LeetCode 518

https://leetcode.com/problems/coin-change-2/description/

Yifeng Zeng

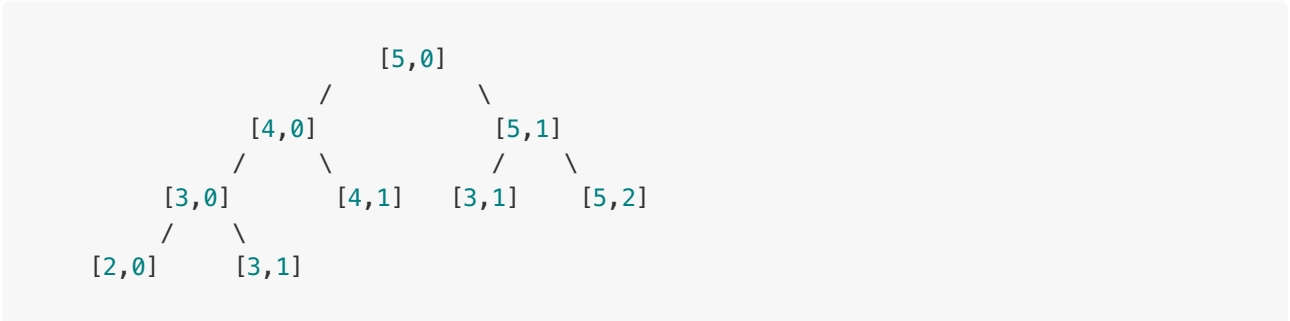
Description

518. Coin Change 2

Idea Report

Still, we can split it into two different situations. 1. choose the current coin, then the next recursion level I'm looking for amount - coins[index], and still looking at this index. 2. not choose the current coin, then the next recursion level I'm still looking for the amount, but the index become index + 1.

Take example [1,2,5] with amount 5. We define a pair [amount, index], then we can draw a solution space tree. Left child is to choose current index, and right child is not to choose current index. We can see that there might be a duplication, so we create a int[][] memo to do the memorized search.



Code

```
class Solution {
    // DFS + memo, choose or not choose AC
    public int change(int amount, int[] coins) {
```

```
        Arrays.sort(coins);
        int[][] memo = new int[amount + 1][coins.length + 1];
        for (int[] row : memo) {
            Arrays.fill(row, -1);
        }
        return helper(amount, coins, 0, memo);
    }

    private int helper(int amount, int[] coins, int index, int[][] memo) {
        if (amount == 0) {
            return 1;
        }
        if (amount < 0 || index >= coins.length) {
            return 0;
        }
        if (memo[amount][index] != -1) {
            return memo[amount][index];
        }

        int number = 0;
        // if we choose index
        if (amount - coins[index] >= 0) {
            number += helper(amount - coins[index], coins, index, memo);
        }
        // if we not choose index
        number += helper(amount, coins, index + 1, memo);
        return memo[amount][index] = number;
    }
}
```

	0	1	2	3	4	5
0	1	0	0	0	0	0
1	1	1	1	1	1	1
2	1	1	2	2	3	3
5	1	1	2	2	3	4

The definition is that use coins i and before i, we want to make amount j, there are f[i][j] ways. For f[i][j], we can split into two parts. One part is f[i-1][j], which means coins i-1 and before i-1, there are f[i-1][j] ways to sum to j. The other part is that if the current coins number is X, we can find f[i][j-x] to add this coin X we can sum to j. So the final f[i][j] = f[i-1][j] + f[i][j-x], where x is the current coin number.

Code

```
class Solution {
    // DP AC
    public int change(int amount, int[] coins) {
        int[][] f = new int[coins.length + 1][amount + 1];
        f[0][0] = 1;
        for (int i = 1; i <= coins.length; i++) {
            f[i][0] = 1;
            int coin = coins[i - 1];
            for (int j = 1; j <= amount; j++) {
                f[i][j] = f[i - 1][j];
                if (j - coin >= 0) {
                    f[i][j] += f[i][j - coin];
                }
            }
        }
        return f[coins.length][amount];
    }
}
```

Because we only need f[i-1] row, so we can optimize the space:

```
class Solution {
    // DP space optimization AC
    public int change(int amount, int[] coins) {
        int[] f = new int[amount + 1];
        f[0] = 1;
        for (int coin : coins) {
            for (int j = 1; j <= amount; j++) {
                if (j - coin >= 0) {
                    f[j] += f[j - coin];
                }
            }
        }
        return f[amount];
    }
}
```

Summary

- Try to create solution space tree which is really helpful

- When there is a duplication in the search, use something to save the values that has already been calculated.
- Drawing the table for DP is easier for analysis.
- Space optimization.