

## 详解动态规划最少硬币找零问题--JavaScript实现

硬币找零问题是动态规划的一个经典问题，其中最少硬币找零是一个变种，本篇将参照上一篇01背包问题的解题思路，来详细讲解一下最少硬币找零问题。如果你需要查看上一篇，可以点击下面链接：

[详解动态规划01背包问题--JavaScript实现](#)

也可以查看下一篇 [详解动态规划最长公共子序列--JavaScript实现](#)

下面让我们开始吧。

### 问题

给定4种面额的硬币1分，2分，5分，6分，如果要想找11分的零钱，怎么做才能使得找的硬币数量总和最少。

### 分析

最少硬币找零问题，是为了求硬币的组合，所以一个大前提是硬币无限量供应。我们建立如下表格来分析问题：

j 表示零钱总额

	j = 0	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7	j = 8	j = 9	j = 10	j = 11
i = 0 1分	0											
i = 1 2分	0											
i = 2 5分	0											
i = 3 6分	0											

其中每列用j表示零钱总额，每行i表示硬币面额。  $T[i][j]$  表示硬币个数，它是我们即将填入表格的数字。

在填写表格之前，我们需要先明确几个规则：


- 当填写第i行时，使用的硬币面额仅能是i以及小于i的面额。举个例子，比如我填写第0行， $i=0$ ，那么这一行只能使用面额为1分的硬币。当我填写第2行， $i=2$ ，那么可以使用1分，2分，5分三种面额的硬币。
- 当填写第j列时，表示当前需要使用硬币凑出的总额。比如 $j=6$ ，表示需要使用硬币组合出总额为6分的情况。

## 1. $i = 0$

当我们只能使用面额为1分的硬币时，根据上面的规则，那么很显然，总额为几分，就需要几个硬币。即  $T[i][j] = j$ 。

j 表示零钱总额

	j = 0	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7	j = 8	j = 9	j = 10	j = 11
i = 0 1分	0	1	2	3	4	5	6	7	8	9	10	11
i = 1 2分	0											
i = 2 5分	0											
i = 3 6分	0											



## 2. $i = 1$

当我们有1分和2分两种面额时，那么组合方式就相对多了点。


$i=1$   $j = 1$ ：总额为1时，只能使用1分的面额。即填1。  $i=1$   $j = 2$ ：总额为2时，可以使用2个1分的，也可以使用1个2分的。因为我们要求最少硬币，所以使用1个2分的。表格所表达的意思是硬币的数量，所以这里也填1。  $i=1$   $j = 3$ ：总额为3时，可以使用3个1分的，也可以使用1个1分加1个2分。因此这里应该填2。  $i=1$   $j = 4$ ：总额为4时，可以使用4个1分的，也可以使用2个1分加1个2分，也可以使用2个2分。其中硬币最少的情况应该是2个2分。因此这里填2。  $i=1$   $j = 5$ ：总额为5时，

组合就更多了，但是聪明的你应该能想到使用2个2分加1个1分，可以实现最少硬币的需求。因此这里填3。

我们来看填写完上面5格后的情况：

j 表示零钱总额

	j = 0	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7	j = 8	j = 9	j = 10	j = 11
i = 0 1分	0	1	2	3	4	5	6	7	8	9	10	11
i = 1 2分	0	1	1	2	2	3						
i = 2 5分	0											
i = 3 6分	0											



建议你自己再纸上照着我这图画一个表格。接下来，别急着填表。我们要根据已有的数据，总结出  $T[i][j]$  的规律，然后通过填写剩余表格来验证。

我们将硬币面额使用数组  $coins[i]$  来表示，根据表格有 1分= $coins[0]$ , 2分= $coins[1]$ 。当  $j < coins[i]$  时， $T[i][j]$  的值，应该等于它的同列，上一行，即使  $T[i][j] == T[i-1][j]$ 。比如我们从表中看到的， $T[1][1]==T[0][1]$ 。当  $j \geq coins[i]$  时，根据已有的  $i=1$  行可以推出一个规律，令  $a = 1+T[i][j-coins[i]]$ ， $T[i][j] = \min(T[i-1][j], a)$ ，即二者比较取最小值。可能一开始你看到这个关于  $a$  的公式，有点太突然，难以接受。稍微解释一下，当第  $i$  行，优先选择这一样的硬币，因为这一行的硬币面额最大，最有可能使得总硬币数量最少。因此  $j-coins[i]$ ，就很好理解了，就是选择了这一行的硬币后，还剩下多少总额。举个例子，当  $i=1, j=3$  时， $j-coins[1]=1$ 。那么选择2分后，还剩余总额为1，这时候我们再定位到  $i=1, j=1$ ，即  $T[1][1]$ ，它的值为1，再加上一个常数1，即得最终结果2。

再举例， $i=1, j=5$ 。由于是从左到右填表的，所以  $i=1, j < 5$  的表格都填完了。 $j-coins[i]=3$ ，定位到  $T[1][3]=2$ ，加上常数1，即得最后结果  $T[1][5]=3$ 。


其实公式本身很短，也很好记。如果实在无法理解，建议先不用纠结。先最小化浏览器，不要看本篇剩余的内容。带着这个解题公式，自己在纸上，把这个表格填写完整，在填表分析的过程中就能慢慢理解了。

### 3. 剩余内容

按照上一步所提供的公式，其实所有的  $T[i][j]$  都可以填完了。如下表格。

j 表示零钱总额

	j = 0	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7	j = 8	j = 9	j = 10	j = 11
i = 0 1分	0	1	2	3	4	5	6	7	8	9	10	11
i = 1 2分	0	1	1	2	2	3	3	4	4	5	5	6
i = 2 5分	0	1	1	2	2	1	2	2	3	3	2	3
i = 3 6分	0	1	1	2	2	1	1	2	2	3	2	2



建议先自己再纸上填表，填完了，再和我的图对比一下，看是否答案存在出入。

#### 4.伪代码

以上的填表逻辑，使用伪代码表示如下

```
if(i == 0){
    T[i][j] = j/coins[i]; //硬币找零一定要有个 最小面额1，否则会无解
}else{
    if(j >= coins[i]){
        T[i][j] = min(T[i-1][j],1+T[i][j-coins[i]])
    }else{
        T[i][j] = T[i-1][j];
    }
}
```

#### 5. 寻找组合

至此，填完表格我们已经接近完成了。接下来要寻找从表格中寻找硬币组合。🤔

与填表顺序相反，寻找组合从有下角开始。

首先需要明确的是如果  $T[i][j] == T[i-1][j]$ ，那么就向上搜索。根据图来分析：

j 表示零钱总额

	j = 0	j = 1	j = 2	j = 3	j = 4	j = 5	j = 6	j = 7	j = 8	j = 9	j = 10	j = 11
i = 0 1分	0	1	2	3	4	5	6	7	8	9	10	11
i = 1 2分	0	1	1	2	2	3	3	4	4	5	5	6
i = 2 5分	0	1	1	2	2	1	2	2	3	3	2	3
i = 3 6分	0	1	1	2	2	1	1	2	2	3	2	2



1. 定位到  $T[3][11]$ ，由于不存在  $T[i][j] == T[i-1][j]$ ，所以不用向上搜索，确定选中一个 6分 硬币。寻找组合的思路和填写  $T[i][j]$  的思路几乎是反过来的。
2. 选择一个6分硬币后，剩余的总额为  $11-6=5$ 。因此定位到  $T[3][5]$  中。由于  $T[3][5] == T[2][5]$ ，因此看图中的蓝色箭头，向上搜索，直到  $T[i][j] != T[i-1][j]$ 。
3. 定位到  $T[2][5]$  中，此时  $coins[i]$  为5分。选中5分硬币只有，剩余的总额为  $5-5=0$ 。
4. 当  $j=0$  时，搜索结束。由上面步骤确定选中的硬币组合为：1个5分，1个6分。

## 代码

以上就是整个最少硬币找零问题的分析思路。最终代码使用 JavaScript 实现，如果你的 Sublime 支持纯 JavaScript，你可以直接复制黏贴代码，command + b 直接运行查看结果，然后修改输入变量，查看更多情况下的输出结果。

```
//动态规划 -- 硬币找零问题
function minCoins(coins,total,n){
    var T = [];

    for(let i = 0;i<n;i++){
        T[i] = []
        for (let j=0;j<= total;j++){
            if(j == 0){
                T[i][j] = 0;
                continue;
            }
        }
    }
}
```

```

        if(i == 0){
            T[i][j] = j/coins[i]; //硬币找零一定要有个 最小面额1，否则会无解
        }else{
            if(j >= coins[i]){
                T[i][j] = Math.min(T[i-1][j],1+T[i][j-coins[i]])

            }else{
                T[i][j] = T[i-1][j];
            }
        }
    }

}

}

findValue(coins,total,n,T);

return T;

}

function findValue(coins,total,n,T){
    var i = n-1, j = total;
    while(i>0 && j >0){
        if(T[i][j]!=T[i-1][j]){
            //锁定位置,确定i,j值，开始找构成结果的硬币组合。 其实根据这种计算方法，只需要考虑当前
            //console.log(T[i][j]);
            break
        }else{
            i--;
        }
    }
}

var s = []; //存储组合结果

while(i >= 0 && j > 0 ){

    s.push(coins[i]);
    j=j-coins[i];
    if(j <= 0){
        break; //计算结束，退出循环
    }
    //如果 i == 0,那么就在第 0 行一直循环计算，直到 j=0即可
    if(i>0){
        //console.log(i);
        while(T[i][j] == T[i-1][j]){
            i--;
            if(i== 0){
                break;
            }
        }
    }
}

```

```
        }  
    }  
}  
  
console.log(s);  
//可以把数组s return 回去  
  
}
```

```
var coins = [1,2,5,6];  
var total = 11  
var n = coins.length  
  
console.log(minCoins(coins,total,n));
```

