L7 All Kind of Patterns in Recursion

Sunday, December 18, 2022 4:19 PM

This lecture we learn all about the different - different pattern based on Recursion In the previously lecture we learn in depth all about the Subsequences in depth

P1 Print All the Subsequence whos Sum is Equal to the K

In this problem our task is print that all the subsequence that sum is equal to the K

→ Here we again used the Concept of Pick and Not Pick

Example arr = [1, 2, 1], Sum = 2
(Now our task is print all the Subsequence that sum is equal to the given sum)

[1, 1]
[2] } → that is all the possible Subsequence that sum is equal to the given sum

Thought Process

→ Again we used the concept of pick and Not pick that we learn previously

```
myfunction(index, mylist, K){
    if(index>= n){
        if(sum(mylist) == k){
            print(mylist)
            return;
        else{
            return;
        }
    mylist.add(arr[index])
    myfunction(index+1, mylist, K) ]
    mylist.removed(arr[index])
    myfunction(index+1, mylist, K) ]
}
```

→ my base case

→ Add the element / Subsequence into the mylist

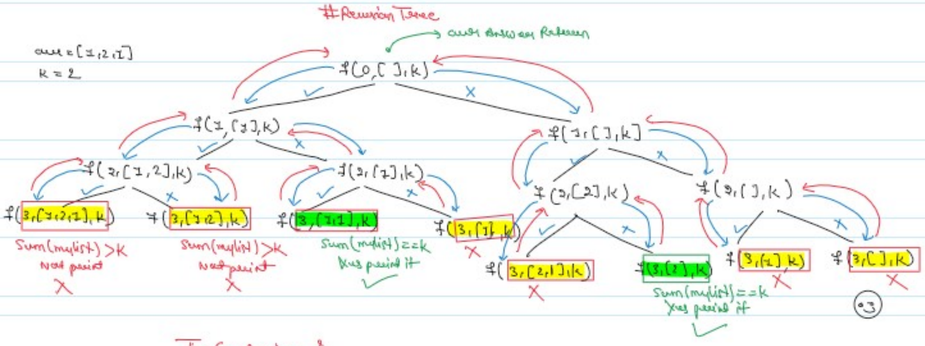
→ pick case

→ Not pick case

→ Here we not draw the Recursion code because that is the same sudo code that we learn previously that is why

Recursion Tree

arr = [1, 2, 1]
k = 2



Time Complexity = ?

→ for every Iteration we have tuple of option take or Not take = $\boxed{2^n}$ (Exponential)

$O(2^n)$

Space Complexity = $O(N) + O(N) \approx O(N)$

→ for array to store the element

→ because we store the array element that is our Subsequences

P2 Print Only One Subsequence whos Sum is Equal to the K

→ This is the modified problem of pervous one here our task is print only one Subsequence

```
myfunction(index, mylist, K){
    if(index>=n){
        if(sum(mylist)==k){
            print(mylist)
            return True
        else{
            return False;
        }
    mylist.add(arr[index])
    if myfunction(index+1, mylist,K) ==True{
        return True;
    }
    mylist.remaved(arr[index])
    if myfunction(index+1, mylist, K) ==True {
        return True;
    }
    return false;
}
```

→ my base case

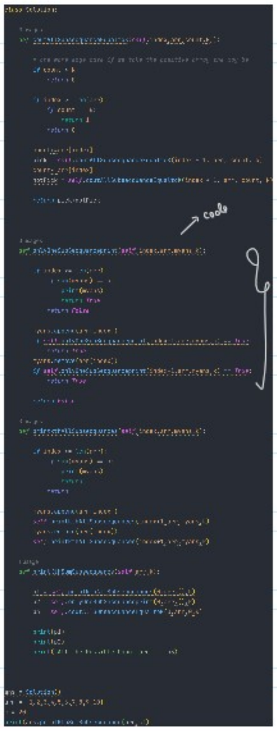
→ Add the element/ Subsequence into the mylist

Pick condition

Not pick condition

→ if my Recursion become True than No Need for futher Recursion Just Return

→ Recursion Tree, Sudo code and Recursion function are Same that is why Here We Not again discuss



```
class Solution:
[illegible]
```

```
[illegible]
```

P3 Total Count the Subsequence whos Sum is Equal to the K

Now in this problem we only count the all the possible Subsequence that Sum is equal to the K

Thought Process

→ Now previously we learned the Concept pick and Not pick that used here again

→ previously we maintain the Array for containing the element that sum is equal to the (K) and that array print it

→ But here in placed of Array we used the Variable and Check

```
if Variable == (k):
    Count += 1
```

→ that is the our Approach for this problem

example

arr = [1,2,1]   K = 2

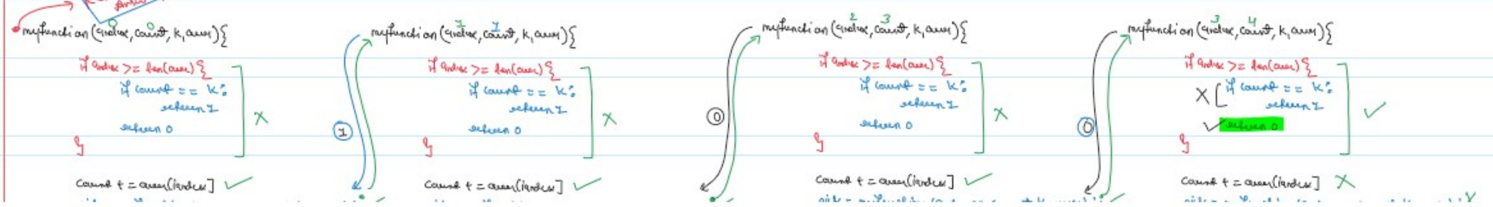
[1,1]
[2]   → Count = 2 → answer

```
myfunction(index, count, k, arr){
    if index >= len(arr){      → Base case
        if count == k:
            return 1
        return 0
    }

    Count += arr[index]
    pick = myfunction(index+1, count, k, arr);
    Count -= arr[index]
    Notpick = myfunction(index+1, count, k, arr);

    return pick + Notpick;
}
```

Let's Dry Run and understand the Recursion

arr = [1,2,1]
K = 2
Count = 0

Return 2 Answer



```
myfunction(index, count, k, arr){      index = 0, count = 0
    if index >= len(arr){
        if count == k:
            return 1
        return 0
    }                                   X
    Count += arr[index]   ✓
```

(1)

```
myfunction(index, count, k, arr){      index = 1, count = 1
    if index >= len(arr){
        if count == k:
            return 1
        return 0
    }                                   X
    Count += arr[index]   ✓
```

(0)

```
myfunction(index, count, k, arr){      index = 2, count = 3
    if index >= len(arr){
        if count == k:
            return 1
        return 0
    }                                   X
    Count += arr[index]   ✓
```

(0)

```
myfunction(index, count, k, arr){      index = 3, count = 4
    if index >= len(arr){
      X [ if count == k:
            return 1
      ✓ return 0
    }                                   ✓
    Count += arr[index]   X
```

```
        return 1
    return 0          ] X
}

Count += arr[index] ✓
pick = myfunction(index+1,count,K,arr); ✓
Count -= arr[index] = 0 ✓
Notpick = myfunction(index+1, count, k, arr); ✓

return pick+Notpick; ✓
        1 + 1
}
```

Again my Same Recursion call till the base case Actived and Return ans like that Similary ③

→ for this case Return Overall ①

```
        return 1
    return 0          ] X
}

Count += arr[index] ✓
pick = myfunction(index+1,count,K,arr); ✓
Count -= arr[index] = 1 ✓
Notpick = myfunction(index+1, count, k, arr); ✓

return pick+Notpick; ✓
        0 + 1
}

myfunction(index = 2, count = 1, k, arr){
    if index >= len(arr){
        if count == k:
            return 1
        return 0
    }                        ] X

    Count += arr[index] ✓
    pick = myfunction(index+1,count,K,arr); ✓
    Count -= arr[index] = 1 ✓
    Notpick = myfunction(index+1, count, k, arr); ✓
      ↰0
    ✓return pick+Notpick;
            1 + 0
}
```

→index became 3 and count 1 So Return 0

```
myfunction(index = 3, count = 2, k, arr){
    if index >= len(arr){
        ✓[ if ~~count~~ == k:
               return 1
        return 0
    }                        ] ✓

    Count += arr[index]
    pick = myfunction(index+1,count,K,arr);
    Count -= arr[index]
    Notpick = myfunction(index+1, count, k, arr);

    return pick+Not pick;     ] X
}
```

```
    return 0          ] X
}

Count += arr[index] ✓
pick = myfunction(index+1,count,K,arr); ✓
Count -= arr[index] = 3 ✓
Notpick = myfunction(index+1, count, k, arr); ✓

return pick+Notpick; ✓
        0 + 0
}

myfunction(index = 3, count = 2, k, arr){
    if index >= len(arr){
        if count == k:
            return 1
        ✓return 0
    }                        ] ✓

    Count += arr[index]
    pick = myfunction(index+1,count,K,arr);
    Count -= arr[index]
    Notpick = myfunction(index+1, count, k, arr);

    return pick+Notpick;     ] X
}
```

```
    ✓return 0         ] ✓
}

Count += arr[index] X
pick = myfunction(index+1,count,K,arr); X
Count -= arr[index] X
Notpick = myfunction(index+1, count, k, arr); X

return pick+Notpick; X
}
```

Time Complexity = ?

→ for every iteration we have tuple of option take on Not take = $\boxed{2^n}$ (Exponential)

$O(2^n)$

Space Complexity = O(N)

→ beause We store the arr element that is arr Subsequence