课程回顾

- 动态规划原理：
  - 最优子结构
  - 注意事项（子问题独立性）
  - 重叠子问题
  - 重构最优解
  - 备忘
- 动态规划问题：最长公共子序列LCS

最长公共子序列LCS (续)

```
LCS_LENGTH(X, Y)
m ← X.length
n ← Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i ← 1 to m do
    c[i, 0] ← 0
for j ← 0 to n do
    c[0, j] ← 0
for i ← 1 to m do      // 依次考虑X_1, X_2, ..., X_m的前缀子列
    for j ← 1 to n do  // 依次考虑Y_1, Y_2, ..., Y_n的前缀子列
        if x_i = y_j           // 两个前缀子列的最后一位相同
            c[i, j] ← c[i-1, j-1] + 1
            b[i, j] ← "↖"
        elseif c[i-1, j] ≥ c[i, j-1] // 两个前缀子列的最后一位不同
            c[i, j] ← c[i-1, j]
            b[i, j] ← "↑"
        else c[i, j] ← c[i, j-1]
            b[i, j] ← "←"
return c and b
```

$\Theta(mn)$

最长公共子序列LCS (续)

	j	0	1	2	3	4	5	6
i		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

X=<*A*, *B*, *C*, *B*, *D*, *A*, *B*>

Y=<*B*, *D*, *C*, *A*, *B*, *A*>

最长公共子序列LCS (续)

4. 构造LCS

- 从$b[m, n]$开始根据箭头上溯至i=0或j=0即可
  - 当$b[i, j]$= "↖"时，有x_i=y_j是LCS的一个元素
  - 逆序构造出LCS，可用递归算法顺序打印

```
PRINT_LCS(b, X, i, j)
if i = 0 or j = 0
    return
if b[i, j] = "↖"
    PRINT_LCS(b, X, i-1, j-1)
    print x_i
elseif b[i, j] = "↑"
    PRINT_LCS(b, X, i-1, j)
else PRINT_LCS(b, X, i, j-1)
```

$O(m+n)$

每次递归调用

i和j至少一个会减少1

最长公共子序列LCS (续)

■算法改进

时空性能常数因子改变，运行时间渐近性能不变

- 可去掉表b——提升空间常数因子

  $c[i, j]$只依赖于$c[i\text{-}1, j\text{-}1]$, $c[i\text{-}1, j]$, $c[i, j\text{-}1]$

  可在$O(1)$判定是由哪项计算得到的，即可不依赖于b

  在$O(m+n)$时间重构LCS

- c只需要两行——提升空间性能

  c只需要当前行和前一行

  但无法构造出解，空间由$O(mn)$变为$O(n)$

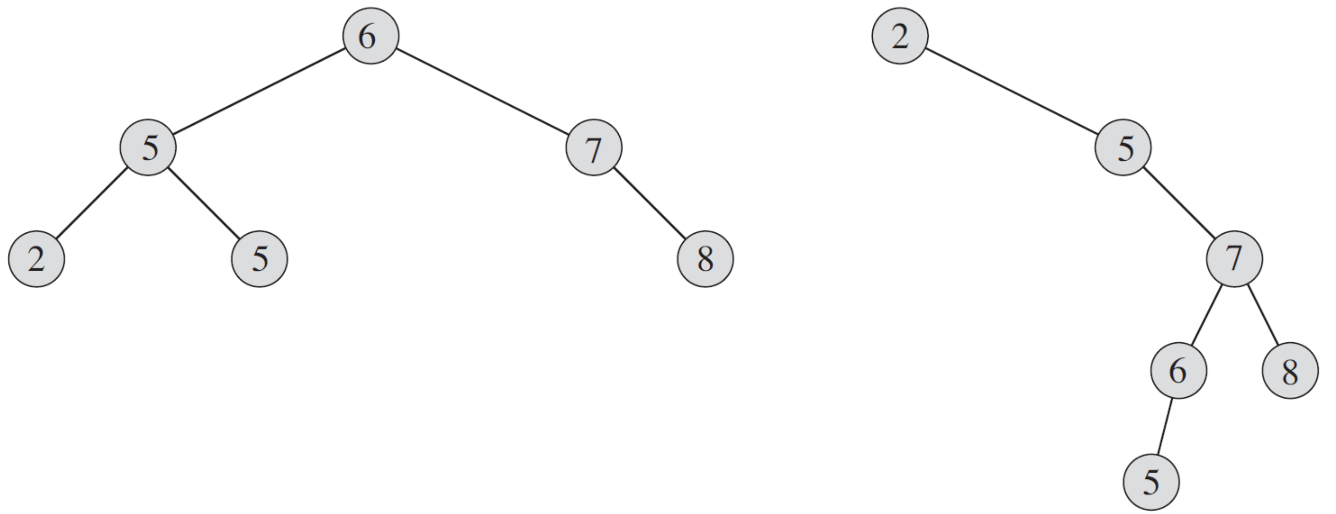
动态规划问题

- 钢条切割
- 矩阵链乘法的最优括号化
- 多边形的最佳三角剖分
- 最长公共子序列
- **最优二叉搜索树**
- 0-1背包

最优二叉搜索树

- 二叉搜索树（教材p161）：设x是二叉搜索树中的一个结点，如果y是x左子树中的一个结点，那么$y.key \leq x.key$；如果y是x右子树中的一个结点，那么$y.key \geq x.key$
  - 若它的左子树不空，则左子树上所有结点的值均小于等于它的根结点的值
  - 若它的右子树不空，则右子树上所有结点的值均大于等于它的根结点的值
  - 它的左、右子树也分别为二叉搜索树

最优二叉搜索树 (续)



(a)　　　　(b)

图 12-1　二叉搜索树。对任何结点 x，其左子树中的关键字最大不超过 $x.key$，其右子树中的关键字最小不低于 $x.key$。不同的二叉搜索树可以代表同一组值的集合。大部分搜索树操作的最坏运行时间与树的高度成正比。(a)一棵包含 6 个结点、高度为 2 的二叉搜索树。(b)一棵包含相同关键字、高度为 4 的低效二叉搜索树

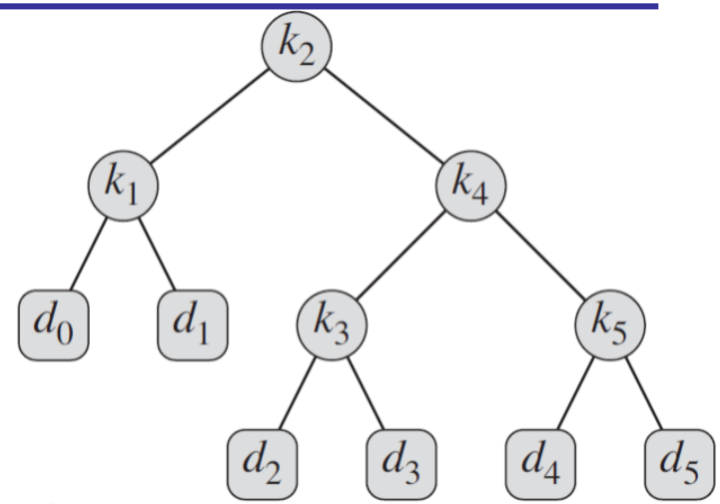
最优二叉搜索树 (续)

- 二叉搜索树上的基本操作花费时间与树高成正比
  - n个结点的完全二叉树：最坏运行时间$\Theta(\lg n)$
  - n个结点的线性链：最坏运行时间$\Theta(n)$
  - 随机构造的二叉搜索树期望高度为$O(\lg n)$，基本操作的平均运行时间$\Theta(\lg n)$

最优二叉搜索树 (续)

- **最优二叉搜索树**：给定一个n个不同关键字的已排序的序列$K=<k_1, k_2, ..., k_n>$ $(k_1<k_2<...<k_n)$，构造一棵**期望搜索代价最小**的二叉搜索树
  - 每个关键字k_i：以概率p_i搜索
  - 搜索的值不在K中：$n+1$个伪关键字d_i $(i=0, 1, ..., n)$，分别以q_i概率搜索
    - d_0：所有小于k_1的值
    - d_n：所有大于k_n的值
    - d_i $(i=1, 2, ..., n-1)$：所有大于k_i小于k_{i+1}的值

最优二叉搜索树 (续)



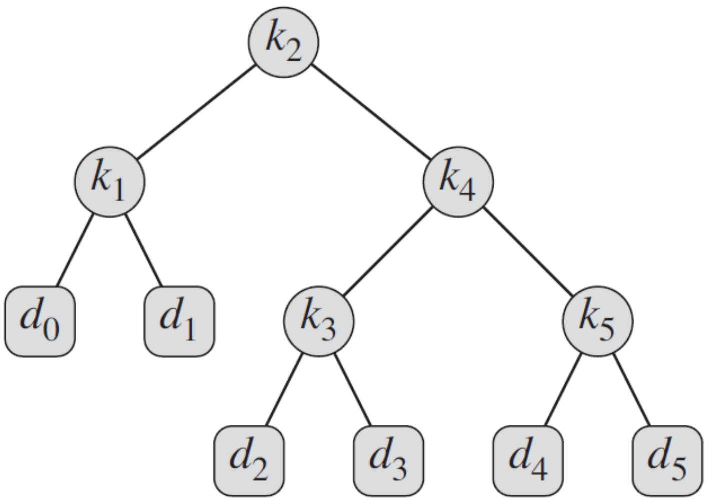
■要么搜索成功要么搜索失败：

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1$$

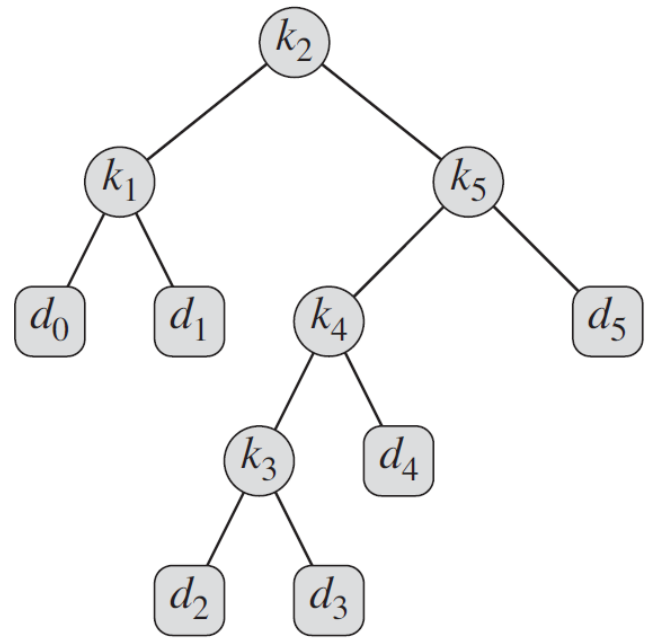
■二叉搜索树进行一次搜索的期望代价：

$$\begin{aligned}
\mathrm{E}[\text{search cost in T}] &= \sum_{i=1}^{n} (\mathrm{depth}_{\mathrm{T}}(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\mathrm{depth}_{\mathrm{T}}(d_i) + 1) \cdot q_i \\
&= 1 + \sum_{i=1}^{n} \mathrm{depth}_{\mathrm{T}}(k_i) \cdot p_i + \sum_{i=0}^{n} \mathrm{depth}_{\mathrm{T}}(d_i) \cdot q_i
\end{aligned}$$

最优二叉搜索树 (续)



(a)



(b)

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

期望搜索代价：
(a)2.80
(b)2.75（最优）

最优二叉搜索树 (续)

- 最优二叉搜索树：
  - 不一定高度最矮
  - 概率最高的关键字不一定出现在根结点
- n个结点的二叉树数量为$\Omega(4^n/n^{3/2})$
- 穷举法需要检查指数棵二叉树
- 动态规划求解：
  1. 最优二叉搜索树的结构
  2. 递归算法
  3. 计算最优二叉搜索树的期望搜索代价
  4. 构造最优二叉搜索树

最优二叉搜索树 (续)

1. 最优二叉搜索树的结构

- 考虑一棵二叉搜索树的任意子树T’：
  包含连续关键字$k_i, \ldots, k_j$, $1 \le i \le j \le n$
  叶节点是伪关键字$d_{i-1}, \ldots, d_j$
- 若T’是最优二叉搜索树T的子树，则T’也是最优的
  “剪切-粘贴” + 反证法

- 是否完全定义了子问题空间？—— “空子树”
  $i \ge 1$， $i-1 \le j \le n$
  $j=i-1$时，子树不包含实际关键字，只包含伪关键字d_{i-1}

最优二叉搜索树 (续)

2. 递归算法

- 子问题：求解包含关键字$k_i, \ldots, k_j$的最优二叉搜索树，其中$i \geq 1$且$i\text{-}1 \leq j \leq n$
- $e[i, j]$：该最优二叉搜索树中进行一次搜索的期望代价
  - $j=i\text{-}1$：子树只包含伪关键字$d_{i\text{-}1}$
    期望搜索代价$e[i, i\text{-}1]=q_{i\text{-}1}$
  - $j \geq i$：需从$k_i, \ldots, k_j$中选择一个根结点k_r，
    构造包含$k_i, \ldots, k_{r\text{-}1}$的最优二叉搜索树作为左子树
    构造包含$k_{r+1}, \ldots, k_j$的最优二叉搜索树作为右子树

最优二叉搜索树 (续)

2. 递归算法

- $e[i, j]$：该最优二叉搜索树中进行一次搜索的期望代价
  - 子树所增加的期望搜索代价：所有概率之和

$$w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

  - 若k_r为k_i, …, k_j最优二叉搜索树的根结点，则有

$$e[i,j] = p_r + (e[i,r-1] + w(i,r-1)) + (e[r+1,j] + w(r+1,j))$$

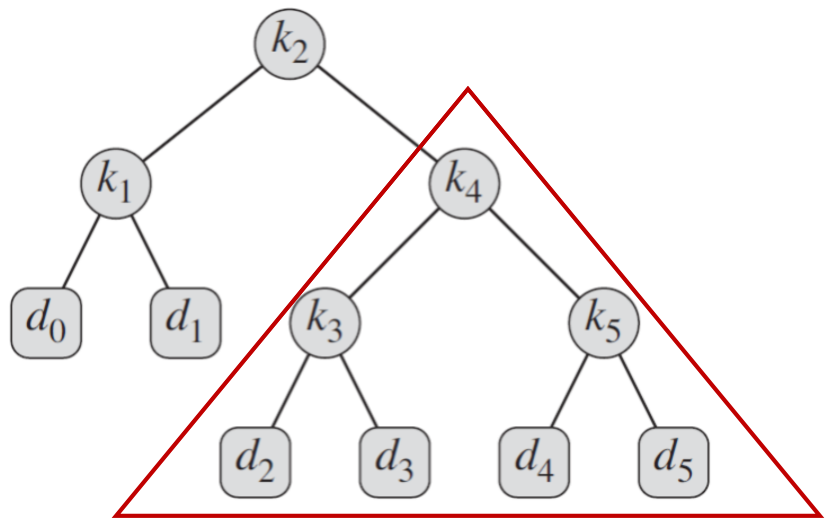
注意到 $w(i,j) = w(i,r-1) + p_r + w(r+1,j)$

因此 $e[i,j] = e[i,r-1] + e[r+1,j] + w(i,j)$

最优二叉搜索树 (续)

2. 递归算法

➢例：



$$e[1,5] = p_2 + \left(e[1,1] + \underbrace{\sum_{l=1}^{1} p_l + \sum_{l=0}^{1} q_l}_{w(1,1)} \right) + \left(\boxed{e[3,5]} + \underbrace{\sum_{l=3}^{5} p_l + \sum_{l=2}^{5} q_l}_{w(3,5)} \right)$$

$$= e[1,1] + e[3,5] + w(1,5) \qquad \text{其中 } p_2 + w(1,1) + w(3,5) = w(1,5)$$

最优二叉搜索树 (续)

2. 递归算法

➢最终递归公式：

$$e[i,j]=\begin{cases} q_{i-1}, & j=i-1, \\ \min\limits_{i\le r\le j}\{e[i,r-1]+e[r+1,j]+w(i,j)\}, & i\le j \end{cases}$$

3. 计算最优二叉搜索树的期望搜索代价

➢*root*$[i, j]$：包含$k_i, \ldots, k_j$ ($1\le i\le j\le n$) 的最优二叉搜索树根结点k_r的下标r

➢保存$w[i, j]$的值避免重复计算：

$$w[i,j]=\begin{cases} q_{i-1}, & j=i-1, \\ w[i,j-1]+p_j+q_j, & i\le j \end{cases}$$

最优二叉搜索树 (续)

```
OPTIMAL_BST(p, q, n)
let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
for i ← 1 to n+1 do
    e[i, i-1] ← q_{i-1}
    w[i, i-1] ← q_{i-1}
for l ← 1 to n do
    for i ← 1 to n-l+1 do
        j ← i + l - 1
        e[i, j] ← ∞
        w[i, j] ← w[i, j-1] + p_j + q_j
        for r ← i to j do
            t ← e[i, r-1] + e[r+1, j] + w[i, j]
            if t < e[i, j]
                e[i, j] ← t
                root[i, j] ← r
return e and root
```

$\Theta(n^3)$

最优二叉搜索树 (续)

3. 计算最优二叉搜索树的期望搜索代价——算法改进
   - 教材p231练习15.5-4：对所有$1 \le i < j \le n$，存在最优二叉搜索树，其根满足$root[i, j\text{-}1] \le root[i, j] \le root[i+1, j]$
   - 运行时间减少为$\Theta(n^2)$

最优二叉搜索树 (续)

```
OPTIMAL_BST(p, q, n)
let e[1..n+1, 0..n], w[1..n+1, 0..n], and root[1..n, 1..n] be new tables
for i ← 1 to n+1 do
    e[i, i-1] ← q_{i-1}
    w[i, i-1] ← q_{i-1}
for l ← 1 to n do
    for i ← 1 to n-l+1 do
        j ← i + l - 1
        e[i, j] ← ∞
        w[i, j] ← w[i, j-1] + p_j + q_j
        for r ← root[i, j-1] to root[i+1, j] do
            t ← e[i, r-1] + e[r+1, j] + w[i, j]
            if t < e[i, j]
                e[i, j] ← t
                root[i, j] ← r
return e and root
```

$\Theta(n^2)$

最优二叉搜索树 (续)

4. 构造最优二叉搜索树

- 教材p230-231练习15.5-1（先序遍历输出BST）

CONSTRUCT_OPTIMAL_BST(*root*)
return RECURSIVE_CONSTRUCT_OPTIMAL_BST(*root*, 1, *n*, 0)

RECURSIVE_CONSTRUCT_OPTIMAL_BST(*root*, *i*, *j*, *indicator*)
if $i \le j$
    if *indicator* = 0
        print “k”$_{root[i,\,j]}$“为根”
    elseif *indicator* = 1
        print “k”$_{root[i,\,j]}$“为k”$_{j+1}$“的左孩子”
    else print “k”$_{root[i,\,j]}$“为k”$_{i\text{-}1}$“的右孩子”
    RECURSIVE_CONSTRUCT_OPTIMAL_BST(*root*, *i*, *root*[*i*, *j*]-1, 1)
    RECURSIVE_CONSTRUCT_OPTIMAL_BST(*root*, *root*[*i*, *j*]+1, *j*, 2)
else
    if *indicator* = 1
        print “d”$_j$“为k”$_{j+1}$“的左孩子”
    else print “d”$_j$“为k”$_{i\text{-}1}$“的右孩子”

动态规划问题

- 钢条切割
- 矩阵链乘法的最优括号化
- 多边形的最佳三角剖分
- 最长公共子序列
- 最优二叉搜索树
- 0-1背包

0-1背包

- 0-1背包问题（0-1 knapsack problem，教材p243）：给定n个物品和一个容量为W的背包，第i个物品价值为v_i、重量为w_i，应当如何选择装入背包的物品使得总价值最大？（参数均为正整数）

- 0-1整数规划问题

穷举法共2^n种装包方式

$$
\begin{aligned}
\max & \sum_{k=1}^{n} v_k x_k \\
\text{s.t.} & \sum_{k=1}^{n} w_k x_k \le W, \\
& x_k \in \{0,1\}, \quad k = 1, \ldots, n.
\end{aligned}
$$

0-1背包 (续)

1. 0-1背包子问题（最优子结构）

$$\max \sum_{k=1}^{\boxed{i}} v_k x_k$$

$$\text{s.t.} \quad \sum_{k=1}^{\boxed{i}} w_k x_k \le \boxed{j},$$

$$x_k \in \{0,1\}, \quad k = 1, \ldots, \boxed{i}.$$

- 子问题：求解背包容量为j，可选物品编号为1, 2, ..., i时的0-1背包问题
  考虑是否放入编号为i的物品：
  $m[i, j] = \max(\underbrace{m[i\text{-}1, j]}_{\text{不装入}},\quad \underbrace{m[i\text{-}1, j\text{-}w_i] + v_i}_{\text{装入}})$

0-1背包 (续)

2. 递归地定义子问题最优解

➢ $m[i, j]$：背包容量为j，可选物品编号为1, 2, ..., i时的0-1背包问题的最优解

$$m[i,j] = \begin{cases} 0, & i = 0 \text{ or } j = 0, \\ m[i-1,j], & 0 < j < w_i, \\ \max(m[i-1,j], m[i-1,j-w_i] + v_i), & w_i \leq j \leq W. \end{cases}$$

0-1背包 (续)

■例：

	物品1	物品2	物品3	物品4	物品5
重量w_i	4	5	4	3	10
价值v_i	9	10	9	2	24
背包容量W	13				

➢$m[5, 13]$：考虑是否选择物品5

- 选择：$m[4, 13\text{-}10]+24 = m[4, 3]+24$
- 不选择：$m[4, 13]$

0-1背包 (续)

	物品1	物品2	物品3	物品4	物品5
重量w_i	4	5	4	3	10
价值v_i	9	10	9	2	24
背包容量W	13				

$m[i, j]$	j=0	1	2	3	4	5	6	7	8	9	10	11	12	13
i = 0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	9	9	9	9	9	9	9	9	9	9
2	0	0	0	0	9	10	10	10	10	19	19	19	19	19
3	0	0	0	0	9	10	10	10	18	19	19	19	19	28
4	0	0	0	2	9	10	10	11	18	19	19	20	21	28
5	0	0	0	2	9	10	10	11	18	19	24	24	24	28

$$m[i, j] = \max(m[i\text{-}1, j], \quad m[i\text{-}1, j\text{-}w_i] + v_i)$$

0-1背包 (续)

3. 自底向上计算最优解的值

- $s[i, j]$：记录背包容量为j时编号为i的物品是否选择
  - 1——选择
  - 0——不选择

$\Theta(nW)$

```
KNAPSACK(w, v, W)
n ← v.length
let m[0..n, 0..W] and s[1..n, 1..W] be new tables
for i ← 1 to n do
    m[i, 0] ← 0
for j ← 1 to W do
    m[0, j] ← 0
for i ← 1 to n do
    for j ← 1 to W do
        if j ≥ w[i] and m[i-1, j] < m[i-1, j-w[i]] + v[i]
            m[i, j] ← m[i-1, j-w[i]] + v[i]
            s[i, j] ← 1
        else m[i, j] ← m[i-1, j]
            s[i, j] ← 0
return m and s
```

0-1背包 (续)

4. 构造最优解

```
CONSTRUCT_KNAPSACK(w, W, s)
n ← w.length;  K ← W
for i ← n downto 1 do
    if K ≤ 0
        return
    if s[i, K] = 1
        print "选择编号"i"物品"
        K ← K – w[i]
```

本章小结

- 动态规划求解步骤：定义子问题、递归定义子问题最优解、自底向上求解、构建原问题最优解
- 动态规划要素：最优子结构、重叠子问题
- 求解方法：自底向上求解、带备忘的自顶向下方法求解
- 具体应用：钢条切割、矩阵链乘法最优括号化、多边形最佳三角剖分、最长公共子序列、最优二叉搜索树、0-1背包