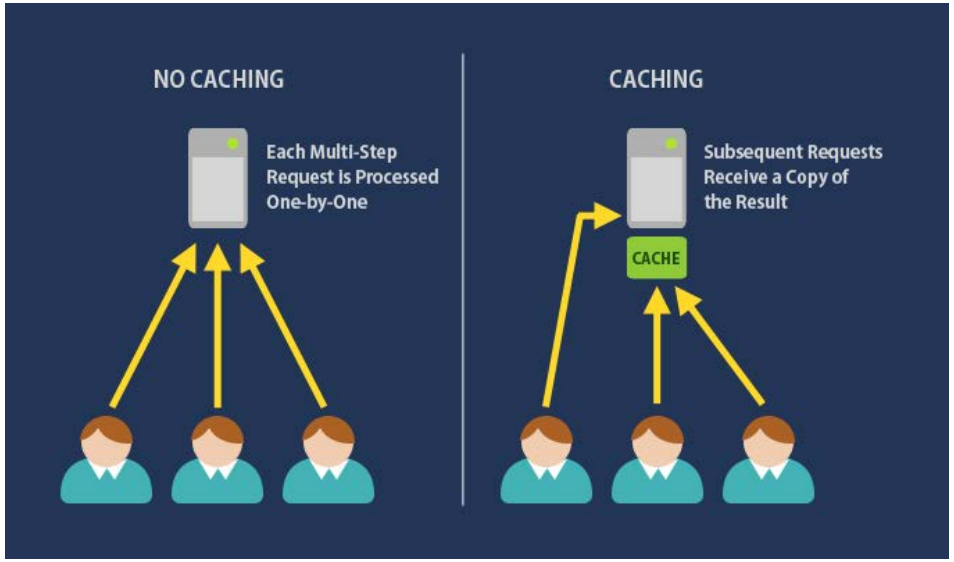
数据结构与算法（Python版）

找零兑换问题的动态规划解法

陈斌 北京大学 gischen@pku.edu.cn

找零兑换：动态规划解法

- 中间结果记录可以很好解决找零兑换问题
- 实际上，这种方法还不能称为动态规划，而是叫做“memoization（记忆化/**函数值缓存**）”的技术提高了递归解法的性能



找零兑换：动态规划解法

- 动态规划算法采用了一种**更有条理**的方式来得到问题的解
- 找零兑换的动态规划算法从**最简单**的“1分钱找零”的最优解开始，**逐步递加**上去，直到我们需要的找零钱数
- 在找零递加的过程中，设法保持**每一分钱的递加都是最优解**，一直加到求解找零钱数，自然得到最优解

找零兑换：动态规划解法

- 递加的过程能保持最优解的**关键**是，其**依赖于**更少钱数最优解的简单计算，而更少钱数的最优解已经得到了。
- **问题的最优解**包含了**更小规模子问题的最优解**，这是一个最优化问题能够用动态规划策略解决的**必要**条件。

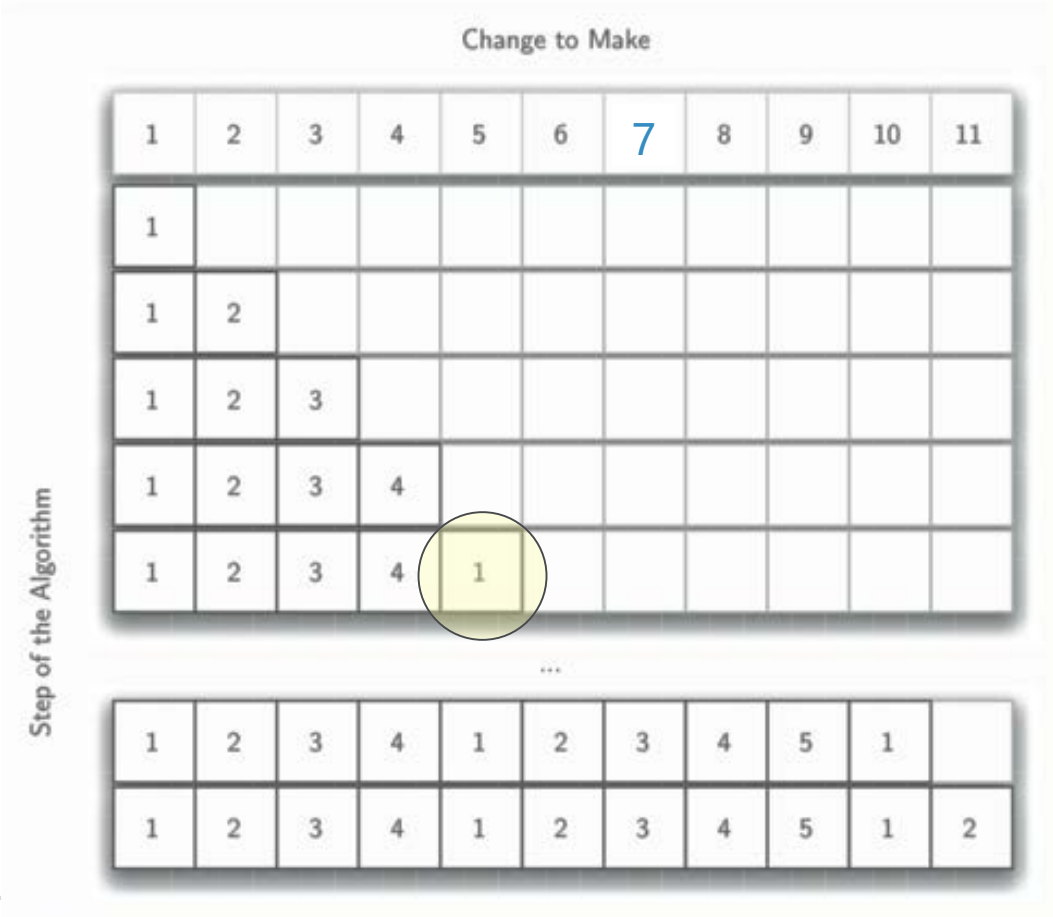
originalamount找零兑换问题具体来说就是：

$$numCoins = min \begin{cases} 1 + numCoins(originalamount - 1) \\ 1 + numCoins(originalamount - 5) \\ 1 + numCoins(originalamount - 10) \\ 1 + numCoins(originalamount - 25) \end{cases}$$

找零兑换：动态规划算法

- 采用动态规划来解决11分钱的兑换问题

  从1分钱兑换开始，逐步建立一个兑换表



找零兑换：动态规划解法

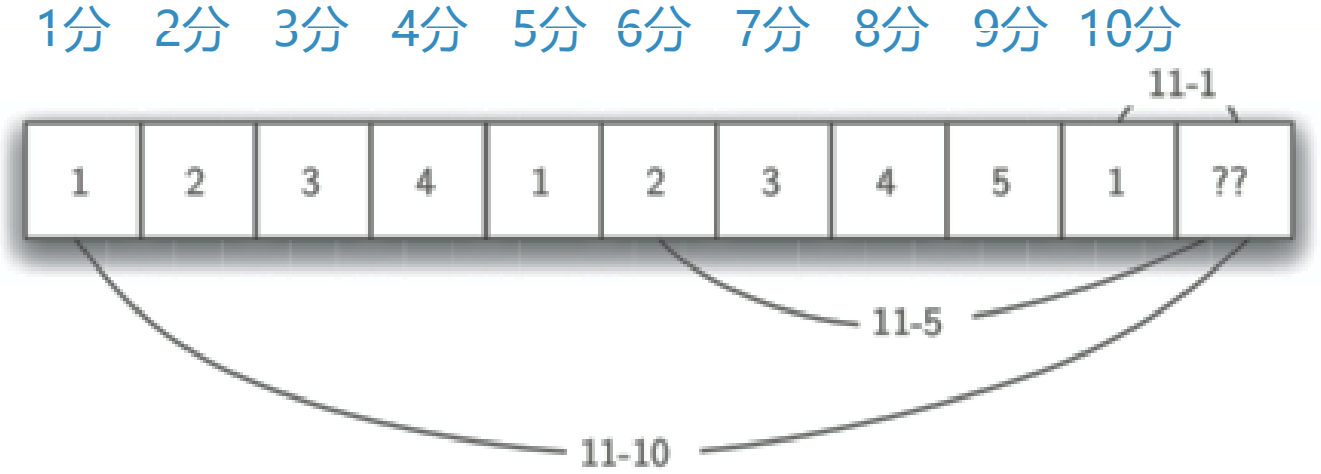
- 计算11分钱的兑换法，我们做如下几步：

  首先减去1分硬币，剩下10分钱查表最优解是1

  然后减去5分硬币，剩下6分钱查表最优解是2

  最后减去10分硬币，剩下1分钱查表最优解是1

- 通过上述最小值得到最优解：2个硬币



找零兑换：动态规划算法代码


```
1  def dpMakeChange(coinValueList, change, minCoins):
2      # 从1分开始到change逐个计算最少硬币数
3      for cents in range(1, change + 1):
4          # 1. 初始化一个最大值
5          coinCount = cents
6          # 2. 减去每个硬币，向后查最少硬币数，同时记录总的最少数
7          for j in [c for c in coinValueList if c <= cents]:
8              if minCoins[cents - j] + 1 < coinCount:
9                  coinCount = minCoins[cents - j] + 1
10         # 3. 得到当前最少硬币数，记录到表中
11         minCoins[cents] = coinCount
12     # 返回最后一个结果
13     return minCoins[change]
14
15
16 print(dpMakeChange([1, 5, 10, 21, 25], 63, [0] * 64))
```

循环结束，得到最优解

```
>>> %Run dpmakechange0.py
  3
>>>
```

找零兑换：动态规划算法扩展

- **我们注意到动态规划算法的dpMakeChange并不是递归函数**

  虽然这个问题是从递归算法开始解决，但最终我们得到一个更有条理的高效非递归算法

- **动态规划中最主要的思想是：**

  从最简单情况开始到达所需找零的循环

  其每一步都依靠以前的最优解来得到本步骤的最优解，直到得到答案。

找零兑换：动态规划算法扩展

- 前面的算法已经得到了最少硬币的数量，但没有返回硬币如何组合
- 扩展算法的思路很简单，只需要在生成最优解列表同时**跟踪记录**所选择的那个硬币币值即可
- 在得到最后的解后，减去选择的硬币币值，**回溯**到表格之前的部分找零，就能逐步得到每一步所选择的硬币币值

找零兑换：动态规划算法扩展代码


```
1  def dpMakeChange(coinValueList, change, minCoins, coinsUsed):
2      for cents in range(change + 1):  #
3          coinCount = cents
4          newCoin = 1  # 初始化一下新加硬币
5          for j in [c for c in coinValueList if c <= cents]:
6              if minCoins[cents - j] + 1 < coinCount:
7                  coinCount = minCoins[cents - j] + 1
8                  newCoin = j  # 对应最小数量，所减的硬币
9          minCoins[cents] = coinCount
10         coinsUsed[cents] = newCoin  # 记录本步骤加的1个硬币
11     return minCoins[change]
12
13
14 def printCoins(coinsUsed, change):
15     coin = change
16     while coin > 0:
17         thisCoin = coinsUsed[coin]
18         print(thisCoin)
19         coin = coin - thisCoin
```

找零兑换：动态规划算法扩展代码


```
21  amnt = 63
22  clist = [1, 5, 10, 21, 25]
23  coinsUsed = [0] * (amnt + 1)
24  coinCount = [0] * (amnt + 1)
25
26  print("Making change for", amnt, "requires")
27  print(dpMakeChange(clist, amnt, coinCount, coinsUsed), "coins")
28  print("They are:")
29  printCoins(coinsUsed, amnt)
30  print("The used list is as follows:")
31  print(coinsUsed)
```

```
>>> %Run dpmakechange.py
Making change for 63 requires
3 coins
They are:
21
21
21
The used list is as follows:
[1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 10, 1, 1,
1, 1, 5, 1, 1, 1, 1, 10, 21, 1, 1, 1, 25
, 1, 1, 1, 1, 5, 10, 1, 1, 1, 10, 1, 1,
1, 1, 5, 10, 21, 1, 1, 10, 21, 1, 1, 1,
25, 1, 10, 1, 1, 5, 10, 1, 1, 1, 10, 1,
10, 21]
```