数学建模模型算法精讲课——

动态规划

—— 江北老师

乾坤未定，

你我皆是黑马

动态规划

- 模型引出
- 模型原理
- 典型例题
- 代码求解

动态规划模型——模型引出

➢ **我们来看这样一个问题**

- 你有三种硬币，分别面值2元、5元和7元，每种硬币都有足够多
- 买一本书需要27元
- 如何用最少的硬币组合起来正好付清，不需要对方找钱

动态规划模型——模型引出

➢ **我们直觉会怎么想?**

你有三种硬币，分别面值2元、5元和7元，每种硬币都有足够多，买一本书需要27元，如何用最少的硬币组合起来正好付清，不需要对方找钱

- 最少的硬币组合➡尽量用面值大的硬币
  - ✓ 7+7+7+7=28
  - ✓ 7+7+7=21
  - ✓ 21+5=26
  - ✓ 这样显然拼不出来
- 改一下思路➡先用面值大的硬币,最后如果可以用一种硬币付清就行
  - ✓ 7+7+7=21
  - ✓ 21+2+2+2=27
  - ✓ 这次应该对了吧，其实正确答案是：7+5+5+5+5=27，5枚

动态规划模型——模型原理

MATH 数模加油站

➢ 动态规划

动态规划是运筹学的一个分支，通常用来解决**多阶段决策过程最优化问题**。动态规划的基本想法就是将原问题转换为一系列**相互联系的子问题**，然后通过**逐层地推**来求得最后的解。目前，动态规划常常出现在各类计算机算法竞赛或者程序员笔试面试中，在数学建模中出现的相对较少，但这个算法的思想在生活中非常实用，会对我们解决实际问题的思维方式有一定启发。

➢ 动态规划组成部分

- 一. 确定状态：解动态规划的时候需要开一个数组，数组的每个元素需要**明确代表什么**，类似于确定数学题中X、Y的含义
  - ✓ 最后一步
  - ✓ 子问题
- 二. 转移方程：把状态**表达成方程**
- 三. 初始条件和边界情况
- 四. 计算顺序

动态规划模型——典型例题

- 你有三种硬币，分别面值2元、5元和7元，每种硬币都有足够多，买一本书需要27元，如何用最少的硬币组合起来正好付清，不需要对方找钱

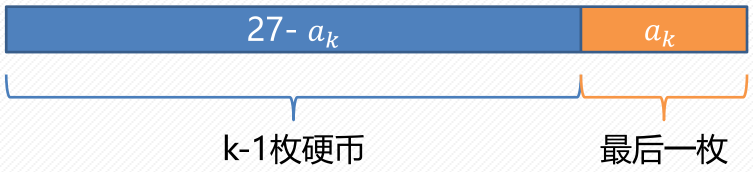
➢ 一. 确定状态

- **最后一步**

✓ 虽然我们不知道最优策略是什么，但是最优策略肯定是有k枚硬币，a_1 ,a_2 ,⋯ a_k加起来面值为27

✓ 所以一定存在有最后一枚硬币： a_k

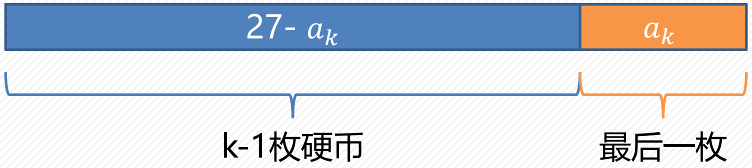
✓ 除了这枚硬币，前面硬币的面值加起来是27- a_k



动态规划模型——典型例题

➢ 一. 确定状态

- 最后一步



✓ 两个关键点

1）我们不关心前面的k-1枚硬币是怎么拼出27- a_k的（可能有很多种拼法），而且我们现在甚至还不知道a_k和k是多少，但我们可以确定前面的硬币拼出了27- a_k

2）因为是最优策略，所以拼出27- a_k的硬币数一定要最少，否则就不是最优策略

- 子问题

✓ **最少用多少枚硬币可以拼出27- a_k**

✓ 原问题是**最少用多少枚硬币可以拼出27**

✓ 我们将原问题可以转化成一个规模更小的子问题：27- a_k

- **状态：** 我们可以设状态$f(x)$ =最少用多少枚硬币拼出x

➢ 递归方法

- 现在我们还不知道最后那枚硬币a_k是多少，但它只能是2，5或7
  - ✓ 如果a_k=2，$f(27)=f(27-2)+1$
  - ✓ 如果a_k=5，$f(27)=f(27-5)+1$
  - ✓ 如果a_k=7，$f(27)=f(27-7)+1$
- 显然**只有这三种可能**
- 要求最小的硬币数，所以：

$$f(27)=\min\{f(27-2)+1,f(27-5)+1,f(27-7)+1\}$$

拼出27所需最少的硬币数

拼出25所需最少的硬币数+1

拼出22所需最少的硬币数+1

拼出20所需最少的硬币数+1

- 如果有学习过递归的，应该看得出来，这可以用递归来进行求解

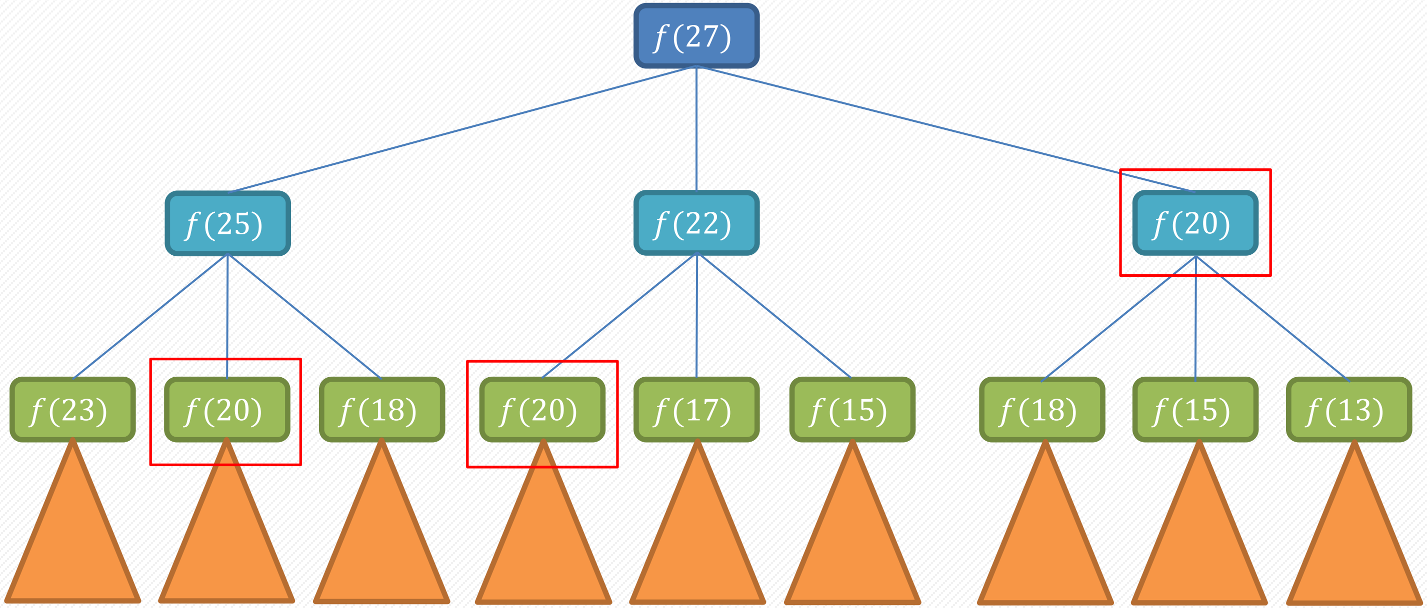
递归方法

$$f(27)=\min\{f(27-2)+1\,,f(27-5)+1\,,f(27-7)+1\}$$

- 递归python代码

```
def f(x):          # 递归函数，用于计算最少硬币数。
     if x == 0:
        return 0      # 如果金额为 0，则不需要任何硬币，直接返回 0
    res = float( ‘inf’ )      # 用一个很大的数表示无穷大，用于比较最小值
    if x >= 2:        # 如果金额大于等于 2 元，尝试使用一枚 2 元硬币
        res = min(f(x - 2) + 1, res)      # 递归调用 f 函数，并加上这一枚硬币
    if x >= 5:        # 如果金额大于等于 5 元，尝试使用一枚 5 元硬币
        res = min(f(x - 5) + 1, res)    # 递归调用 f 函数，并加上这一枚硬币
    if x >= 7:        # 如果金额大于等于 7 元，尝试使用一枚 7 元硬币
        res = min(f(x - 7) + 1, res)      # 递归调用 f 函数，并加上这一枚硬币
    return res       # 返回最少硬币数量，如果无法拼出来，则返回无穷大
```

- 递归方法的问题



动态规划模型——典型例题

二. 转移方程

- **状态：** 我们可以设状态$f[x]$ =最少用多少枚硬币拼出x
- 对于任意$\boldsymbol{x}$

$$f[x]=\min\{f[x-2]+1,f[x-5]+1,f[x-7]+1\}$$

拼出x所需最少的硬币数

拼出$x-2$所需最少的硬币数+1

拼出$x-5$所需最少的硬币数+1

拼出$x-7$所需最少的硬币数+1

三. 初始条件和边界情况

- 转移方程有两个问题：$x-2$，$x-5$，或$x-7$小于0怎么办？什么时候停下来？
  - ✓ 如果不能拼出Y，那么就定义$f[Y]$ =正无穷，例如$f[-1]=f[-2]=f[-3]=\cdots$ =正无穷
  - ✓ 所以$f[1]=\min\{f[-1]+1,f[-4]+1,f[-6]+1\}$=正无穷，表示拼不出来
  - ✓ 初始条件$f[0]=0$

动态规划模型——典型例题

四. 计算顺序

- 拼出x所需要的最少硬币数：$f[x]=\min\{f[x-2]+1,f[x-5]+1,f[x-7]+1\}$
- 初始条件$f[0]=0$
- 然后计算$f[1],f[2],\cdots,f[x]$
- 这样当我们计算到$f[x]$时，$f[x-2],f[x-5],f[x-7]$都已经算过了
- 我们来演示一下：

$\cdots$	$f[-1]$	$f[0]$	$f[1]$	$f[2]$	$f[3]$	$f[4]$	$f[5]$	$f[6]$	$\cdots$	$f[27]$
∞	∞	0	∞	1	∞	2	1	3	$\cdots$	5

- 每一步是算三次，算到27是第27步，故**时间复杂度**（需要的步数）为27×3（金额$\times$硬币种数）
- 递归时间复杂度$>>27\times3$

➢ 背包问题

有一个小偷去偷东西，他的背包可以容纳总重量为W的物品，现在有n件物品，每件物品的重量为w_i，价值为v_i，求能够放进背包的物品的最大价值。

- 状态：$dp[i][j]$表示前i件物品放入容量为j的背包中所获得的最大价值
- 状态转移方程：对于第i件物品，可以选择放或不放
  - ✓ 如果不放，那么$dp[i][j] = dp[i-1][j]$
  - ✓ 如果放，那么$dp[i][j] = dp[i-1][j-w_i] + v_i$
  - ✓ 选择获得最大价值的情况，即

$$dp[i][j] = max(dp[i-1][j], dp[i-1][j-w_i] + v_i)$$

- 初始条件：
  - ✓ $dp[0][0] = 0$，将前 0 个物品放入容量为 0 的背包中能获得的最大价值为 0
  - ✓ 如果容量为 0，则无法放入任何物品，$dp[i][0] = 0$
  - ✓ 如果没有物品可选，则无法放入任何物品，$dp[0][j] = 0$。
- 求解顺序：从第一个物品开始，求解到n
- 最终，$dp[n][W]$即为问题的解

动态规划模型——代码求解

MATH 数模加油站

➤ **凑硬币*py*代码**

```
def coinChange(n):
    dp = [float('inf')] * (n + 1)  # 初始化动态规划数组
    dp[0] = 0  # 找零金额为 0 时，需要 0 枚硬币
    for i in range(1, n + 1):
        if i >= 2:
            dp[i] = min(dp[i], dp[i - 2] + 1)
        if i >= 5:
            dp[i] = min(dp[i], dp[i - 5] + 1)
        if i >= 7:
            dp[i] = min(dp[i], dp[i - 7] + 1)
    if dp[n] != float('inf'):
        return dp[n]
    else:
        return -1
n=int(input('请输入要拼的金额：'))
res=coinChange(n)
print(res)
```

➢ 背包问题*py*代码

```
def knapsack(weights, values, capacity):
n = len(weights)  # 物品数量
    dp = [[0 for j in range(capacity + 1)] for i in range(n + 1)]  # 初始化动态规划数组
    # 动态规划求解过程
    for i in range(1, n + 1):
        for j in range(1, capacity + 1):
            if j < weights[i - 1]:  # 背包容量小于当前物品重量，不能选择当前物品
                dp[i][j] = dp[i - 1][j]
            else:  # 能选择当前物品，要选择价值更大的方案
                dp[i][j] = max(dp[i - 1][j], dp[i - 1][j - weights[i - 1]] + values[i - 1])
    return dp[n][capacity]
w = input('请输入物品的重量列表，用逗号分隔：')
v = input('请输入物品的价值列表，用逗号分隔：')
c = int(input('请输入背包的容量：'))
weights = [int(x) for x in w.split(',')]  # 将输入的字符串转换为整数列表
values = [int(x) for x in v.split(',')]
res = knapsack(weights, values, c)
print('最大价值为:', res)
```

欢迎关注数模加油站

THANKS