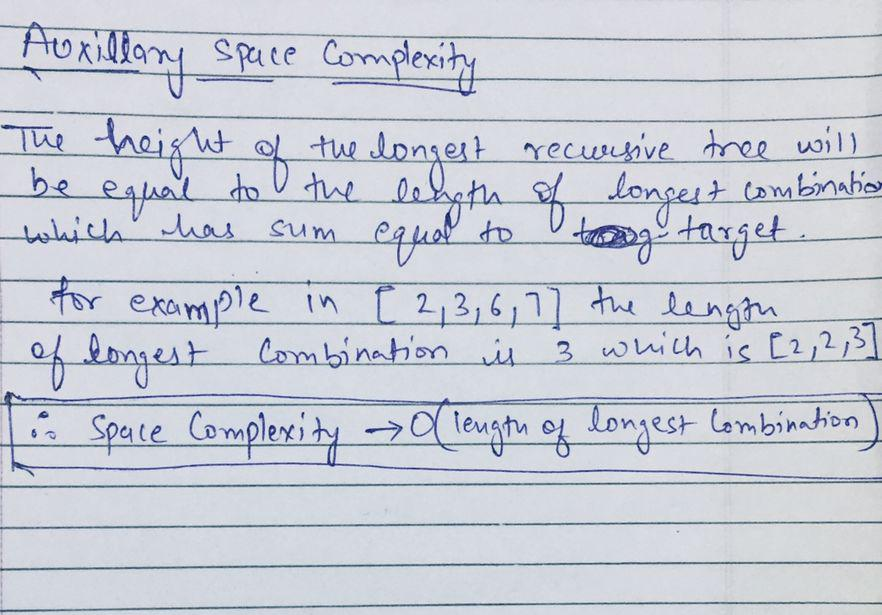
**Auxiliary Space Complexity** - O(length\_of\_longest\_combination).

**Detailed explanation for time and auxiliary space complexity**

**Note-** *the space complexity i mentioned is the auxiliary space complexity i.e. recursive stack space*







**Refer to Deepseek**

To understand how the backtracking solution works for the Combination Sum problem using the example candidates = [2,3,6,7] and target = 7, let's walk through the steps and observe when backtracking occurs. The solution uses backtracking to explore all valid combinations while avoiding duplicates and efficiently reusing elements.

**Key Steps for the Example:**

Initial Call: helper(7, candidates, result, tmp, 0)

tmp = [], target = 7, index = 0.

First Path (Using 2s):

Add 2 → tmp = [2], new target: 7 - 2 = 5.

Recurse with helper(5, ..., index=0).

Add 2 → tmp = [2,2], new target: 5 - 2 = 3.

Recurse with helper(3, ..., index=0).

Add 2 → tmp = [2,2,2], new target: 3 - 2 = 1.

Recurse with helper(1, ..., index=0).

Add 2 → target becomes -1 (invalid). **Backtrack**: Remove last 2 → tmp = [2,2,2].

Try 3: 1 - 3 = -2 (invalid). **Backtrack**.

Try 6, 7 (both invalid). **Backtrack** → tmp = [2,2].

Now in helper(3, ...), try 3 → tmp = [2,2,3], target 0. Valid combination added → result = [[2,2,3]]. **Backtrack** → tmp = [2,2].

Continue exploring other candidates at higher levels (no further valid paths here). **Backtrack** → tmp = [2].

Second Path (Using 3):

In the initial call, try i=1 (candidate 3):

Add 3 → tmp = [3], new target: 7 - 3 = 4.

Recurse with helper(4, ..., index=1).

Try 3 again → tmp = [3,3], new target: 4 - 3 = 1.

Recurse with helper(1, ...), no valid paths. **Backtrack** → tmp = [3].

Try 6, 7 → invalid. **Backtrack** → tmp = [].

Third Path (Using 7):

In the initial call, try i=3 (candidate 7):

Add 7 → tmp = [7], target 0. Valid combination added → result = [[2,2,3], [7]].

**When Backtracking Happens:**

After Invalid Paths: When the target becomes negative (e.g., adding 2 four times), backtrack by removing the last element.

After Finding a Valid Combination: Once a valid combination is added to result, backtrack to explore other possibilities (e.g., after adding [2,2,3] or [7]).

Exhausting All Options: When all candidates from the current index have been tried (e.g., after testing 3, 6, 7 in a subtree).

**Why Backtracking (Not Plain DFS):**

State Management: Backtracking modifies a single tmp list by adding/removing elements, avoiding the overhead of creating new lists at each step (memory-efficient).

Avoiding Duplicates: The index parameter ensures combinations are explored in a non-redundant order (e.g., [2,3] is allowed, but [3,2] is not, since once we move to i=1, we don’t revisit i=0).

Reuse of Elements: By passing i (not i+1) to the next recursion, the same element can be reused (e.g., 2 used multiple times).

**Example Flow Summary:**

Start with target=7, tmp=[]

├─ Add 2 → target=5

│ ├─ Add 2 → target=3

│ │ ├─ Add 2 → target=1 (invalid after further steps)

│ │ └─ Add 3 → target=0 ✅ [2,2,3]

├─ Add 3 → target=4 (no valid paths)

└─ Add 7 → target=0 ✅ [7]

Result: [[2,2,3], [7]].

Backtracking ensures all valid paths are explored while efficiently reusing the same tmp list and avoiding duplicates. A plain DFS without backtracking would require creating new lists at each step, leading to higher memory usage and potential duplication issues.

**Solution 2: Backtracking style 2 (10min)**

class Solution {

    public List<List<Integer>> combinationSum(int[] candidates, int target) {

        List<List<Integer>> result = new ArrayList<List<Integer>>();

        helper(target, candidates, result, new ArrayList<Integer>(), 0);

        return result;

    }

    private void helper(int target, int[] candidates, List<List<Integer>> result, List<Integer> tmp, int index) {

        if(target == 0) {

            result.add(new ArrayList<Integer>(tmp));

            return;

        }

        if(target < 0 || index >= candidates.length) {

            return;

        }

        // Not pick

        // 'index + 1' to skip current element

        helper(target, candidates, result, tmp, index + 1);

        // Pick

        tmp.add(candidates[index]);

        // 'index' not change since same number can use unlimited number of times

        helper(target - candidates[index], candidates, result, tmp, index);

        tmp.remove(tmp.size() - 1);

    }

}

Time Complexity: O(2^k) where k is the sum of target/candidate[i] from i = 0 to size of candidate - 1

Space Complexity: O(length\_of\_longest\_combination)

**Progress of Backtracking style 2**

target=7,candidates={2,3,6,7},result={},tmp={},index=0

--------------------------------------

Round 1:

helper(7,{2,3,6,7},{},{},0)

target=7

index=0

--------------------------------------

Round 2:

helper(7,{2,3,6,7},{},{},0+1) not pick

target=7

index=1

--------------------------------------

Round 3:

helper(7,{2,3,6,7},{},{},1+1) not\_pick

target=7

index=2

--------------------------------------

Round 4:

helper(7,{2,3,6,7},{},{},2+1) not\_pick

target=7

index=3

--------------------------------------

Round 5:

helper(7,{2,3,6,7},{},{},3+1) not\_pick

target=7

index=4 >= candidates.length

return to Round 4 statistics

--------------------------------------

Round 6:

index=3

tmp.add(candidates[3])={7}

helper(7-7,{2,3,6,7},{},{7},3) pick

target=0

index=3

--------------------------------------

Round 7:

target=0 -> result={{7}}

return to Round 6 statistics

tmp.remove(1 - 1)={}

end statement back to Round 3 statistics

--------------------------------------

Round 8:

index=2

tmp.add(candidates[2])={6}

helper(7-6,{2,3,6,7},{{7}},{6},2) pick

target=1

index=2

--------------------------------------

Round 9:

helper(1,{2,3,6,7},{{7}},{6},2+1) not\_pick

target=1

index=3

--------------------------------------

Round 10:

helper(1,{2,3,6,7},{{7}},{6},3+1) not\_pick

target=1

index=4 >= candidates.length

return to Round 9 statistics

--------------------------------------

Round 11:

index=3

tmp.add(candidates[3])={6,7}

helper(1-7,{2,3,6,7},{{7}},{6,7},3) pick

target=-6 < 0

return to Round 9 statistics

tmp.remove(2 - 1)={6}

end statement back to Round 8 statistics

--------------------------------------

Round 12:

index=2

tmp.add(candidates[2])={6,6}

helper(1-6,{2,3,6,7},{{7}},{6,6},2) pick

target=-5 < 0

return to Round 9 satistics

tmp.remove(2-1)={6}

end statement back to Round 8 statistics

tmp.remove(1-1)={}

end statement back to Round 2 statistics

--------------------------------------

Round 13:

index=1

tmp.add(candidates[1])={3}

helper(7-3,{2,3,6,7},{{7}},{3},1+1) pick

target=4

index=2

--------------------------------------

Round 14:

helper(4,{2,3,6,7},{{7}},{3},2+1) not\_pick

target=4

index=3

--------------------------------------

Round 15:

helper(7-3,{2,3,6,7},{{7}},{3},3) pick

index=4 >= candidates.length

return to Round 14 statistics

tmp.add(candidates[3])={3,7}

helper(4-7,{2,3,6,7},{{7}},{3,7},2) pick

target=-3 < 0

return to Round 14 satistics

tmp.remove(2-1)={3}

end statement back to Round 13 statistics

--------------------------------------

Round 16:

index=2

tmp.add(candidates[2])={3,6}

helper(4-6,{2,3,6,7},{{7}},{3,6},2) pick

target=-2 < 0

return to Round 14 satistics

tmp.remove(2-1)={3}

end statement back to Round 13 statistics

--------------------------------------

Round 17:

index=1

tmp.add(candidates[1])={3,3}

helper(4-3,{2,3,6,7},{{7}},{3,3},1) pick

target=1

index=1

--------------------------------------

Round 18:

helper(1,{2,3,6,7},{{7}},{3,3},1+1) not\_pick

target=1

index=2

--------------------------------------

Round 19:

helper(1,{2,3,6,7},{{7}},{3,3},2+1) not\_pick

target=1

index=3

--------------------------------------

Round 20:

helper(1,{2,3,6,7},{{7}},{3,3},3+1) not\_pick

target=1

index=4 >= candidates.length

return to Round 14 statistics

tmp.add(candidates[3])={3,3,7}

--------------------------------------

Round 21:

helper(1-7,{2,3,6,7},{{7}},{3,3,7},3) pick

target=-6 < 0

return to Round 19 satistics

tmp.remove(3-1)={3,3}

--------------------------------------

Round 22:

helper(1-6,{2,3,6,7},{{7}},{3,3,6},2) pick

target=-5 < 0

return to Round 19 satistics

tmp.remove(3-1)={3,3}

--------------------------------------

Round 23:

helper(1-3,{2,3,6,7},{{7}},{3,3,3},1) pick

target=-2 < 0

return to Round 19 satistics

tmp.remove(3-1)={3,3}

end statement back to Round 17 statistics

tmp.remove(2-1)={3}

end statement back to Round 17 statistics

tmp.remove(1-1)={}

--------------------------------------

Round 24:

index=0

tmp.add(candidates[0])={2}

helper(7-2,{2,3,6,7},{{7}},{2},0) pick

target=5

index=0

--------------------------------------

Round 25:

helper(5,{2,3,6,7},{{7}},{2},0+1) not\_pick

target=5

index=1

--------------------------------------

Round 26:

helper(5,{2,3,6,7},{{7}},{2},1+1) not\_pick

target=5

index=2

--------------------------------------

Round 27:

helper(5,{2,3,6,7},{{7}},{2},2+1) not\_pick

target=5

index=3

--------------------------------------

Round 28:

helper(5,{2,3,6,7},{{7}},{2},3+1) not\_pick

target=5

index=4 >= candidates.length

return to Round 27 statistics

--------------------------------------

Round 29:

index=3

tmp.add(candidates[3])={2,7}

helper(5-7,{2,3,6,7},{{7}},{2,7},3) pick

target=-2 < 0

return to Round 27 statistics

tmp.remove(2-1)={2}

end statement back to Round 26 statistics

--------------------------------------

Round 30:

index=2

tmp.add(candidates[2])={2,6}

helper(5-6,{2,3,6,7},{{7}},{2,6},2) pick

target=-1 < 0

return to Round 27 statistics

tmp.remove(2-1)={2}

end statement back to Round 26 statistics

--------------------------------------

Round 31:

index=1

tmp.add(candidates[1])={2,3}

helper(5-3,{2,3,6,7},{{7}},{2,3},1) pick

target=2

index=1

--------------------------------------

Round 32:

helper(2,{2,3,6,7},{{7}},{2,3},1+1) not\_pick

target=2

index=2

--------------------------------------

Round 33:

helper(2,{2,3,6,7},{{7}},{2,3},2+1) not\_pick

target=2

index=3

--------------------------------------

Round 34:

helper(2,{2,3,6,7},{{7}},{2,3},3+1) not\_pick

target=2

index=4 >= candidates.length

return to Round 33 statistics

--------------------------------------

Round 35:

index=3

tmp.add(candidates[3])={2,3,7}

helper(2-7,{2,3,6,7},{{7}},{2,3,7},3) pick

target=-5 < 0

return to Round 33 statistics

tmp.remove(3-1)={2,3}

end statement back to Round 32 statistics

--------------------------------------

Round 36:

index=2

tmp.add(candidates[3])={2,3,6}

helper(2-6,{2,3,6,7},{{7}},{2,3,6},2) pick

target=-4 < 0

return to Round 33 statistics

tmp.remove(3-1)={2,3}

end statement back to Round 32 statistics

--------------------------------------

Round 37:

index=1

tmp.add(candidates[3])={2,3,3}

helper(2-3,{2,3,6,7},{{7}},{2,3,3},1) pick

target=-1 < 0

return to Round 33 statistics

tmp.remove(3-1)={2,3}

end statement back to Round 32 statistics

tmp.remove(2-1)={2}

end statement back to Round 24 statistics

--------------------------------------

Round 38:

index=0

tmp.add(candidates[0])={2,2}

helper(5-2,{2,3,6,7},{{7}},{2,2},0) pick

target=3

index=0

--------------------------------------

Round 39:

index=0

helper(3,{2,3,6,7},{{7}},{2,2},0+1) not\_pick

target=3

index=1

--------------------------------------

Round 40:

index=1

helper(3,{2,3,6,7},{{7}},{2,2},1+1) not\_pick

target=3

index=2

--------------------------------------

Round 41:

index=2

helper(3,{2,3,6,7},{{7}},{2,2},2+1) not\_pick

target=3

index=3

--------------------------------------

Round 42:

index=3

helper(3,{2,3,6,7},{{7}},{2,2},3+1) not\_pick

target=3

index=4 >= candidates.length

return to Round 41 statistics

--------------------------------------

Round 43:

index=3

tmp.add(candidates[3])={2,2,7}

helper(3-7,{2,3,6,7},{{7}},{2,2,7},3) pick

target=-4 < 0

return to Round 42 statistics

tmp.remove(3-1)={2,2}

end statement back to Round 41 statistics

--------------------------------------

Round 44:

index=2

tmp.add(candidates[2])={2,2,6}

helper(3-6,{2,3,6,7},{{7}},{2,2,6},2) pick

target=-3 < 0

return to Round 42 statistics

tmp.remove(3-1)={2,2}

end statement back to Round 41 statistics

--------------------------------------

Round 45:

index=1

tmp.add(candidates[1])={2,2,3}

helper(3-3,{2,3,6,7},{{7}},{2,2,3},1) pick

target=0 -> result={{7}{2,2,3}}

return to Round 39 statistics

tmp.remove(3-1)={2,2}

end statement back to Round 39 statistics

--------------------------------------

Round 46:

index=0

tmp.add(candidates[0])={2,2,2}

helper(3-2,{2,3,6,7},{{7}},{2,2,2},0) pick

target=1

index=0

--------------------------------------

Round 47:

helper(1,{2,3,6,7},{{7}},{2,2,2},0+1) not\_pick

target=1

index=1

--------------------------------------

Round 48:

helper(1,{2,3,6,7},{{7}},{2,2,2},1+1) not\_pick

target=1

index=2

--------------------------------------

Round 49:

helper(1,{2,3,6,7},{{7}},{2,2,2},2+1) not\_pick

target=1

index=3

--------------------------------------

Round 50:

helper(1,{2,3,6,7},{{7}},{2,2,2},3+1) not\_pick

target=1

index=4 >= candidates.length

return to Round 49 statistics

--------------------------------------

Round 51:

index=3

tmp.add(candidates[3])={2,2,2,7}

helper(1-7,{2,3,6,7},{{7}},{2,2,2,7},3) pick

target=-6 < 0

return to Round 50 statistics

tmp.remove(4-1)={2,2,2}

end statement back to Round 49 statistics

--------------------------------------

Round 52:

index=2

tmp.add(candidates[2])={2,2,2,6}

helper(1-6,{2,3,6,7},{{7}},{2,2,2,6},2) pick

target=-5 < 0

return to Round 50 statistics

tmp.remove(4-1)={2,2,2}

end statement back to Round 49 statistics

--------------------------------------

Round 53:

index=1

tmp.add(candidates[1])={2,2,2,3}

helper(1-3,{2,3,6,7},{{7}},{2,2,2,3},1) pick

target=-2 < 0

return to Round 50 statistics

tmp.remove(4-1)={2,2,2}

end statement back to Round 49 statistics

--------------------------------------

Round 54:

index=0

tmp.add(candidates[1])={2,2,2,2}

helper(1-2,{2,3,6,7},{{7}},{2,2,2,2},0) pick

target=-1 < 0

return to Round 50 statistics

tmp.remove(4-1)={2,2,2}

tmp.remove(3-1)={2,2}

tmp.remove(2-1)={2}

tmp.remove(1-1)={}

======================================

End

======================================

Result time elapsed statistics:

result={{7}}

result={{7}{2,2,3}}

**Backtracking style 2 recursion step by step (to view stretch on Notepad++)**

candidates={2,3,6,7}, target=7(tg7), index=0(idx0), not take x(ntx), take x(tx)

branch ending condition: (1)index >=4 -> idx4 OR (2)target < 0 -> tg(<0)

combination found: target == 0(tg0)

                                                                                                                      { }tg7,idx0

                                                / not take 2(nt2)                                                                                                            \ take 2(t2)

                                              { }tg7,idx1                                                                                                                    {2}tg5,idx0

                        /nt3                                              \t3                                                                  /nt2                                                          \t2

                      { }tg7,idx2                                          {3}tg4,idx1                                                        {2}tg5,idx1                                                      {2,2}tg3,idx0

          /nt6                  \t6                          /nt3                          \t3                                  /nt3                          \t3                                  /nt2                        \t2

        { }tg7,idx3            {6}tg1,idx2                {3}tg4,idx2                    {3,3}tg1,idx1                      {2}tg5,idx2                    {2,3}tg2,idx1                      {2,2}tg3,idx1                {2,2,2}tg1,idx0

  /nt7      \t7          /nt6        \t6              /nt6    \t6                  /nt3        \t3                    /nt6      \t6                /nt3          \t3                  /nt3        \t3                  /nt2      \t2

{ }tg7,idx4 {7}tg0,idx3 {6}tg1,idx3 {6,6}tg-5,idx2  {3}tg4,idx3 {3,6}tg-2,idx2  {3,3}tg1,idx2 {3,3,3}tg-2,idx1        {2}tg5,idx3 {2,6}tg-1,idx2    {2,3}tg2,idx2  {2,3,3}tg-1,idx1  {2,2}tg3,idx2 {2,2,3}tg0,idx1  {2,2,2}tg1,idx1 {2,2,2,2}tg-1,idx0

                        /nt7    \t7                /nt7  \t7                  /nt6    \t6                        /nt7      \t7                  /nt6      \t6                    /nt6      \t6                    /nt3      \t3

                  {6}tg1,idx4 {6,7}tg-6,idx3 {3}tg4,idx4 {3,7}tg-3,idx3  {3,3}tg1,idx3 {3,3,6}tg-5,idx2        {2}tg5,idx4 {2,7}tg-2,idx3    {2,3}tg2,idx3 {2,3,6}tg-4,idx2    {2,2}tg3,idx3 {2,2,6}tg-3,idx2  {2,2,2}tg1,idx2 {2,2,2,3}tg-2,idx1

                                                                        /nt7  \t7                                                          /nt7  \t7                        /nt7    \t7                      /nt6      \t6

                                                                {3,3}tg1,idx4 {3,3,7}tg-6,idx3                                    {2,3}tg2,idx4 {2,3,7}tg-5,idx3      {2,2}tg3,idx4 {2,2,7}tg-4,idx3      {2,2,2}tg1,idx3 {2,2,2,6}tg-5,idx2

                                                                                                                                                                                                          /nt7      \t7

                                                                                                                                                                                                {2,2,2}tg1,idx4  {2,2,2,7}tg-6,idx3

**Refer to**

<https://leetcode.com/problems/combination-sum/discuss/1777334/C%2B%2BorDetailed-Explanation-w-TREE-DIAGRAMor-RECURSION-%2B-BACKTRACKINGor-EACH-STEP-EXPLAINED>

***Brief note about Question-***

In simple words, we have to *return all possible combination of array whose sum is equal to a particular target.*

Let's take an example not given in question - Suppose our array is given to us as arr[]: [2, 3] and target as 6

Then, all possible combinations of array whose sum is equal to 6 is - [[2,2,2] , [3,3]]

***General discussion on how we develop approach-***

This was a problem where we have to explore all possibility, make each combination and if sum of a combination becomes equal to target sum then we have to store that possible combination in our answer array.

One more thing we have to notice here is that, **for a particular element we have unlimited choice** i.e. we can choose a element as many times as we want.

But their is some restriction also on choosing a number.

See for every number in making our target sum, we have two possibility i.e.

**Whether to include that element in making our target sum**. **Whether not to include that element in making our target sum**.

We will try and explore each possible combination and if at a point we got our sum as zero then we will say yeah!!, we find a possible combination and include that combination in our answer.

Suppose if at any point our target sum becomes less than zero, then we will return from that point and will not explore further possibility by saying that, ok our target sum becomes negative that means from now no any combination is possible because we have given a non - negative array.

See below tree diagram for more clarity.

***How Tree diagram will work-***

We will make a op array, that contains all the possible combinations sum of the array.

We will start from the index 0 and as we already discussed that for each and every element we have two possibility whether to include this element in making our answer or not, so we will explore all possibilities.

op array represents which elements this array contains now in making combination sum.

Target represents the left combination sum that we have to make. Initially it is same as the original target that we have to make.

The red cursor below the array, points that on which index we are currently standing.

If at any point our target becomes zero, then we will include that combination in our answer array saying that yes, this is an possible combination and return from there.

If at any point our target becomes less than zero, then we return from there saying that we are never able to make our combination sum by this combination.

If at any point we cross the size of the array then we will return from there saying that no more element is left to choose.

As we discussed for every element we have two choices whether to include in our answer or not include in our answer.

So, if we do not a include a element in our answer then without decreasing target sum, we will move to next index.

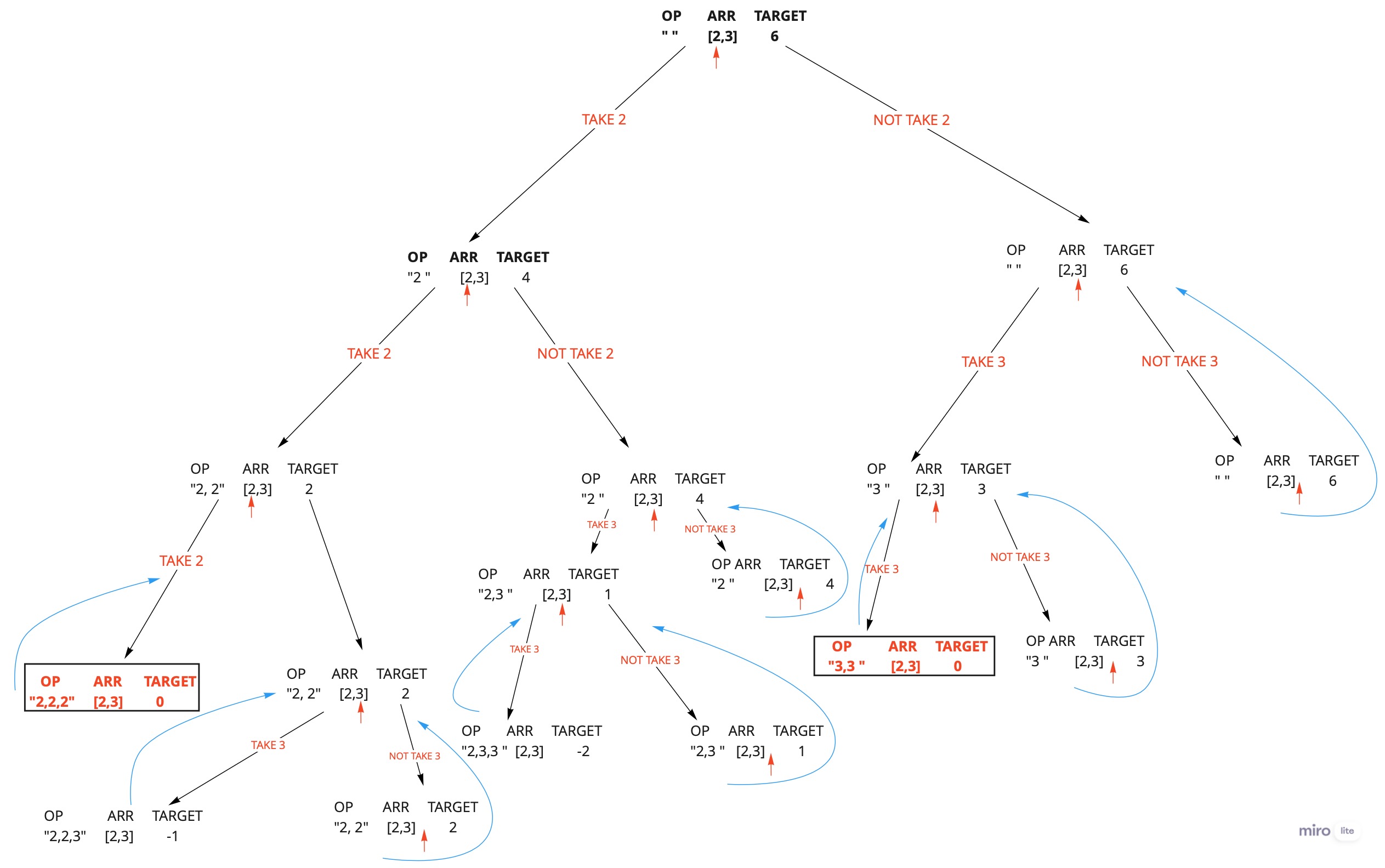
**Why we do not decrees sum?** because we will choose not to include in this element in our combination, hence it does not contribute in making our sum.

But, if we choose a particular element to include in our answer, then we will decrease the target sum but we will not move to next index.

**Why we will not move to next index?** because for a specific element *we have unlimited number of choice*, so it may be possible that specific element again contribute in making our sum.

If Image is not clearly visible to you, then for that I have uploaded it on my drive.

You may visit this link to see Image in good quality.



***Solution - I (using backtracking, Accepted)-***

class Solution {

public:

    vector<vector<int>> ans; // 2 D vector to store our answer

    void solve(int i, vector<int>& arr, vector<int>& temp, int target)

    {

        // if our target becomes zero at any point, then yess!! we wil find a possible combination

        if(target == 0)

        {

            ans.push\_back(temp); // include that combination in our answer

            return; // and then return, we are now not gonna explore more possiblity

        }

        // if at any point target becomes less than zero, then simply return, saying that no it is notpossible to our target combination sum

        if(target < 0)

            return;

        // if index crosses the last index, we will return saying that no more element is left to choosee

        if(i == arr.size())

            return;

        // As we dicussed for every element we have two choices whether to include in our answer or not include in our answer.

//so now, we are doing that

        // we are not taking the ith element,

        // so without decreasing sum we will move to next index because it will not contribute in making our sum

        solve(i + 1, arr, temp, target);

        // we are taking the ith element and not moving onto the next element because it may be possible that this element again contribute in making our sum.

        // but we decrease our target sum as we are consediring that this will help us in making our target sum

        temp.push\_back(arr[i]); // including ith element

        solve(i, arr, temp, target - arr[i]); // decreasing sum,and call again function

        temp.pop\_back(); // backtrack

    }

    vector<vector<int>> combinationSum(vector<int>& arr, int target) {

        ans.clear(); // clear global array, make to sure that no garbage value is present in it

        vector<int> temp; // temporary vector that tries all possible combination

        solve(0, arr, temp, target); // calling function, and see we start from index zero

        return ans; // finally return the answer array

    }

};

**Refer to**

[L40.Combination Sum II (Refer L39.Combination Sum and L90/P11.2.Subsets II)](note://7895D4AD64EC43CA903579F3C8F803AE)

[L216.Combination Sum III](note://WEB322b73c0787cd12b4eed700838bed14d)

[L322.Coin Change (Ref.L39,L46,L377)](note://WEB99a663d0191c2bc0c88aec7cf5f3d9fd)

[L377.P16.4.Combination Sum IV(Backpack VI) (Refer L39.Combination Sum)](note://8000F8241A0B4F3AA39E5CA16B6A7BB8)