Coin Change

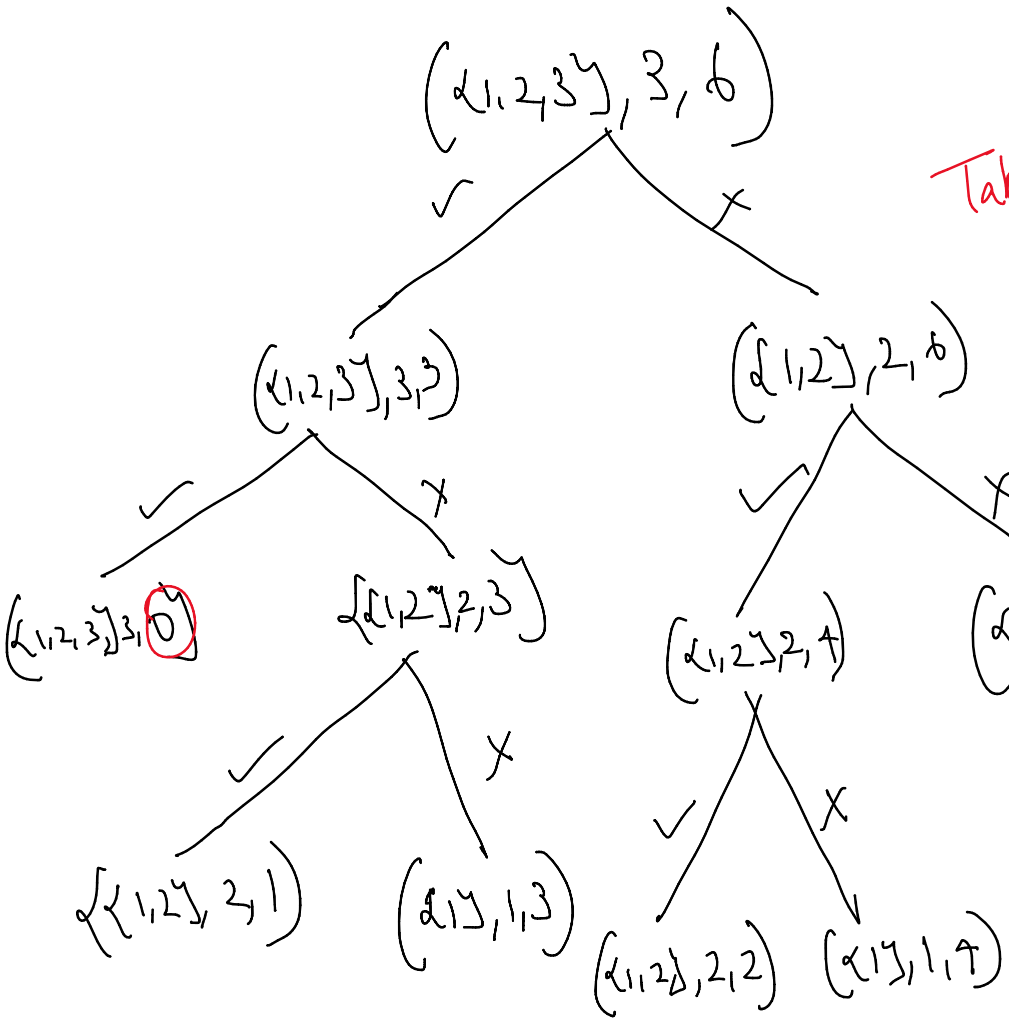
19 October 2022 14:08

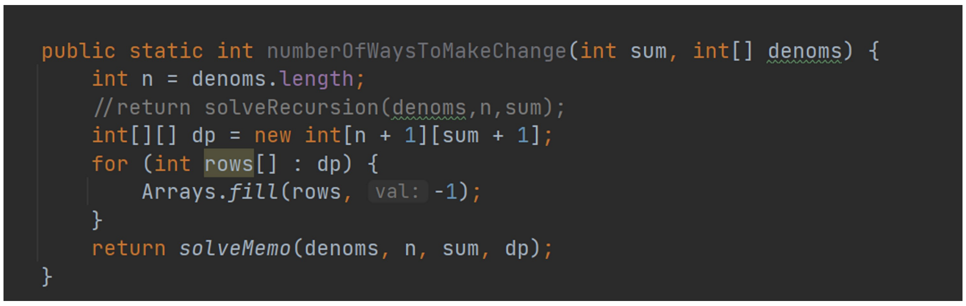
S = {1, 2, 3} V = 6

{array, size, V}

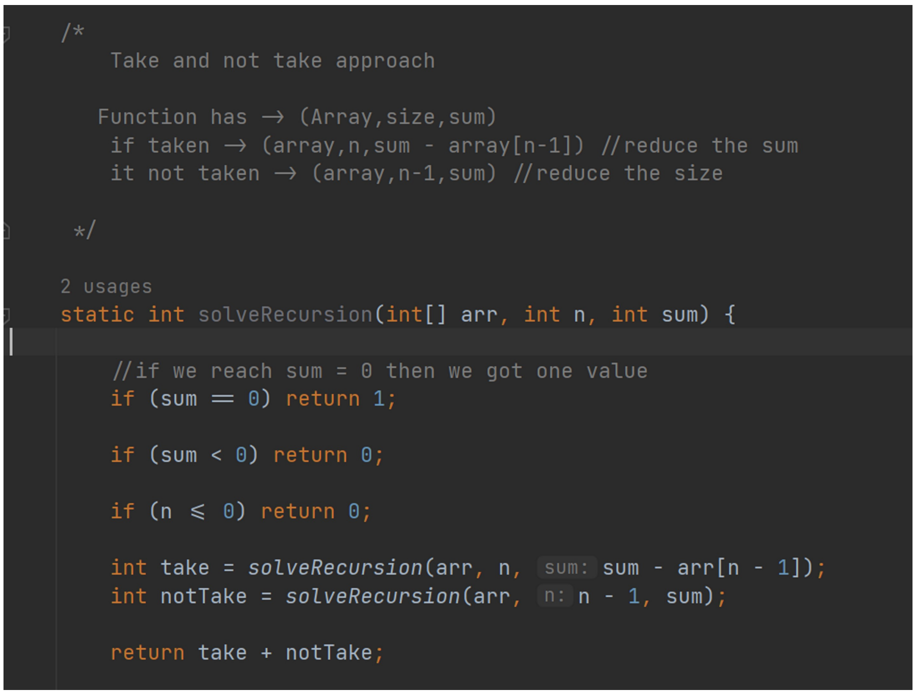
Take → (a, n, v-a[n-1])

not take (a, n-1, v)

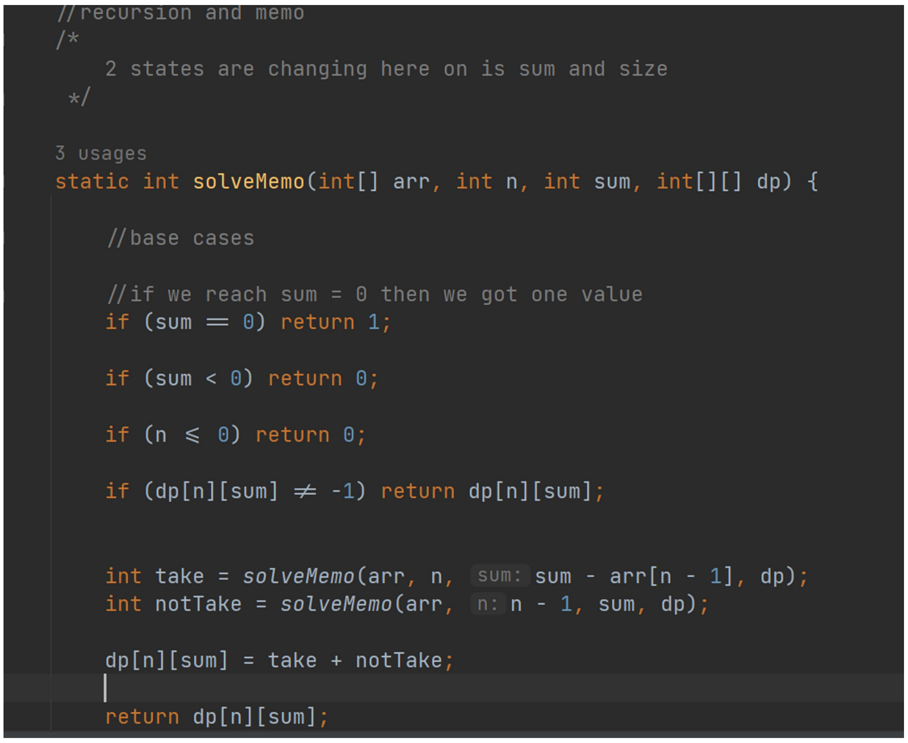




```java
public static int numberOfWaysToMakeChange(int sum, int[] denoms) {
    int n = denoms.length;
    //return solveRecursion(denoms,n,sum);
    int[][] dp = new int[n + 1][sum + 1];
    for (int rows[] : dp) {
        Arrays.fill(rows, val: -1);
    }
    return solveMemo(denoms, n, sum, dp);
}
```



```
/*
    Take and not take approach

   Function has → (Array,size,sum)
    if taken → (array,n,sum - array[n-1]) //reduce the sum
    it not taken → (array,n-1,sum) //reduce the size

 */

2 usages
static int solveRecursion(int[] arr, int n, int sum) {

    //if we reach sum = 0 then we got one value
    if (sum == 0) return 1;

    if (sum < 0) return 0;

    if (n <= 0) return 0;

    int take = solveRecursion(arr, n, sum: sum - arr[n - 1]);
    int notTake = solveRecursion(arr, n: n - 1, sum);

    return take + notTake;
```



```
//recursion and memo
/*
    2 states are changing here on is sum and size
 */

3 usages
static int solveMemo(int[] arr, int n, int sum, int[][] dp) {

    //base cases

    //if we reach sum = 0 then we got one value
    if (sum == 0) return 1;

    if (sum < 0) return 0;

    if (n <= 0) return 0;

    if (dp[n][sum] != -1) return dp[n][sum];


    int take = solveMemo(arr, n, sum: sum - arr[n - 1], dp);
    int notTake = solveMemo(arr, n: n - 1, sum, dp);

    dp[n][sum] = take + notTake;

    return dp[n][sum];
```