算法进阶

贪心算法

贪心算法

- 贪心算法（又称贪婪算法）是指，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的是在某种意义上的局部最优解。
- 贪心算法并不保证会得到最优解，但是在某些问题上贪心算法的解就是最优解。要会判断一个问题能否用贪心算法来计算。

找零问题

- 假设商店老板需要找零n元钱，钱币的面额有：100元、50元、20元、5元、1元，如何找零使得所需钱币的数量最少？

背包问题

- 一个小偷在某个商店发现有n个商品，第i个商品价值v_i元，重w_i千克。他希望拿走的价值尽量高，但他的背包最多只能容纳W千克的东西。他应该拿走哪些商品?
  - **0-1背包**：对于一个商品，小偷要么把它完整拿走，要么留下。不能只拿走一部分，或把一个商品拿走多次。（商品为金条）
  - **分数背包**：对于一个商品，小偷可以拿走其中任意一部分。（商品为金砂）

背包问题

- 举例：
  - 商品1：$v_1=60$ $w_1=10$
  - 商品2：$v_2=100$ $w_2=20$
  - 商品3：$v_3=120$ $w_3=30$
  - 背包容量：$W=50$

- 对于**0–1背包**和**分数背包**，贪心算法是否都能得到最优解？为什么？

拼接最大数字问题

- 有n个非负整数，将其按照字符串拼接的方式拼接为一个整数。如何拼接可以使得得到的整数最大？
  - 例：32,94,128,1286,6,71可以拼接除的最大整数为94716321286128

活动选择问题

- 假设有n个活动，这些活动要占用同一片场地，而场地在某时刻只能供一个活动使用。
- 每个活动都有一个开始时间s_i和结束时间f_i（题目中时间以整数表示）,表示活动在$[s_i, f_i)$区间占用场地。
- 问：安排哪些活动能够使该场地举办的活动的个数最多?

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

活动选择问题

- 贪心结论：最先结束的活动一定是最优解的一部分。
- 证明：假设a是所有活动中最先结束的活动，b是最优解中最先结束的活动。
  - 如果a=b，结论成立。
  - 如果a≠b，则b的结束时间一定晚于a的结束时间，则此时用a替换掉最优解中的b，a一定不与最优解中的其他活动时间重叠，因此替换后的解也是最优解。

动态规划

从斐波那契数列看动态规划

- 斐波那契数列：$F_n = F_{n-1} + F_{n-2}$
- 练习：使用**递归**和**非递归**的方法来求解斐波那契数列的第n项

钢条切割问题

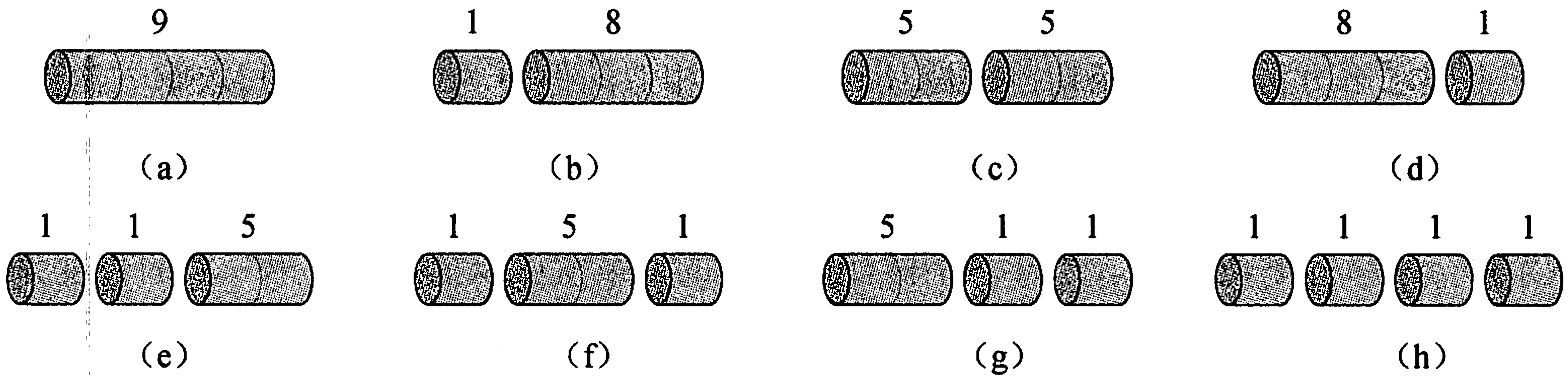
- 某公司出售钢条，出售价格与钢条长度之间的关系如下表：

长度i	1	2	3	4	5	6	7	8	9	10
价格p_i	1	5	8	9	10	17	17	20	24	30

- 问题：现有一段长度为n的钢条和上面的价格表，求切割钢条方案，使得总收益最大。

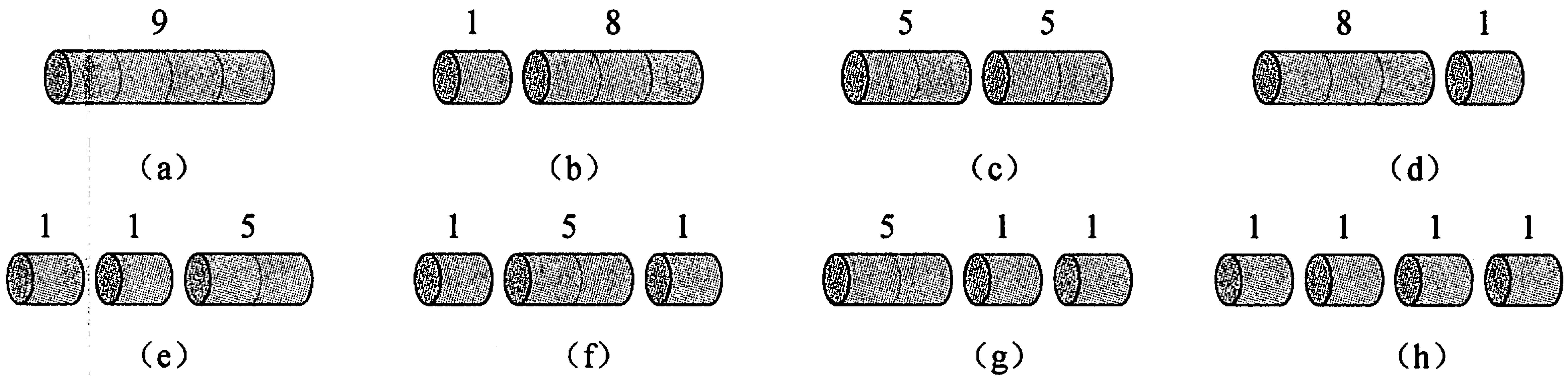
钢条切割问题

- 长度为4的钢条的所有切割方案如下：（c方案最优）



- 思考：长度为n的钢条的不同切割方案有几种?

钢条切割问题



长度 i	1	2	3	4	5	6	7	8	9	10
价格 p_i	1	5	8	9	10	17	17	20	24	30

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30

钢条切割问题——递推式

- 设长度为n的钢条切割后最优收益值为r_n，可以得出递推式：
  - $r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1)$
- 第一个参数p_n表示不切割
- 其他n-1个参数分别表示另外n-1种不同切割方案，对方案i=1,2,...,n-1
  - 将钢条切割为长度为i和n-i两段
  - 方案i的收益为切割两段的最优收益之和
- 考察所有的i，选择其中收益最大的方案

钢条切割问题——最优子结构

- 可以将求解规模为n的原问题，划分为规模更小的子问题：完成一次切割后，可以将产生的两段钢条看成两个独立的钢条切个问题。
- 组合两个子问题的最优解，并在所有可能的两段切割方案中选取组合收益最大的，构成原问题的最优解。
- 钢条切割满足**最优子结构**：问题的最优解由相关子问题的最优解组合而成，这些子问题可以独立求解。

钢条切割问题——最优子结构

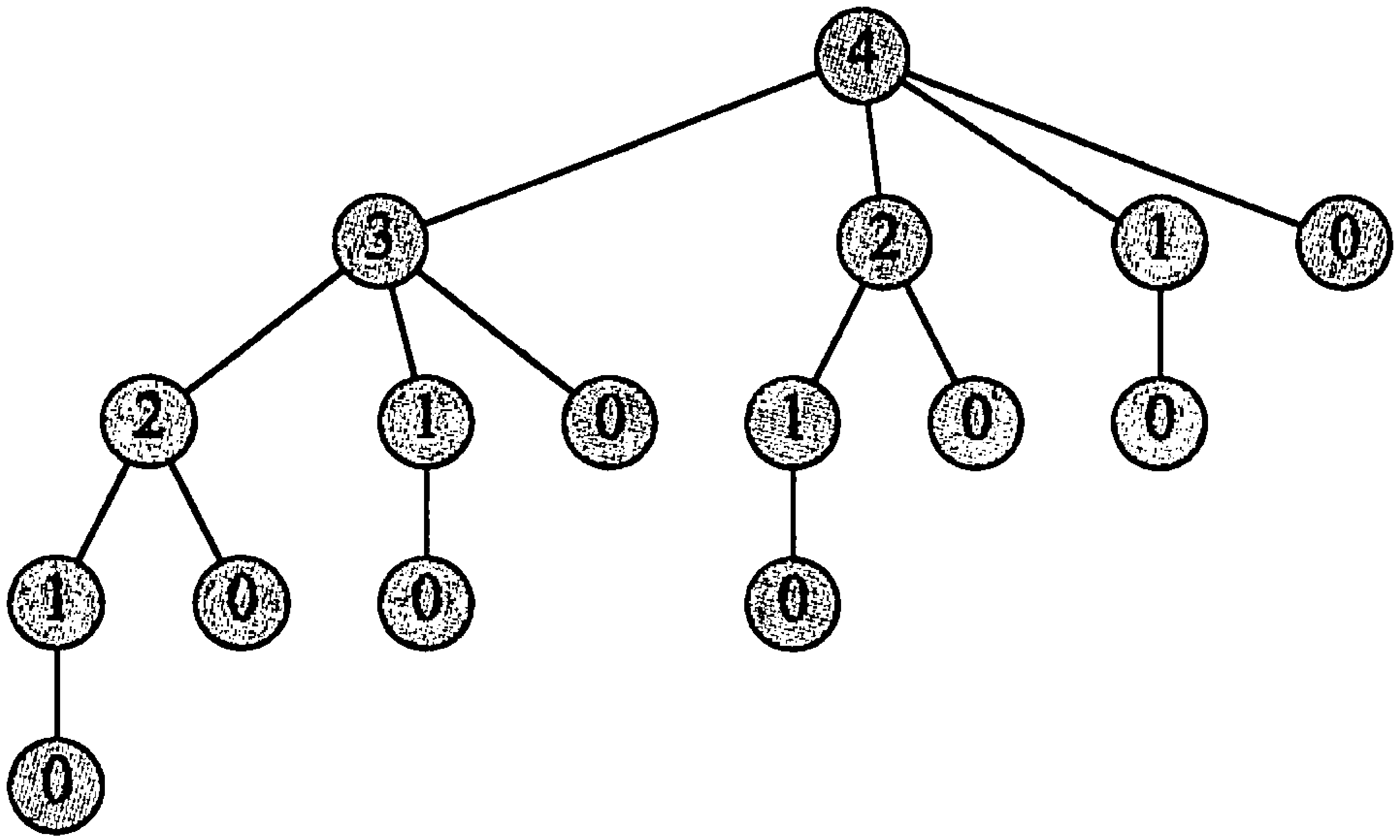
- 钢条切割问题还存在更简单的递归求解方法
  - 从钢条的左边切割下长度为i的一段，只对右边剩下的一段继续进行切割，左边的不再切割
  - 递推式简化为 $r_n = \max\limits_{1 \leq i \leq n} (p_i + r_{n-i})$
  - 不做切割的方案就可以描述为：左边一段长度为n，收益为p_n，剩余一段长度为0，收益为$r_0=0$。

钢条切割问题——自顶向下递归实现

```
def _cut_rod(p, n):
    if n == 0:
        return 0
    q = 0
    for i in range(1, n+1):
        q = max(q, p[i] + _cut_rod(p, n-i))
    return q
```

钢条切割问题——自顶向下递归实现

- 为何自顶向下递归实现的效率会这么差?
  - 时间复杂度 $O(2^n)$



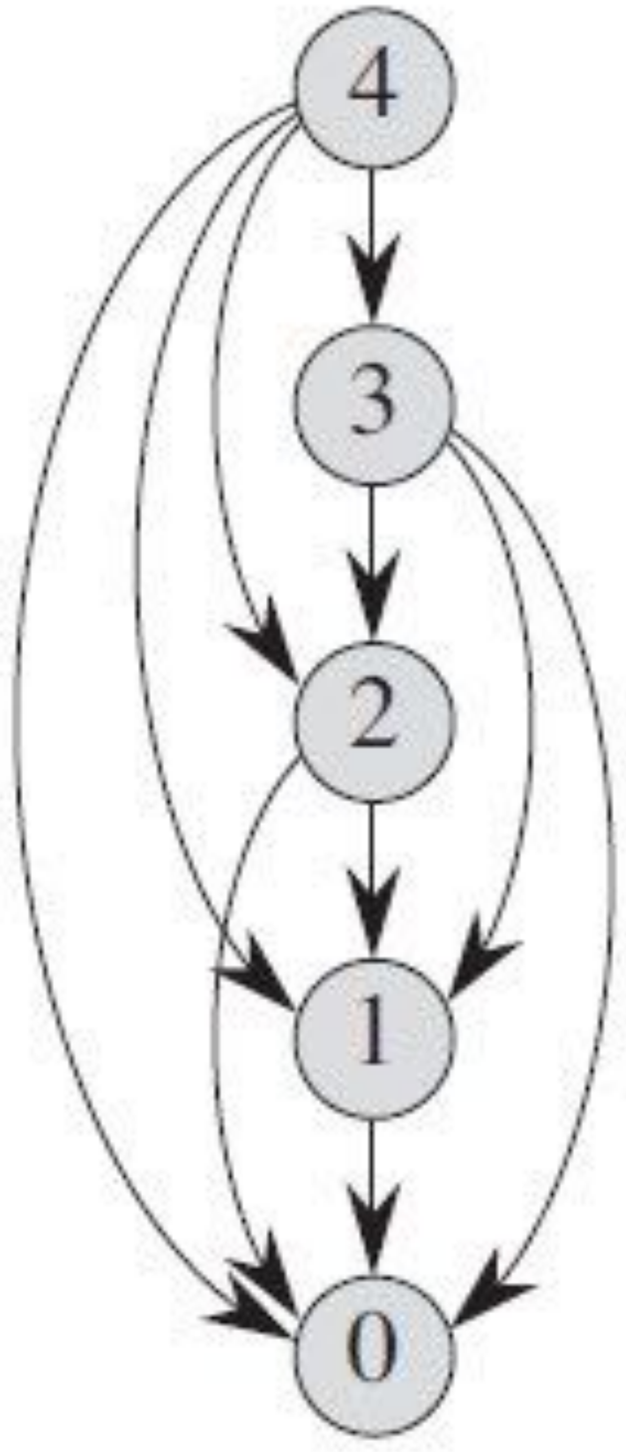
钢条切割问题——动态规划解法

- 递归算法由于重复求解相同子问题，效率极低
- 动态规划的思想：
  - 每个子问题只求解一次，保存求解结果
  - 之后需要此问题时，只需查找保存的结果

钢条切割问题——动态规划解法

```python
def cut_rod_dp(p, n):
    r = [0 for _ in range(n+1)]
    for j in range(1, n+1):
        q = 0
        for i in range(1, j+1):
            q = max(q, p[i] + r[j-i])
        r[j] = q
    return r[n]
```

- 时间复杂度：$O(n^2)$



钢条切割问题——重构解

- 如何修改动态规划算法，使其不仅输出最优解，还输出最优切割方案?

- 对每个子问题，保存切割一次时左边切下的长度

长度i	1	2	3	4	5	6	7	8	9	10
价格p_i	1	5	8	9	10	17	17	20	24	30

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

动态规划问题关键特征

- 什么问题可以使用动态规划方法?
  - **最优子结构**
    - 原问题的最优解中涉及多少个子问题
    - 在确定最优解使用哪些子问题时，需要考虑多少种选择
  - **重叠子问题**

最长公共子序列

- 一个序列的子序列是在该序列中删去若干元素后得 到的序列。
  - 例：“ABCD”和“BDF”都是“ABCDEFG”的子序列
- 最长公共子序列（LCS）问题：给定两个序列X和Y，求X和Y长度最大的公共子序列。
  - 例：X="ABBCBDE" Y="DBBCDB" LCS(X,Y)="BBCD"
- 应用场景：字符串相似度比对

最长公共子序列

- 思考：暴力穷举法的时间复杂度是多少?
- 思考：最长公共子序列是否具有最优子结构性质?

最长公共子序列

定理 15.1（LCS 的最优子结构） 令 $X=\langle x_1, x_2, \cdots, x_m\rangle$ 和 $Y=\langle y_1, y_2, \cdots, y_n\rangle$ 为两个序列，$Z=\langle z_1, z_2, \cdots, z_k\rangle$ 为 X 和 Y 的任意 LCS。

1. 如果 $x_m=y_n$，则 $z_k=x_m=y_n$ 且 Z_{k-1} 是 X_{m-1} 和 Y_{n-1} 的一个 LCS。
2. 如果 $x_m\neq y_n$，那么 $z_k\neq x_m$ 意味着 Z 是 X_{m-1} 和 Y 的一个 LCS。
3. 如果 $x_m\neq y_n$，那么 $z_k\neq y_n$ 意味着 Z 是 X 和 Y_{n-1} 的一个 LCS。

- 最优解的递推式：

$$c[i,j]=\begin{cases}0 & \text{若 } i=0 \text{ 或 } j=0\\ c[i-1,j-1]+1 & \text{若 } i,j>0 \text{ 且 } x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{若 } i,j>0 \text{ 且 } x_i\neq y_j\end{cases}$$

  - $c[i,j]$表示X_i和Y_j的LCS长度

最长公共子序列

- 例如：要求a="ABCBDAB"与b="BDCABA"的LCS：
  - 由于最后一位"B"≠"A"：
    - 因此LCS(a,b)应该来源于LCS(a[:-1],b)与LCS(a,b[:-1])中更大的那一个

最长公共子序列

i		j = 0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

$$c[i,j]=\begin{cases}0 & \text{若 } i=0 \text{ 或 } j=0\\ c[i-1,j-1]+1 & \text{若 } i,j>0 \text{ 且 } x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{若 } i,j>0 \text{ 且 } x_i\neq y_j\end{cases}$$

最长公共子序列

```
def lcs_length(x, y):
    m = len(x)
    n = len(y)
    c = [[0 for _ in range(n+1)] for _ in range(m+1)]
    for i in range(1, m+1):
        for j in range(1, n+1):
            if x[i-1] == y[j-1]:
                c[i][j] = c[i-1][j-1] + 1
            else:
                c[i][j] = max(c[i-1][j], c[i][j-1])
    return c[m][n]
```

最长公共子序列

- 思考：如何输出最长公共子序列的值?

欧几里得算法

最大公约数

- 约数：如果整数a能被整数b整除，那么a叫做b的倍数，b叫做a的约数。
- 给定两个整数a,b，两个数的所有公共约数中的最大值即为最大公约数（Greatest Common Divisor, GCD）。
- 例：12与16的最大公约数是4

最大公约数

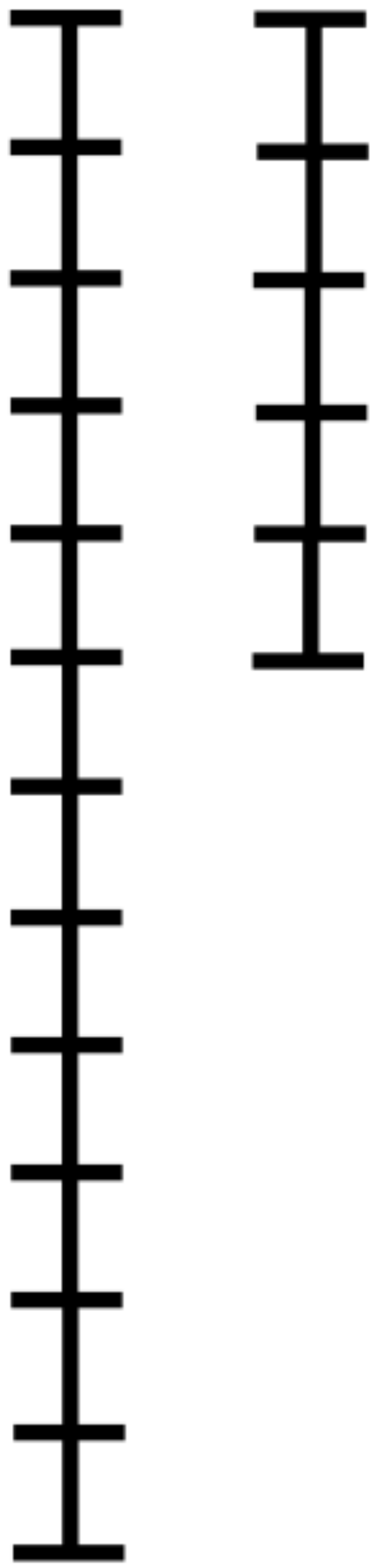
- 如何计算两个数的最大公约数：
  - 欧几里得：**辗转相除法（欧几里得算法）**
  - 《九章算术》：更相减损术

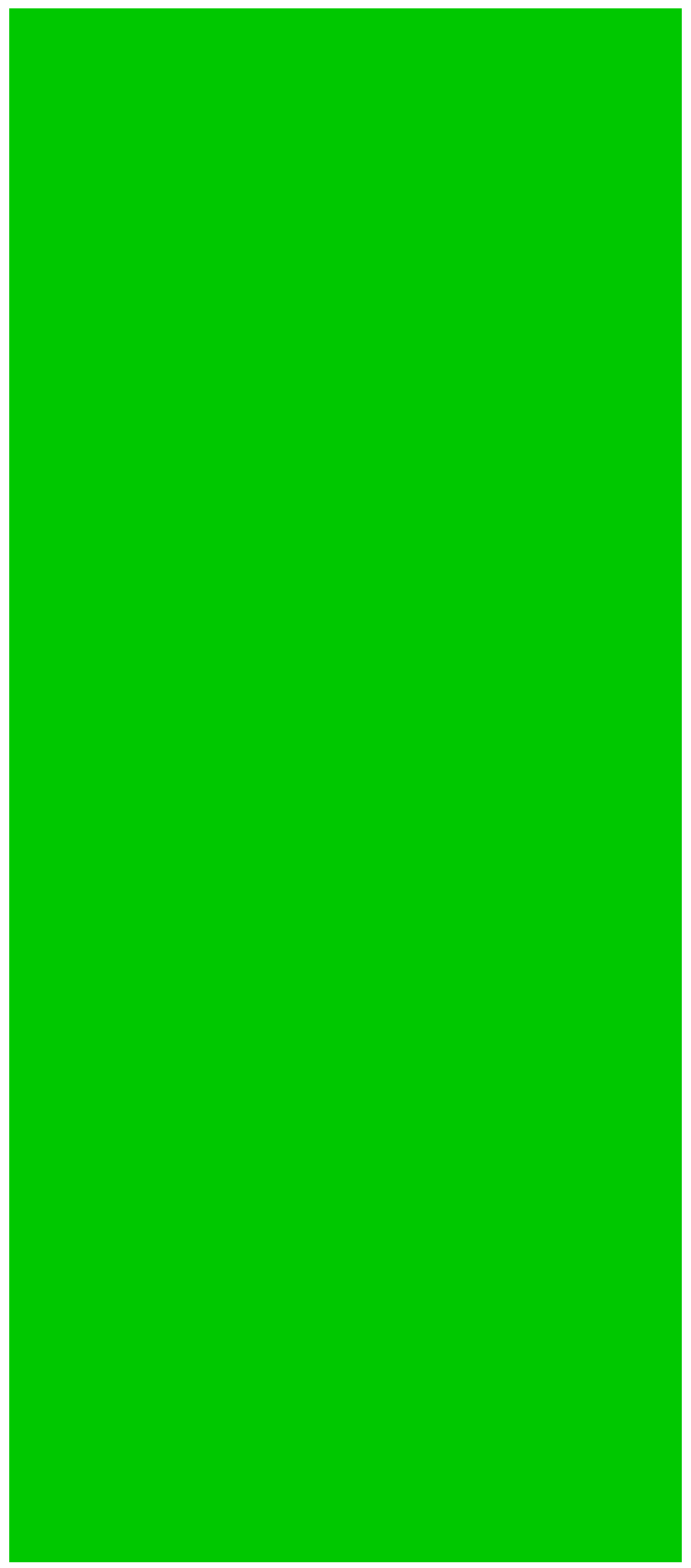
最大公约数——欧几里得算法

- 欧几里得算法：gcd(a, b) = gcd(b, a mod b)
  - 例：gcd(60, 21) = gcd(21, 18) = gcd(18, 3) = gcd(3, 0) = 3
- 证明略

最大公约数——欧几里得算法

gcd(252, 105)=21





gcd(1071, 462)=21

应用：实现分数计算

- 利用欧几里得算法实现一个分数类，支持分数的四则运算。

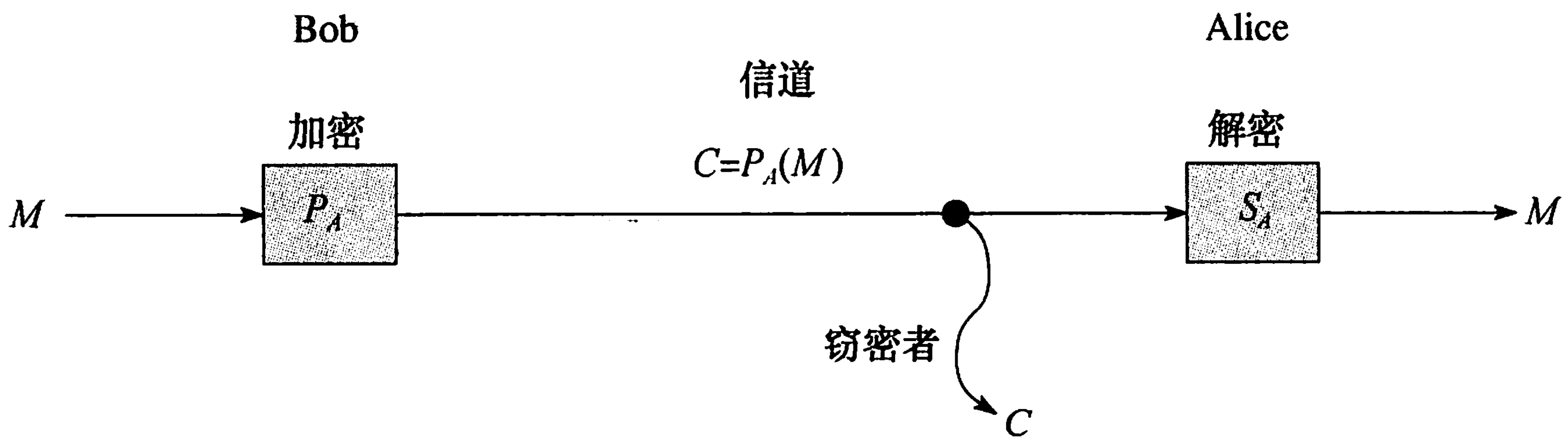
RSA加密算法简介

密码与加密

- 传统密码：加密算法是秘密的
- 现代密码系统：加密算法是公开的，密钥是秘密的
  - 对称加密
  - 非对称加密

RSA加密算法

- RSA非对称加密系统：
  - 公钥：用来加密，是公开的
  - 私钥：用来解密，是私有的



RSA加密算法过程

- 1. 随机选取两个质数p和q
- 2. 计算n=pq
- 3. 选取一个与φ(n)互质的小奇数e,φ(n)=(p–1)(q–1)
- 4. 对模φ(n),计算e的乘法逆元d，即满足 (e*d) mod φ(n) = 1
- 5. 公钥(e, n)   私钥(d, n)

RSA加密算法过程

- 加密过程： $c = (m^e) \bmod n$
- 解密过程： $m = (c^d) \bmod n$