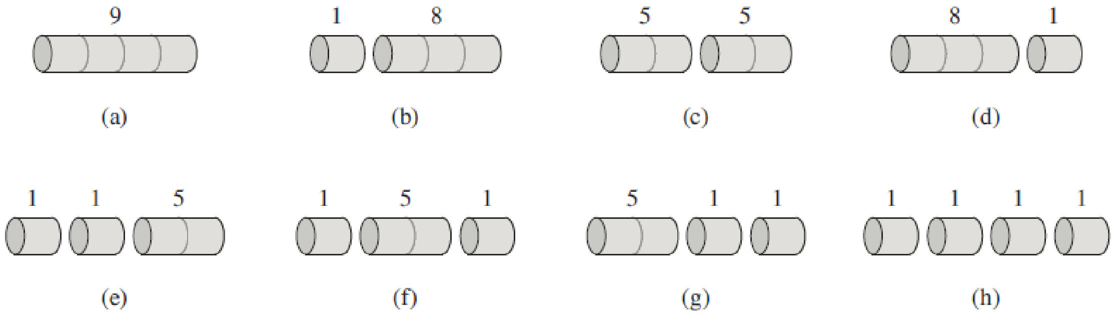
15 Dynamic Programming

- Assembly lines scheduling (ALS) （流水线调度）
- Steel rod cutting（钢条、钢管切割）
- Matrix-chain multiplication（矩阵链相乘，矩阵连乘）
- Characteristics of dynamic programming

  （动态规划法的特征）
- Longest common subsequence （最长相同子序列）
- Optimal binary search trees （最优二叉搜索树）

songyou@buaa.edu.cn

15.1 Rod cutting（钢管切割）

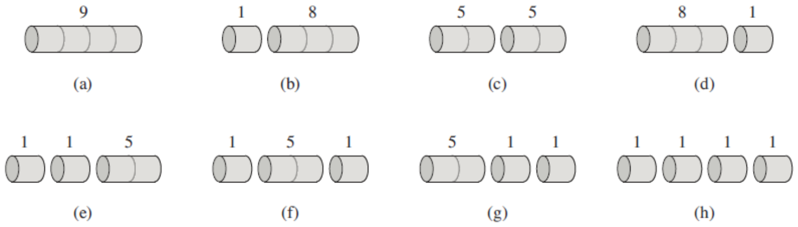


length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Step 1: The structure of the optimal decomposition（最优子结构）

1节卖1元，2节卖5元，3节卖8元，……，10节卖30元，
如何切割，使得卖出的钱最多?

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



$$
\begin{aligned}
r_1 &= 1 && \text{from solution } 1 = 1 \quad \text{(no cuts)},\\
r_2 &= 5 && \text{from solution } 2 = 2 \quad \text{(no cuts)},\\
r_3 &= 8 && \text{from solution } 3 = 3 \quad \text{(no cuts)},\\
r_4 &= 10 && \text{from solution } 4 = 2 + 2,\\
r_5 &= 13 && \text{from solution } 5 = 2 + 3,\\
r_6 &= 17 && \text{from solution } 6 = 6 \quad \text{(no cuts)},\\
r_7 &= 18 && \text{from solution } 7 = 1 + 6 \text{ or } 7 = 2 + 2 + 3,\\
r_8 &= 22 && \text{from solution } 8 = 2 + 6,\\
r_9 &= 25 && \text{from solution } 9 = 3 + 6,\\
r_{10} &= 30 && \text{from solution } 10 = 10 \quad \text{(no cuts)}.
\end{aligned}
$$

钢管长9米，切成两段卖。

钢管长10米，整卖最好。

Optimal substructure?

长度为 n 的钢管，一种最优的切割方式：

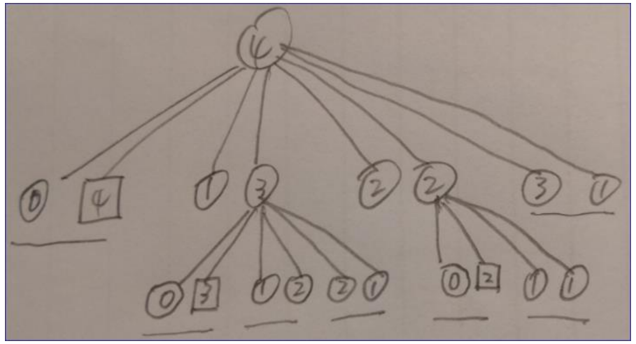
$$n = i_1 + i_2 + \cdots + i_k$$

切割为 k 段，$1 \le k \le n$, 每段的长度为 $i_1, i_2, \ldots, i_k$,
最大收益为：

$$r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$$

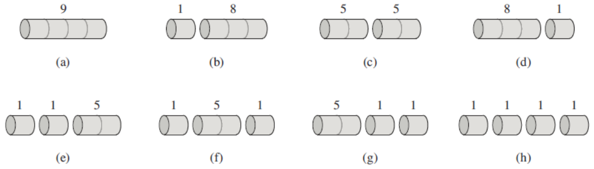
r_n : n 米长钢管的最优切割方式（最优：价格最高，收益最大）

$$r_n = \max(\underbrace{p_n}_{r_0 + p_n}, r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1) \qquad (15.1)$$



Step 1: The structure of the optimal decomposition（最优子结构）

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



$$
\begin{aligned}
r_1 &= 1 && \text{from solution } 1 = 1 \quad \text{(no cuts)}\,,\\
r_2 &= 5 && \text{from solution } 2 = 2 \quad \text{(no cuts)}\,,\\
r_3 &= 8 && \text{from solution } 3 = 3 \quad \text{(no cuts)}\,,\\
r_4 &= 10 && \text{from solution } 4 = 2+2\,,\\
r_5 &= 13 && \text{from solution } 5 = 2+3\,,\\
r_6 &= 17 && \text{from solution } 6 = 6 \quad \text{(no cuts)}\,,\\
r_7 &= 18 && \text{from solution } 7 = 1+6 \text{ or } 7 = 2+2+3\,,\\
r_8 &= 22 && \text{from solution } 8 = 2+6\,,\\
r_9 &= 25 && \text{from solution } 9 = 3+6\,,\\
r_{10} &= 30 && \text{from solution } 10 = 10 \quad \text{(no cuts)}\,.
\end{aligned}
$$

Optimal substructure? 例:

原问题 r_7： 一个最优切割（最优解）是 2+2+3（对应的最优值为18）；

子问题r_4：(因为 r_4+r_3 ∈ r_7)，那么，可行解 2+2一定是 r_4 的一个最优切割（最优解）。

最优子结构：原问题的解（2+2+3）包括子问题的解（2+2）。

长度为 n 的钢管，一种最优的切割方式：

$$n = i_1 + i_2 + \cdots + i_k$$

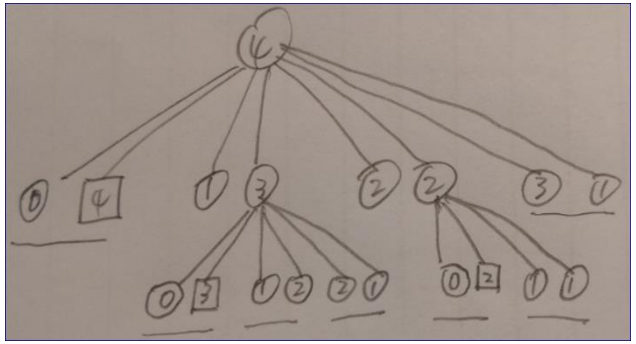
切割为 k 段，$1 \le k \le n$, 每段的长度为 $i_1, i_2, \ldots, i_k$，

最大收益为：

$$r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$$

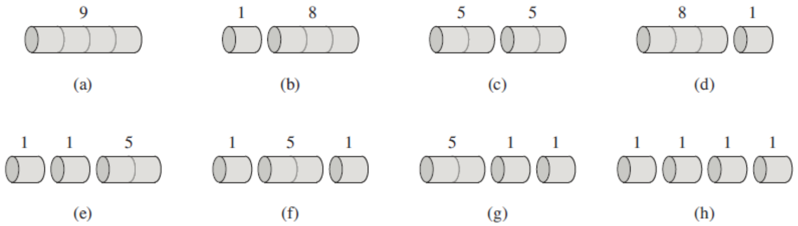
r_n : n 米长钢管的最优切割方式（最优：价格最高，收益最大）

$$r_n = \max(\underset{\downarrow\; r_0 + p_n}{p_n}, r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1) \qquad (15.1)$$



Step 2: A recursive solution（递归解）

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



$r_1 = 1$ from solution $1 = 1$ (no cuts) ,
$r_2 = 5$ from solution $2 = 2$ (no cuts) ,
$r_3 = 8$ from solution $3 = 3$ (no cuts) ,
$r_4 = 10$ from solution $4 = 2 + 2$,
$r_5 = 13$ from solution $5 = 2 + 3$,
$r_6 = 17$ from solution $6 = 6$ (no cuts) ,
$r_7 = 18$ from solution $7 = 1 + 6$ or $7 = 2 + 2 + 3$,
$r_8 = 22$ from solution $8 = 2 + 6$,
$r_9 = 25$ from solution $9 = 3 + 6$,
$r_{10} = 30$ from solution $10 = 10$ (no cuts) .

长度为 n 的钢管，一种最优的切割方式：

$$n = i_1 + i_2 + \cdots + i_k$$

切割为 k 段，$1 \le k \le n$, 每段的长度为 $i_1, i_2, \ldots, i_k$,

最大收益为：

$$r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$$

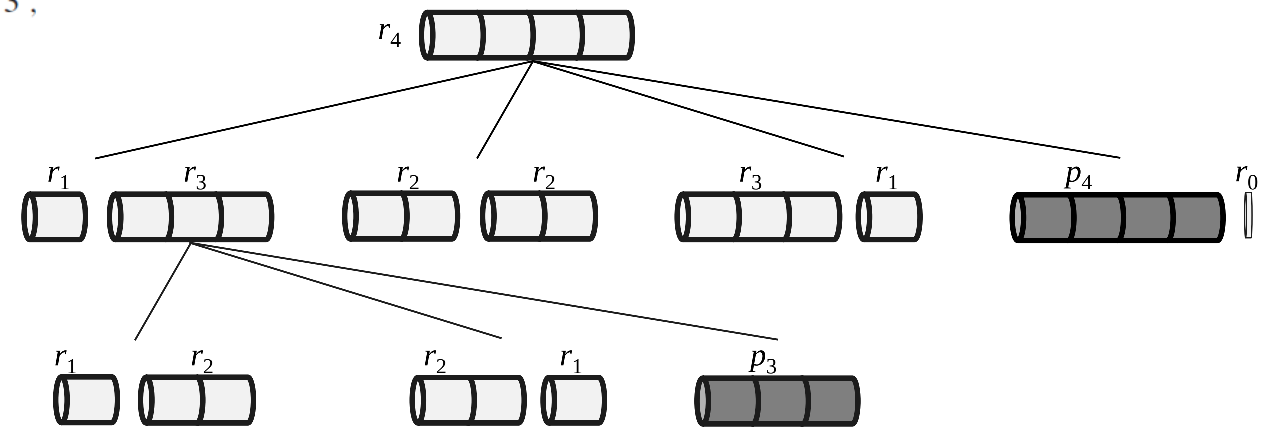
r_n : n 米长rod的最优切割方式（最优：价格最高，收益最大）

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1) \qquad (15.1)$$

$r_n = \max(r_1 + r_{n-1} , r_2 + r_{n-2} , \ldots , r_{n-1} + r_1 , p_n + r_0)$

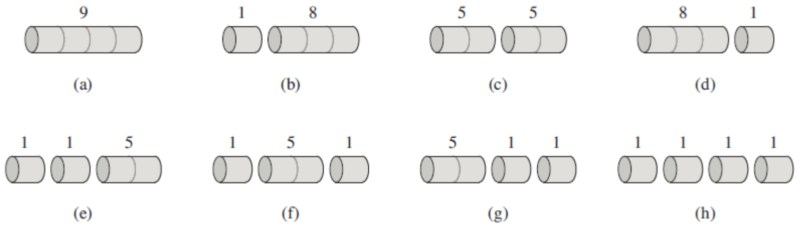
（按图示，写成这样更好理解？）

可以根据15.1直接写递归方程进行求解，但效率很低！



Step 2: A recursive solution（递归解）

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



$$
\begin{aligned}
r_1 &= 1 && \text{from solution } 1 = 1 \quad \text{(no cuts)}, \\
r_2 &= 5 && \text{from solution } 2 = 2 \quad \text{(no cuts)}, \\
r_3 &= 8 && \text{from solution } 3 = 3 \quad \text{(no cuts)}, \\
r_4 &= 10 && \text{from solution } 4 = 2 + 2, \\
r_5 &= 13 && \text{from solution } 5 = 2 + 3, \\
r_6 &= 17 && \text{from solution } 6 = 6 \quad \text{(no cuts)}, \\
r_7 &= 18 && \text{from solution } 7 = 1 + 6 \ \text{ or } \ 7 = 2 + 2 + 3, \\
r_8 &= 22 && \text{from solution } 8 = 2 + 6, \\
r_9 &= 25 && \text{from solution } 9 = 3 + 6, \\
r_{10} &= 30 && \text{from solution } 10 = 10 \quad \text{(no cuts)}.
\end{aligned}
$$

长度为 n 的钢管，一种最优的切割方式：

$$n = i_1 + i_2 + \cdots + i_k$$

切割为 k 段，$1 \le k \le n$, 每段的长度为 $i_1, i_2, \ldots, i_k$,

最大收益为：

$$r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$$

r_n : n 米长rod的最优切割方式（最优：价格最高，收益最大）

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1) \qquad (15.1)$$

$\Downarrow$?

$$\boxed{r_n = \max_{1 \le i \le n}(p_i + r_{n-i})} \qquad (15.2)$$

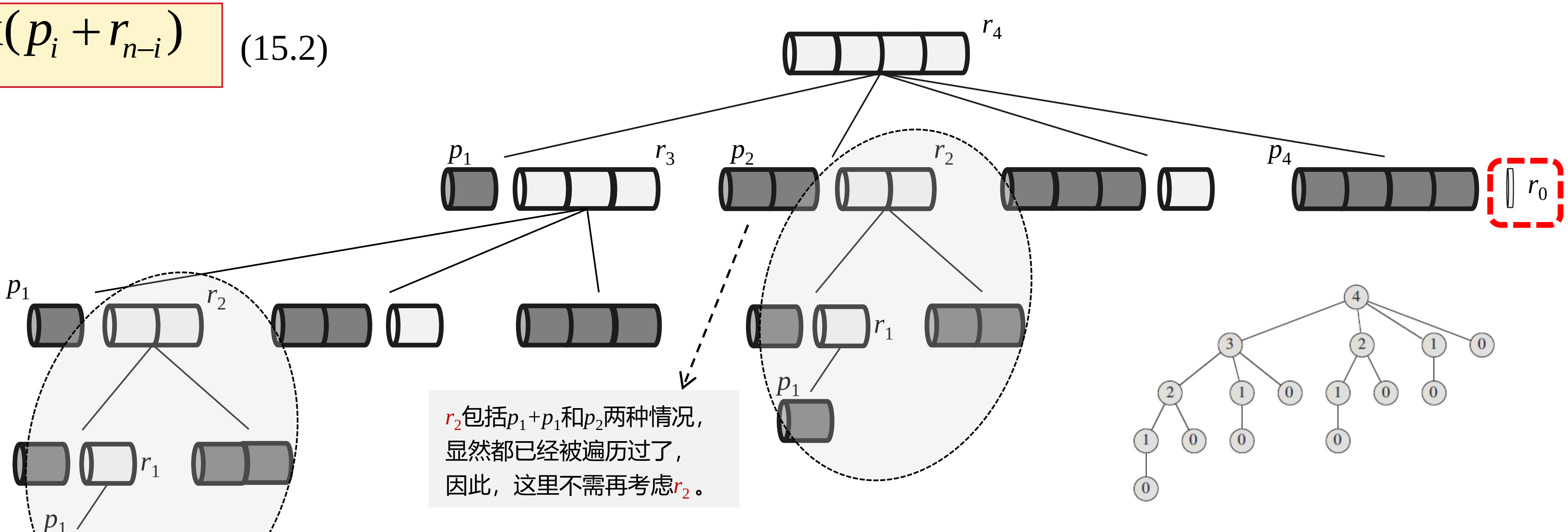
不用递归计算式(15.1)中 $r_i + r_{n-i}$ 部分的 r_i （因为 $r_i + r_{n-i}$ 与 $r_{n-i} + r_i$ 在式(15.1)中是对称的，r_i 从一个方向递归，就可以遍历所有切割情况。）

Step 2: A recursive solution（递归解）

1 $$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1) \quad (15.1)$$

2 $$r_n = \max_{1 \le i \le n}(p_i + r_{n-i}) \quad (15.2)$$

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



计算“式(15.2)”中的 $p_i + r_{n-i}$，就不用递归计算“式(15.1)”中 $r_i + r_{n-i}$ 部分的 r_i（对称情况，取其一）。

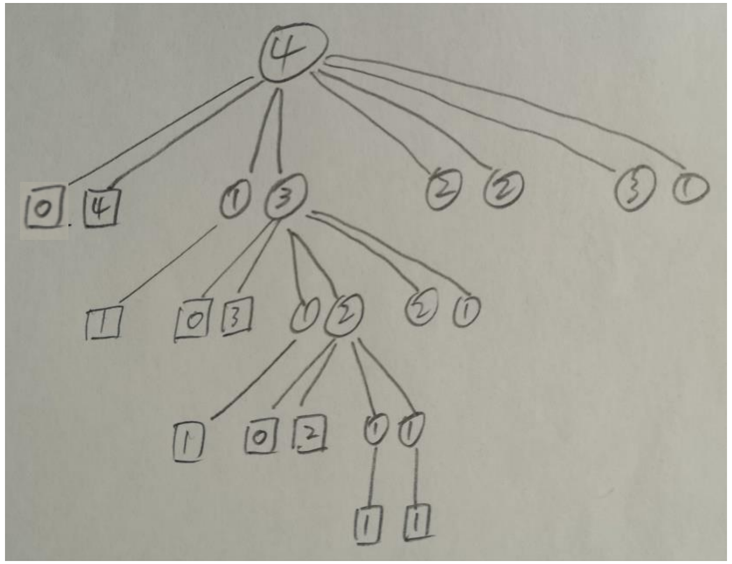
浅色钢管需要进行递归计算，深色钢管直接返回（不用切割）。即便是式15.2，采用递归，也有冗余计算，从图中还能看出，r_2 仍然被递归调用了两次。

Step 2: A recursive solution（递归解）

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

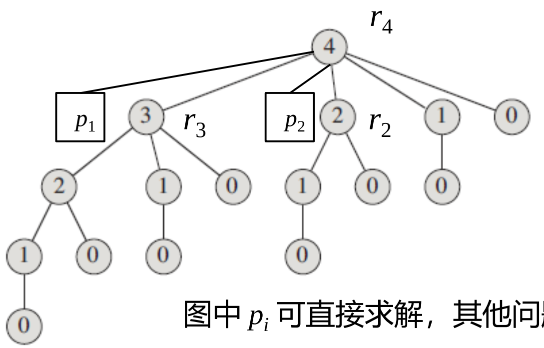
1 $$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1)$$

?



子问题多，递归过程中的冗余计算很多。

2 $$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

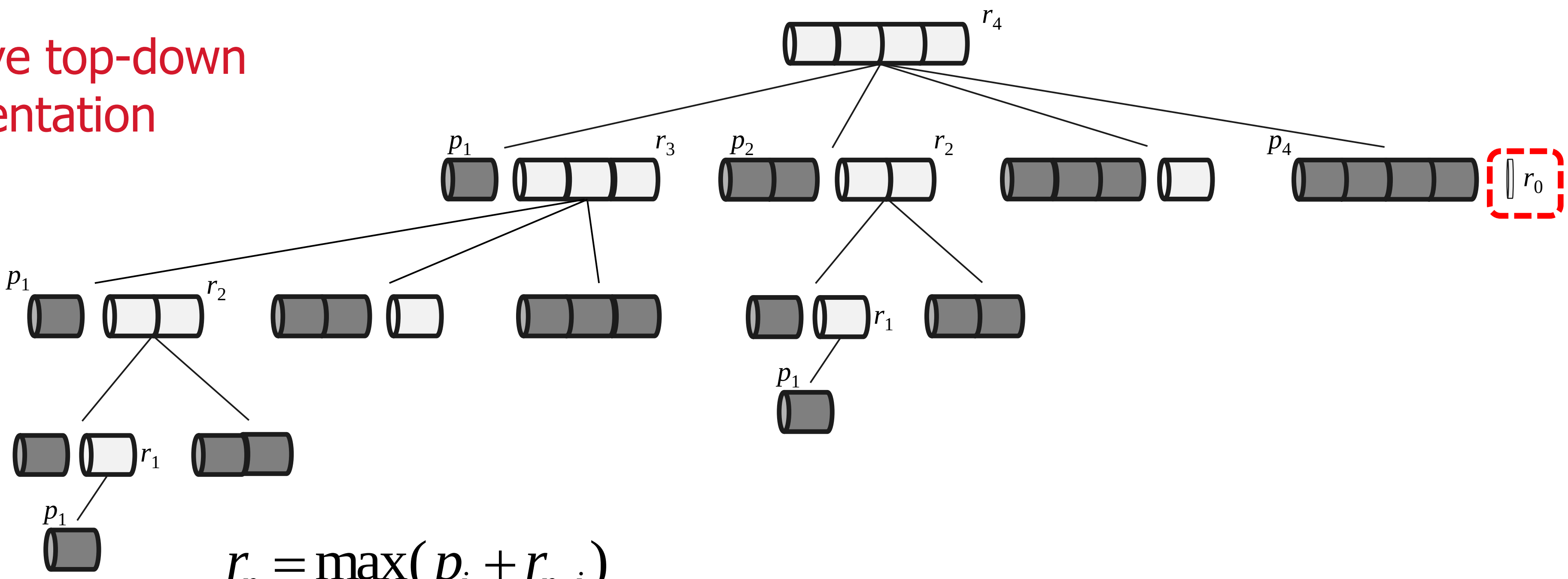


图中 p_i 可直接求解，其他问题需要递归计算

子问题减少，效率提升。
递归过程中的冗余计算仍然不少。

Step 2: A recursive solution（递归解）

Recursive top-down implementation



$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

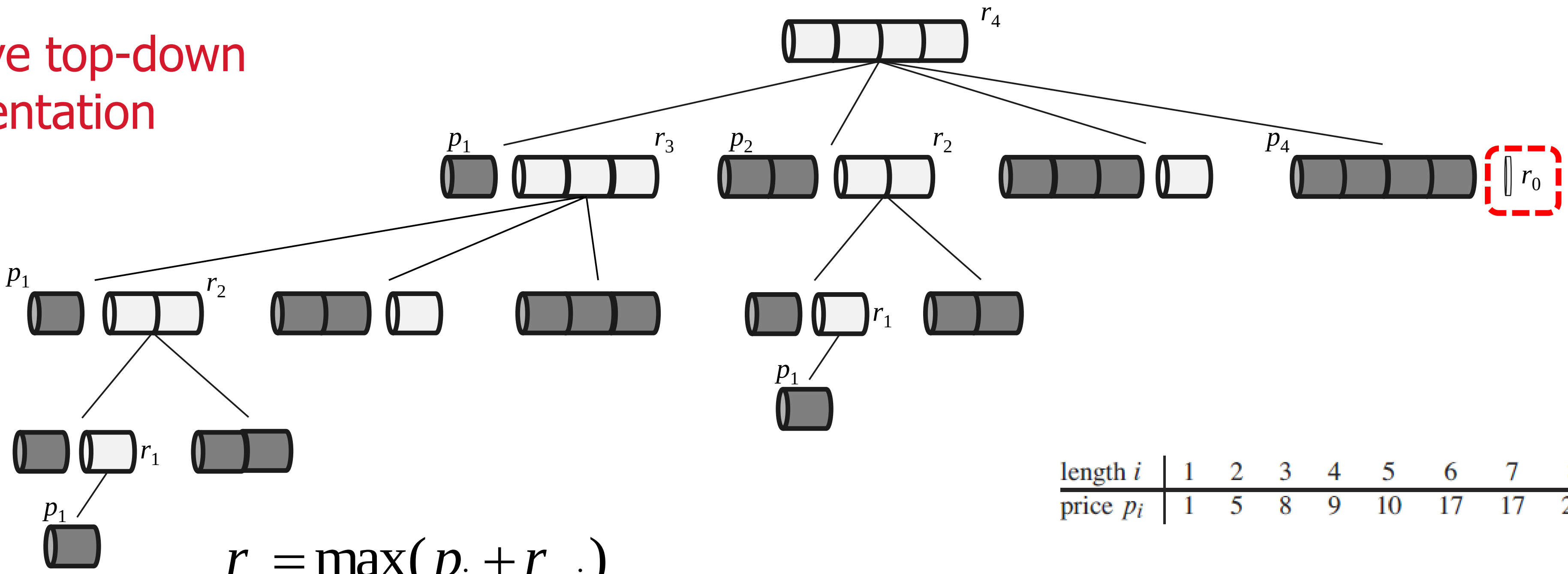
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

```
CUT-ROD(p, n)
if n == 0
    return 0
q = −∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n − i))
return q
```

Running time?

Step 2: A recursive solution（递归解）

Recursive top-down implementation



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

$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

```
CUT-ROD(p, n)
if n == 0
    return 0
q = -∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n - i))
return q
```

Running time?

$$T(n) = n + \sum_{j=0}^{n-1} T(j) \quad \Rightarrow \quad T(n) = 2^n$$

Step 2: A recursive solution（递归解）

Recursive top-down implementation

$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

```
CUT-ROD(p, n)
if n == 0
    return 0
q = −∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n − i))
return q
```

$$T(n) = n + \sum_{j=0}^{n-1} T(j) \quad \Rightarrow \quad T(n) = 2^n$$

置换法（归纳法）：

猜 $T(j) = O(2^j)$，即 $T(j) \le c2^j$

则 $T(n) = n + c(1+2+2^2+\ldots+2^{(n-1)})$

$= n + c(2^n - 1) = O(2^n)$?

$$T(n) = n + c(1+2+2^2+\ldots+2^{(n-1)})$$
$$= n + c(2^n - 1) = n + c2^n - c \le c2^n$$

$\Rightarrow$ $n \le c$ 显然不对！ 猜想有错?

Step 2: A recursive solution（递归解）

Recursive top-down implementation

$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

```
CUT-ROD(p, n)
if n == 0
    return 0
q = −∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n − i))
return q
```

$$T(n) = n + \sum_{j=0}^{n-1} T(j) \Rightarrow T(n) = 2^n$$

置换法（归纳法）：

猜 $T(j) = O(2^j)$，即$T(j) \le c2^j - k$，

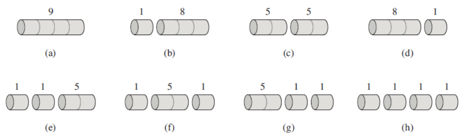
k 为常数，则

$$T(n) = n + c(1+2+2^2+\ldots+2^{(n-1)} - nk) = n + c(2^n - 1) - cnk \le c2^n - k$$

$\Rightarrow$ $n - c - cnk \le -k$ $\Rightarrow$ $c \ge k$ （显然，存在常数满足关系 $ck = 1$ 且 $c \ge k$）

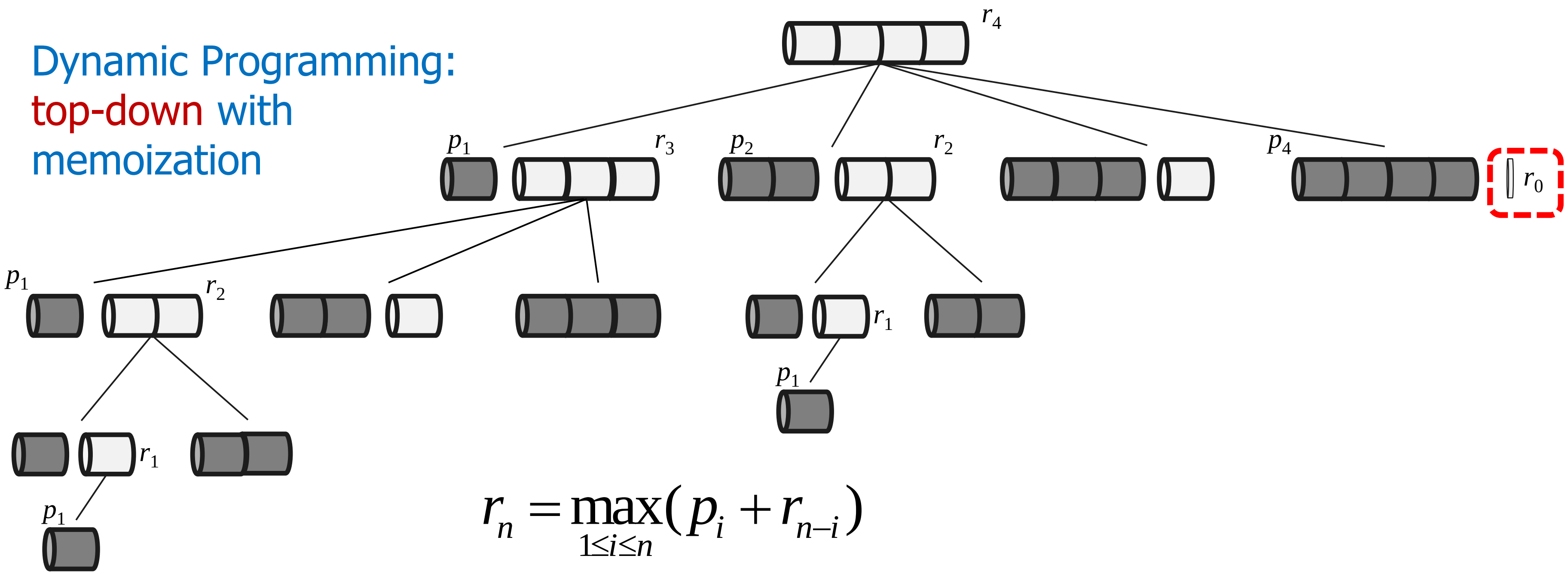
Step 3: Computing the optimal value（最优值）

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



$r_1 = 1$ from solution $1 = 1$ (no cuts) ,
$r_2 = 5$ from solution $2 = 2$ (no cuts) ,
$r_3 = 8$ from solution $3 = 3$ (no cuts) ,
$r_4 = 10$ from solution $4 = 2 + 2$,
$r_5 = 13$ from solution $5 = 2 + 3$,
$r_6 = 17$ from solution $6 = 6$ (no cuts) ,
$r_7 = 18$ from solution $7 = 1 + 6$ or $7 = 2 + 2 + 3$,
$r_8 = 22$ from solution $8 = 2 + 6$,
$r_9 = 25$ from solution $9 = 3 + 6$,
$r_{10} = 30$ from solution $10 = 10$ (no cuts) .

Dynamic Programming:
top-down with
memoization



$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

time-memory trade-off ?
时间记忆权衡。开辟空间 $r[]$，存储中间值，减少递归，节省时间。

```
MEMOIZED-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  for i = 0 to n
3      r[i] = −∞
4  return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

```
MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0              // 已经被计算出结果
2      return r[n]         // 直接返回，不重复计算
3  if n == 0
4      q = 0
5  else q = −∞
6      for i = 1 to n
7          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r))
8  r[n] = q
9  return q
```

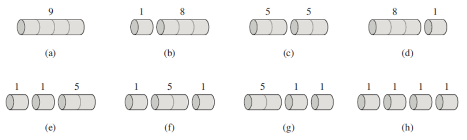
Running time?

$\Theta(n^2)$

Amortized Analysis: 每个顶点（子问题）被多次访问，但只递归计算一次。

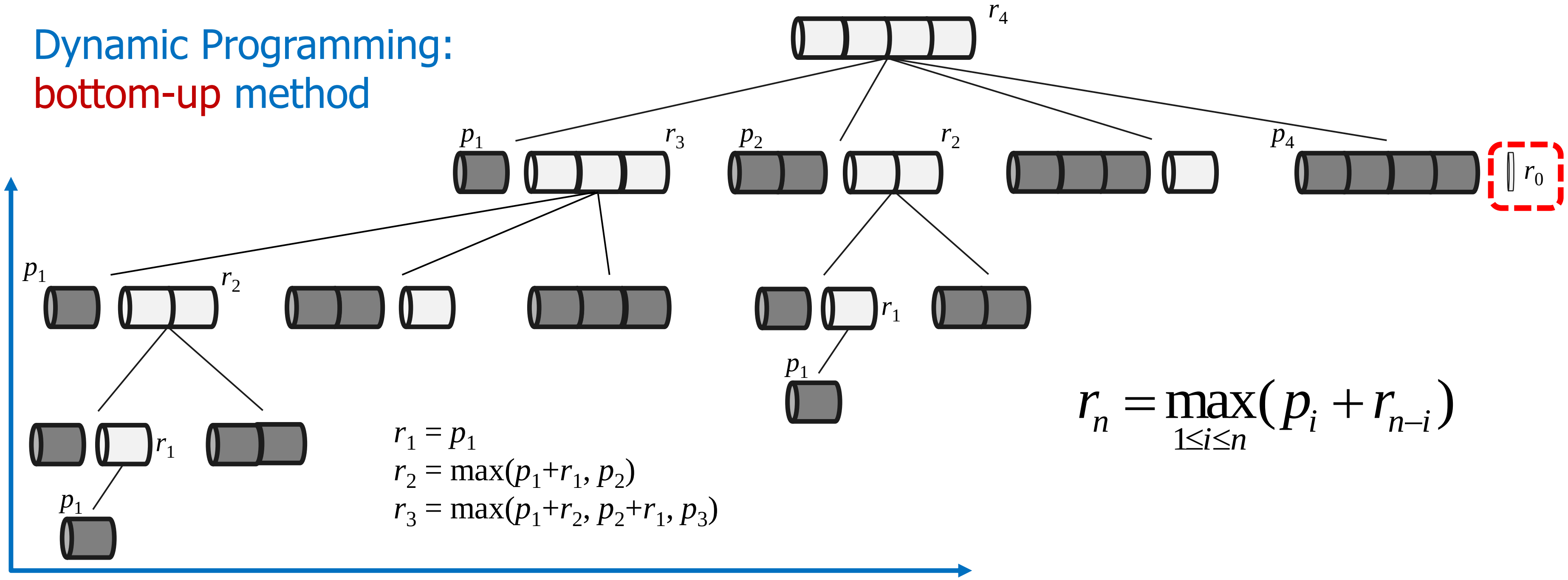
Step 3: Computing the optimal value（最优值）

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



$$
\begin{aligned}
r_1 &= 1 && \text{from solution } 1 = 1 \quad \text{(no cuts)},\\
r_2 &= 5 && \text{from solution } 2 = 2 \quad \text{(no cuts)},\\
r_3 &= 8 && \text{from solution } 3 = 3 \quad \text{(no cuts)},\\
r_4 &= 10 && \text{from solution } 4 = 2 + 2,\\
r_5 &= 13 && \text{from solution } 5 = 2 + 3,\\
r_6 &= 17 && \text{from solution } 6 = 6 \quad \text{(no cuts)},\\
r_7 &= 18 && \text{from solution } 7 = 1 + 6 \text{ or } 7 = 2 + 2 + 3,\\
r_8 &= 22 && \text{from solution } 8 = 2 + 6,\\
r_9 &= 25 && \text{from solution } 9 = 3 + 6,\\
r_{10} &= 30 && \text{from solution } 10 = 10 \quad \text{(no cuts)}.
\end{aligned}
$$

Dynamic Programming:
bottom-up method

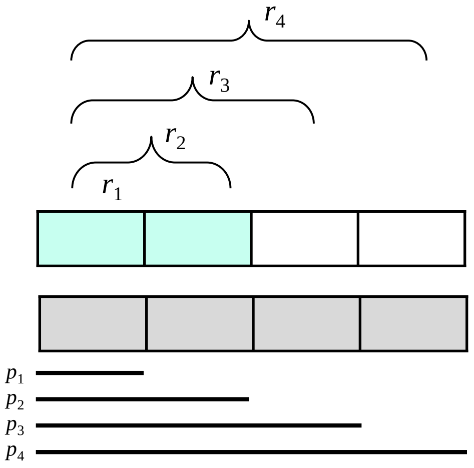


$r_1 = p_1$

$r_2 = \max(p_1+r_1, p_2)$

$r_3 = \max(p_1+r_2, p_2+r_1, p_3)$

$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$



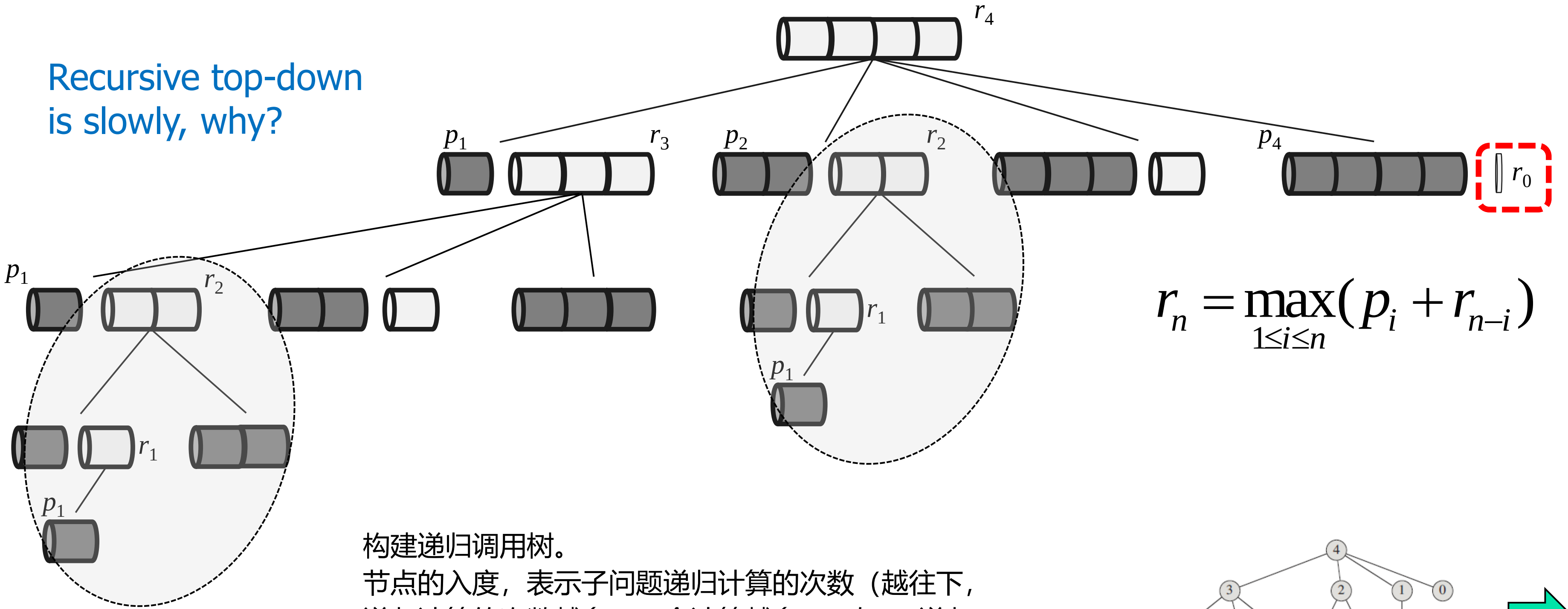

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        q = max(q, p[i] + r[j - i])
    r[j] = q   // programming: filling table
return r[n]
```

Running time?

$\Theta(n^2)$

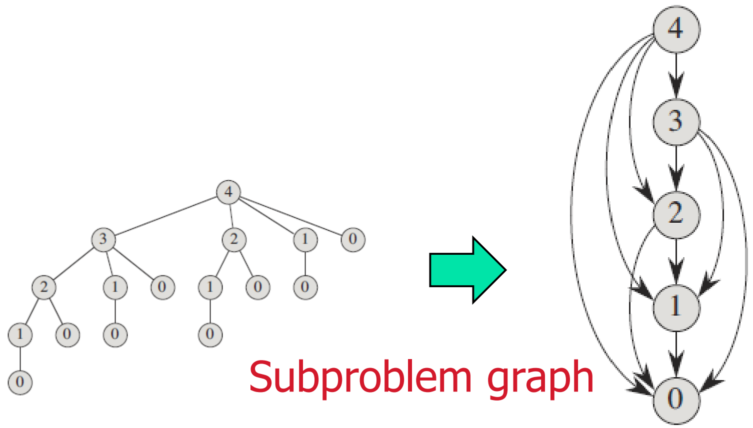
Step 3: Computing the optimal value（最优值）

Recursive top-down is slowly, why?



$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

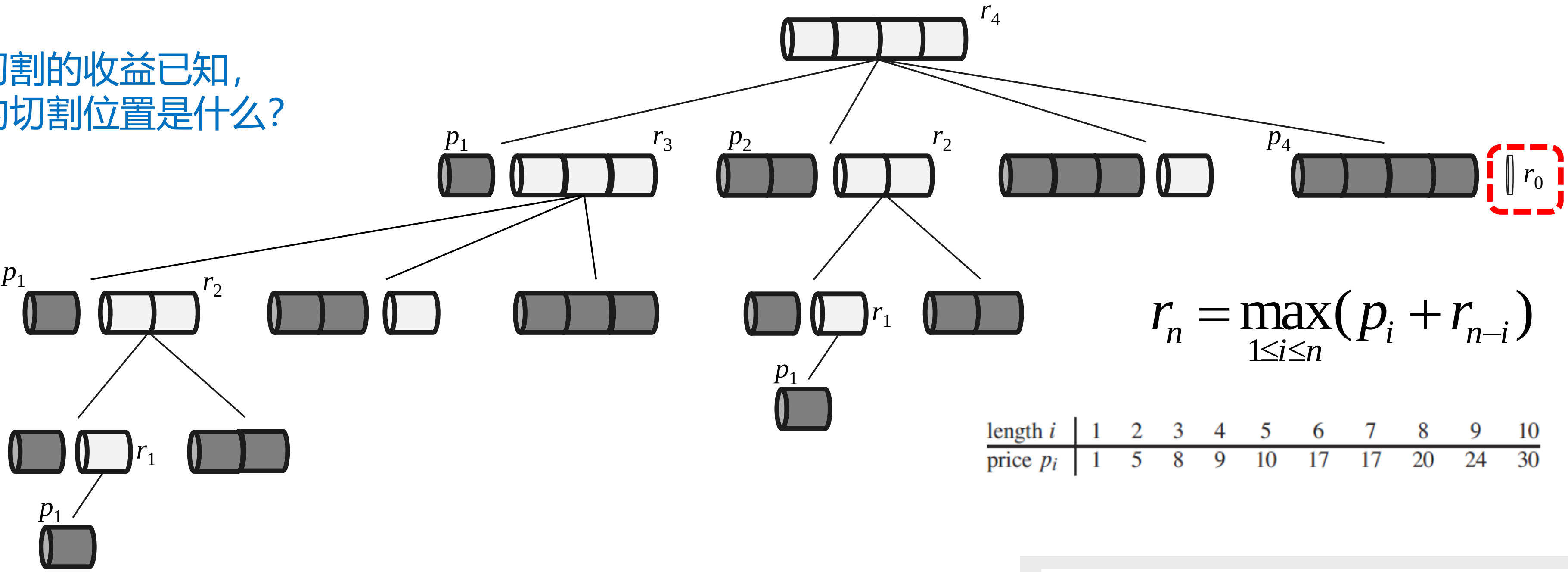
构建递归调用树。
节点的入度，表示子问题递归计算的次数（越往下，递归计算的次数越多，冗余计算越多）。如 r_3 递归计算一次，r_2 递归计算两次（调用过程：32，42），r_1 递归计算三次（调用过程：21，31，41）



1. 递归调用（自顶向下）：拓扑序，顶点之间的调用关系；每个顶点多次被访问（每次访问都重新进行递归计算）。
2. 带备忘录的递归调用（自顶向下）：每个顶点在递归调用中被访问多次，但只计算一次，有值后，其余访问直接返回值（不再往下递归计算，比如：42路线第一次遇到2，会递归求2；再调用32路线时遇到2，2已有存储的值，直接返回该值，不需要再递归求2，即，2往下的递归路线只走一次）。
3. 填表方法（自底向上）：拓扑逆序（把图中边的方向反向），每个顶点计算一次，每条边被访问一次（备忘录方法同理）。

Step 4: Reconstructing a solution（最优解）

最优切割的收益已知，
最优的切割位置是什么？



$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

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

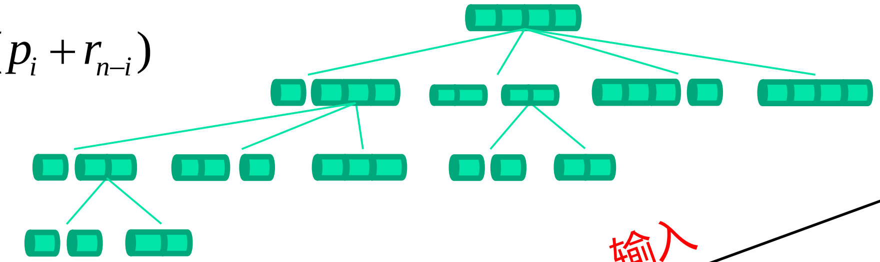
```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = −∞
    for i = 1 to j
        q = max(q, p[i] + r[j − i])
    r[j] = q
return r[n]
```

不仅仅求最大值 q，把取得最大值的位置 i 也记录下来（记录在 $s[j]$ 中）。

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] and s[0..n] be new arrays
r[0] = 0
for j = 1 to n
    q = −∞
    for i = 1 to j
        if q < p[i] + r[j − i]
            q = p[i] + r[j − i]
            s[j] = i
    r[j] = q
return r and s
```

Step 4：Reconstructing a solution（最优解）

$$r_n = \max_{1\le i\le n}(p_i + r_{n-i})$$



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

输入

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] and s[0..n] be new arrays
r[0] = 0
for j = 1 to n
    q = −∞
    for i = 1 to j
        if q < p[i] + r[j − i]
            q = p[i] + r[j − i]
            s[j] = i
    r[j] = q
return r and s
```

输出（填表）

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

输入

```
PRINT-CUT-ROD-SOLUTION(p, n)
(r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
while n > 0
    print s[n]
    n = n − s[n]
```

输出

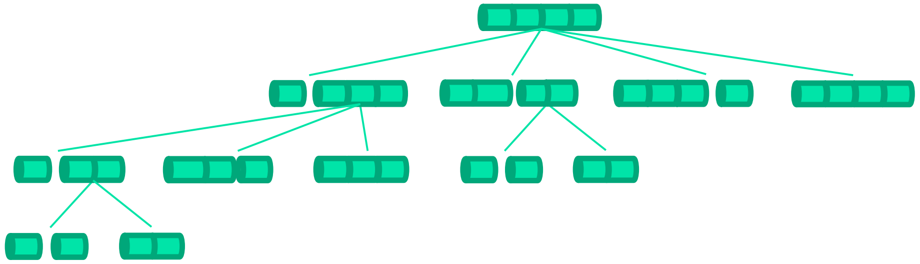
PRINT程序中实际上取参数 s 即可

$r_1 = 1$ from solution $1 = 1$ (no cuts) ,
$r_2 = 5$ from solution $2 = 2$ (no cuts) ,
$r_3 = 8$ from solution $3 = 3$ (no cuts) ,
$r_4 = 10$ from solution $4 = 2 + 2$,
$r_5 = 13$ from solution $5 = 2 + 3$,
$r_6 = 17$ from solution $6 = 6$ (no cuts) ,
$r_7 = 18$ from solution $7 = 1 + 6$ or $7 = 2 + 2 + 3$,
$r_8 = 22$ from solution $8 = 2 + 6$,
$r_9 = 25$ from solution $9 = 3 + 6$,
$r_{10} = 30$ from solution $10 = 10$ (no cuts) .

输出示例：
1. 长度为10米时，整卖最好，不需要切割，能卖30元；
2. 长度为9米时，两截，分别为3米和6米，共卖25元；
3. 长度为8米时，两截，分别为2米和6米，共卖22元；

……

Exercise 1



$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] and s[0..n] be new arrays
r[0] = 0
for j = 1 to n
    q = −∞
    for i = 1 to j
        if q < p[i] + r[j − i]
            q = p[i] + r[j − i]
            s[j] = i
    r[j] = q
return r and s
```

```
PRINT-CUT-ROD-SOLUTION(p, n)
(r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
while n > 0
    print s[n]
    n = n − s[n]
```

i	1	2	3	4	5	6	7
p_i	2	5	10	11	13	15	20

i	0	1	2	3	4	5	6	7
$r[i]$					?			
$s[i]$								

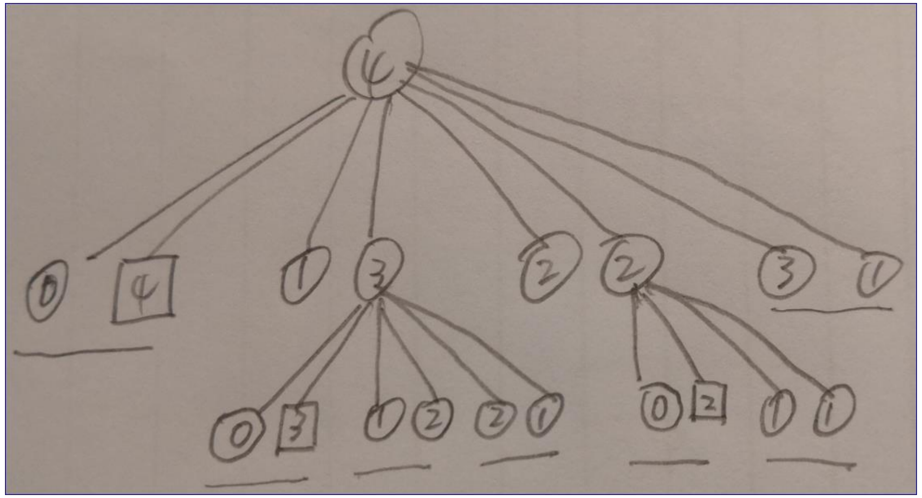
长度为7米时，如何切割收益最高？收益是多少？

Exercise 2

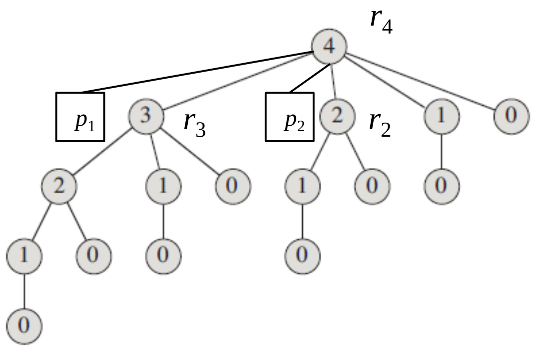
2. 针对(15.1)所述的自上而下的递归式，递归算法如何写？

其运行时间是多少？

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1) \qquad (15.1)$$



$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i}) \qquad (15.2)$$



?

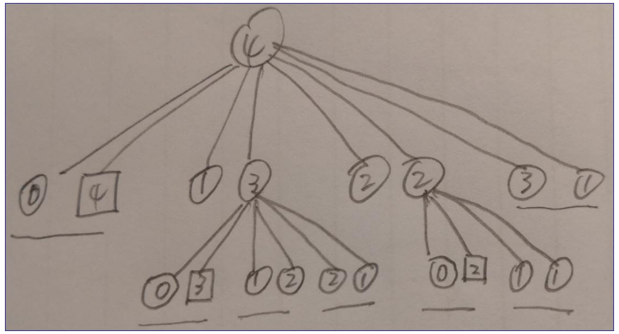
```
CUT-ROD(p, n)
    if n == 0
        return 0
    q = −∞
    for i = 1 to n
        q = max(q, p[i] + CUT-ROD(p, n − i))
    return q
```

$$T(n) = 2^n$$

Exercise 3

3. 用自底向上的DP算法，(15.1)和(15.2)哪个更快?

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1) \qquad (15.1)$$

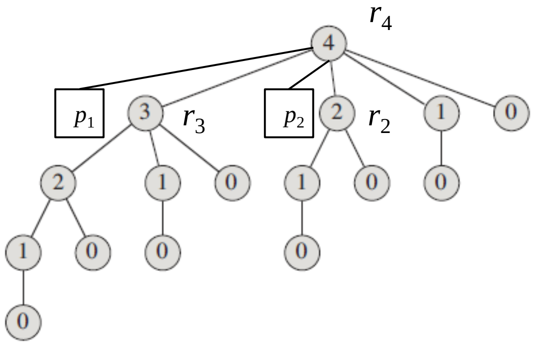


原始的递归式

?

$$r_n = \max_{1 \le i \le n}(p_i + r_{n-i}) \qquad (15.2)$$

改进的递归式




```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] and s[0..n] be new arrays
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        if q < p[i] + r[j - i]
            q = p[i] + r[j - i]
            s[j] = i
    r[j] = q
return r and s
```