

16 贪心算法

- 和动态规划相似，贪心算法被用于求解优化问题。
- 优化问题经常经过一系列步骤，每步都有一组选择。
- 对于许多优化问题，使用动态规划来求最优解有些多余。
- 贪心算法：更简单，更有效

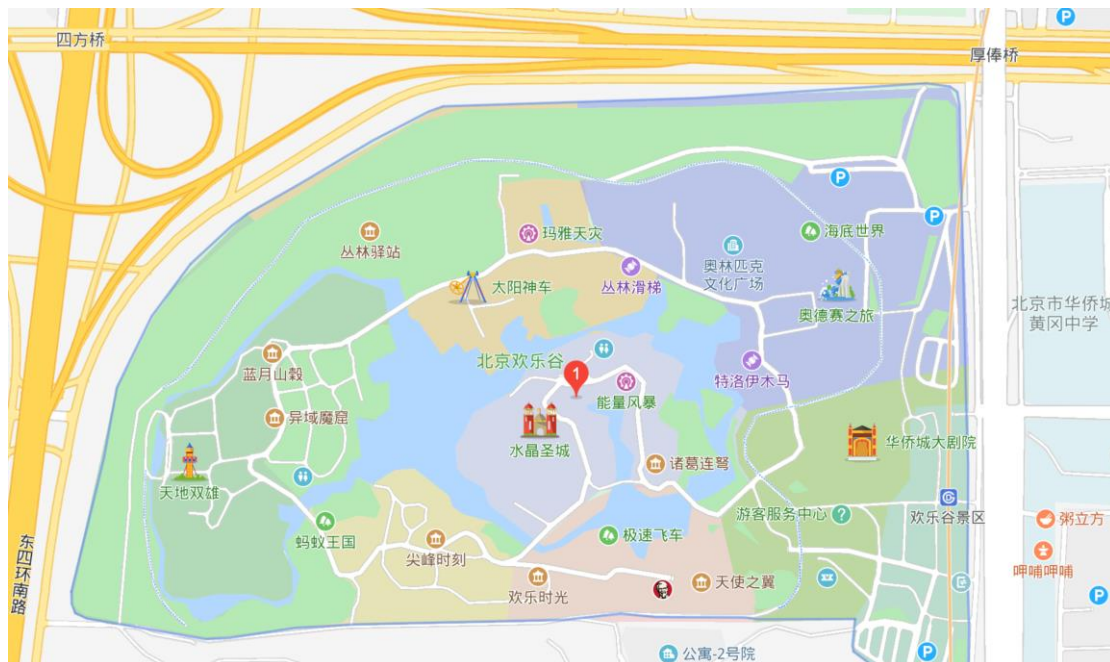
16 贪心算法

- 贪心算法 (GA) 不一定能求得最优解, 但对许多问题使用贪心算法足矣.
 - ◆ 16.1, 活动安排问题
 - ◆ 16.2, 贪婪算法的基本特征; 背包问题
 - ◆ 16.3, 一个重要应用: 数据压缩编码. (哈夫曼编码)
 - ◆ *16.4 拟阵和贪心方法
 - ◆ *16.5, 一个任务调度问题的拟阵(有限期作业调度)

16 贪心算法

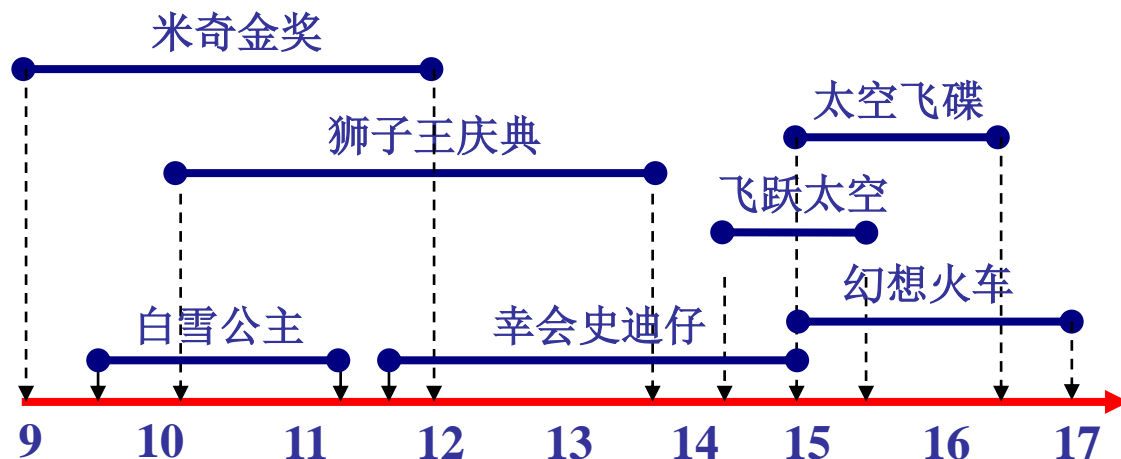
- 贪心算法非常强大，适用于许多问题：
 - ◆ 最小生成树 (Chap 23)
 - ◆ 最短路径 (Chap 24)
 - ◆ 集合覆盖 (Chap 35).
 - ◆ ...

Example: 北京欢乐谷游玩的活动安排

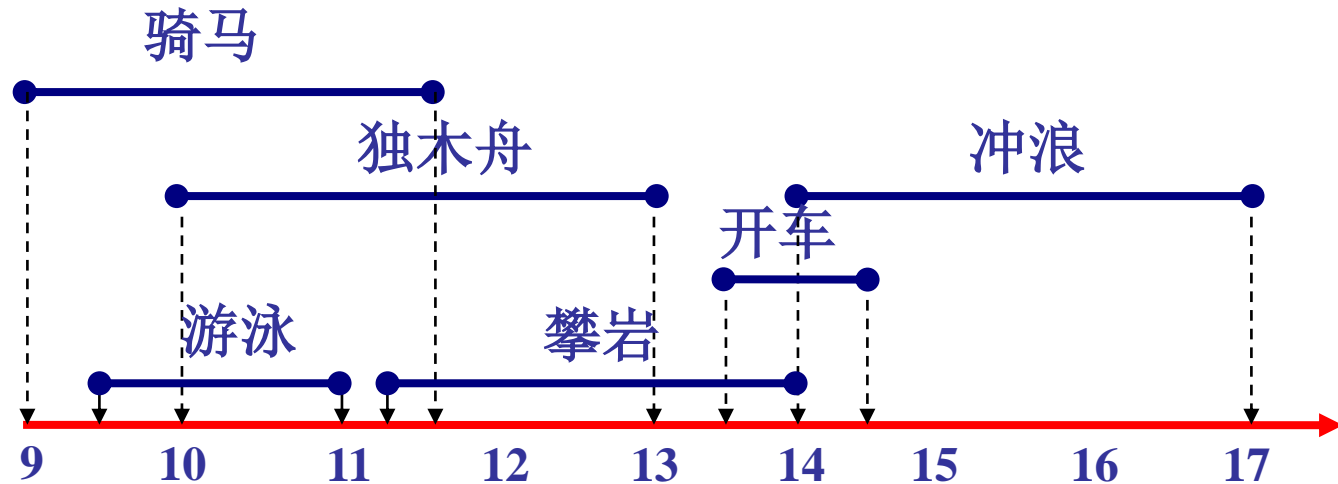


Activity Selection

- 欢乐谷
- Disneyland



Example: Activity Selection



- 怎样安排更多的活动？
 - ◆ S1. 最短活动优先原则
游泳, 开车
 - ◆ S2. 最早开始活动优先原则
骑马, 开车
 - ◆ S3. 最早结束活动优先原则
游泳, 攀岩, 冲浪

16.1 活动安排问题

应用场景:

借体育馆、借会议室

n 个活动

互斥使用公共资源

例如安排教室的使用

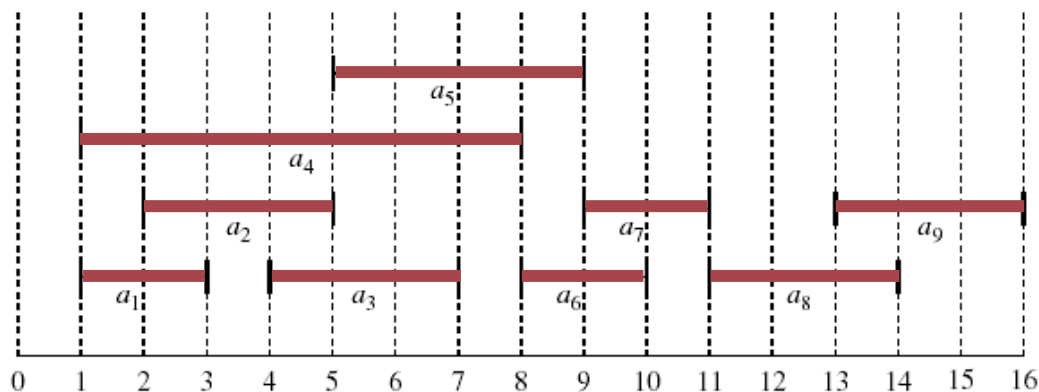
- ◆ 活动集 $S = \{a_1, a_2, \dots, a_n\}$.
- ◆ a_i 在 $[s_i, f_i)$ 期间需要资源, 这是一个半开区间, s_i 是开始时间, f_i 是结束时间。
- ◆ 目标: 安排一个活动计划, 使得相容的活动数目最多
- ◆ 其他目标: 最大化收入租金, ...



16.1 活动安排问题

- n 个活动需要互斥使用公共资源。
 - ◆ 活动集 $S = \{a_1, a_2, \dots, a_n\}$
 - ◆ a_i 在 $[s_i, f_i)$ 期间使用资源
- *Example:* S 按完成时间排序:

i	1	2	3	4	5	6	7	8	9
s_i	1	2	4	1	5	8	9	11	13
f_i	3	5	7	8	9	10	11	14	16



最大相互兼容集:

$\{a_1, a_3, a_6, a_8\}$.

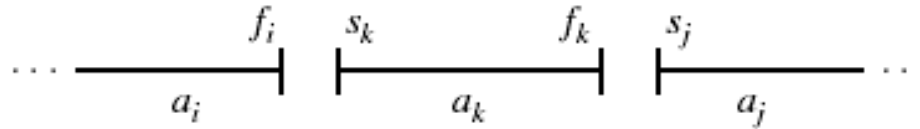
并不唯一, 还有

$\{a_2, a_5, a_7, a_9\}$.

16.1.1 活动选择的最优子结构

子问题空间

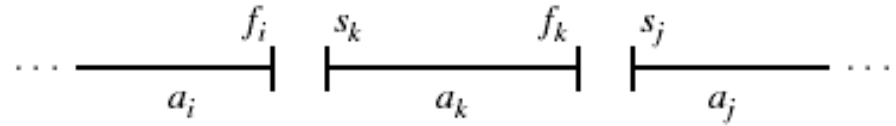
- $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$
= 在 a_i 结束后开始 和 在 a_j 开始前结束的活动



- S_{ij} 中的活动与以下兼容
 - ◆ 完成时间早于 f_i 的活动
 - ◆ 开始不早于 s_j 的活动.
- 为了表现完整的问题, 添加虚构的活动:
 - ◆ $a_0 = [-\infty, 0)$; $a_{n+1} = [\infty, \infty+1)$
 - ◆ 我们不关心 a_0 中的 $-\infty$ 或 a_{n+1} 中的“ $\infty+1$ ”
- 那么 $S = S_{0,n+1}$. S_{ij} 的范围是 $0 \leq i, j \leq n+1$.

16.1.1 活动选择的最优子结构

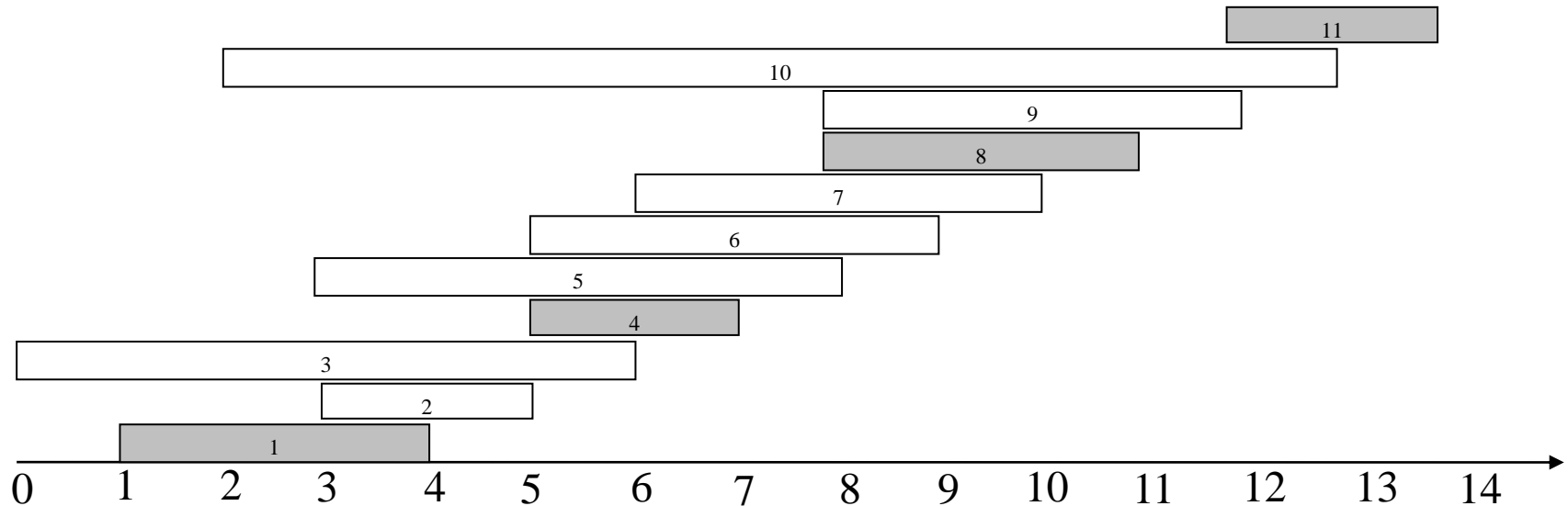
子问题空间



- $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$

- 以结束时间单调增的方式对活动进行排序

$$f_0 \leq f_1 \leq f_2 \leq \cdots \leq f_n < f_{n+1} \quad (\text{if } i \leq j, \text{ then } f_i \leq f_j) \quad (16.1)$$



16.1.1 活动选择的最优子结构

如果 $f_0 \leq f_1 \leq f_2 \leq \cdots \leq f_n < f_{n+1}$ (如果 $i \leq j$, 那么 $f_i \leq f_j$)
(16.1)

- 那么 $i \geq j \Rightarrow S_{ij} = \emptyset$

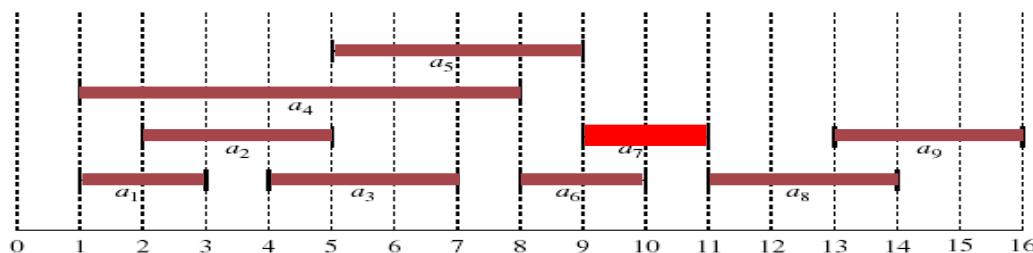
证明 如果存在 $a_k \in S_{ij}$, 那么

$$f_i \leq s_k < f_k \leq s_j < f_j \Rightarrow f_i < f_j.$$

但 $i \geq j \Rightarrow f_i \geq f_j$. 矛盾。

- 所以只需要考虑 S_{ij} $0 \leq i < j \leq n + 1$.
所有其他的 S_{ij} 都是 \emptyset .

16.1.1 活动选择的最优子结构



- 假设 S_{ij} 的一个解包括 a_k . 有两个子问题
 - ◆ S_{ik} (在 a_i 结束后开始, 在 a_k 开始前结束)
 - ◆ S_{kj} (在 a_k 结束后开始, 在 a_j 开始前结束)
- S_{ij} 的解 = (S_{ik} 的解) \cup $\{a_k\}$ \cup (S_{kj} 的解)
因为 a_k 不在子问题中, 且子问题互不相交,
 $|S_{ij} \text{ 的解}| = |S_{ik} \text{ 的解}| + 1 + |S_{kj} \text{ 的解}|$.
- **最优子结构:** 如果 S_{ij} 的一个最优解包括 a_k , 那么这个解中 S_{ik} 和 S_{kj} 的解也必须是最优的. (使用通常的复制粘贴参数).
- 假设: S_{ij} 非空; 已知 a_k ,
令 A_{ij} 为 S_{ij} 的最优解,
那么 $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$,

(16.2)

16.1.2 递归解法

- $c[i, j]$ 表示 S_{ij} 相容的最大活动数
 $i \geq j \Rightarrow S_{ij} = \emptyset \Rightarrow c[i, j] = 0.$
- 若 $S_{ij} \neq \emptyset$, 假设 a_k 包含在 S_{ij} 某个最大相容活动子集.
那么 $c[i, j] = c[i, k] + 1 + c[k, j].$
- 当然我们不知道用哪个 k , 所以

$$c[i, j] = \begin{cases} 0 & , \text{ if } S_{ij} = \emptyset, \\ \max_{i < k < j} \{c[i, k] + c[k, j] + 1\} & , \text{ if } S_{ij} \neq \emptyset. \end{cases} \quad (16.3)$$

为什么 k 是这个范围? 因为 $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\} \Rightarrow a_k$ 不能是 a_i 或 a_j (若 $k = i$, 我们得到 $c[i, j] = c[i, j] + 1$).

核心要素: 1. 递归; 2. 遍历n次, 选k

16.1.3 把 DP 解法转换为贪心解法

$$c[i, j] = \begin{cases} 0 & , \text{ if } S_{ij} = \emptyset, \\ \max_{i < k < j} \{c[i, k] + c[k, j] + 1\} & , \text{ if } S_{ij} \neq \emptyset. \end{cases} \quad (16.3)$$

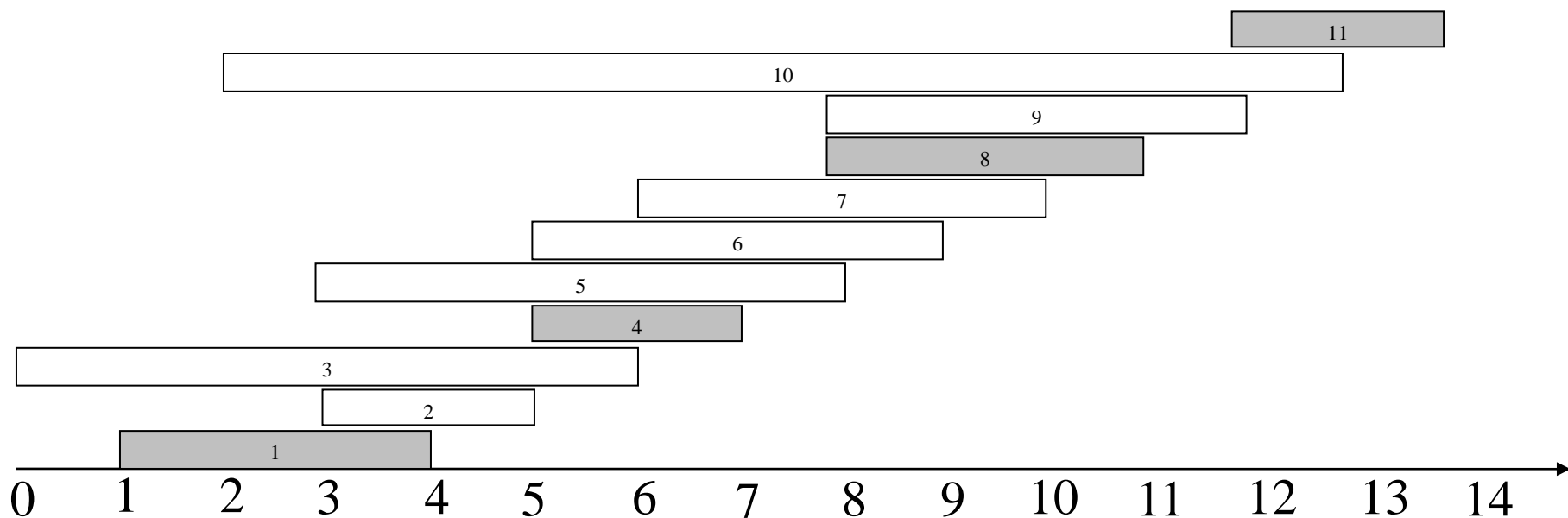
- 能很轻易的给问题16.3设计一个基于递归的算法。
 - (1) 直接递归算法? 复杂度?
 - (2) 动态规划算法? 复杂度?
- 对于 (16.3):
 - ◆ 有多少种选择?
 - ◆ 一个选择有多少子问题?
- 能简化解法吗?

16.1.3 把 DP 解法转换为贪心解法

□ Theorem 16.1

让 $S_{ij} \neq \emptyset$, 然后令 a_m 是 S_{ij} 最早结束的活动: $f_m = \min \{f_k : a_k \in S_{ij}\}$. 那么

1. a_m 包含在 S_{ij} 某个最大相容活动子集中.
2. $S_{im} = \emptyset$, 选择 a_m 后 S_{mj} 成为唯一一个非空子问题.



16.1.3 把 DP 解法转换为贪心解法

□ Theorem 16.1

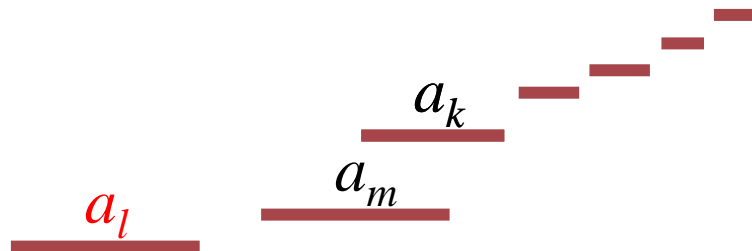
让 $S_{ij} \neq \emptyset$, 然后令 a_m 是 S_{ij} 最早结束的活动: $f_m = \min \{f_k : a_k \in S_{ij}\}$. 那么

1.

2. $S_{im} = \emptyset$, 选择 a_m 后 S_{mj} 成为唯一一个非空子问题.

证明

2. 假设有 $a_l \in S_{im}$. $f_i \leq s_l < f_l \leq s_m < f_m \Rightarrow f_l < f_m$. 那么 $a_l \in S_{ij}$ 同时它的结束时间早于 f_m , 这和我们选择 a_m 矛盾。因此不存在 $a_l \in S_{im} \Rightarrow S_{im} = \emptyset$.



16.1.3 把 DP 解法转换为贪心解法

□ **Theorem 16.1** 让 $S_{ij} \neq \emptyset$, 然后令 a_m 是 S_{ij} 最早结束的活动: $f_m = \min \{f_k : a_k \in S_{ij}\}$. 那么

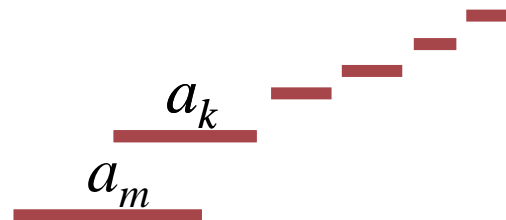
1. a_m 包含在 S_{ij} 某个最大相容活动子集中.

证明 1. 令 A_{ij} 是 S_{ij} 的最大相容活动子集. 以结束时间单调递增排序 A_{ij} 中的活动. 令 a_k 是 A_{ij} 第一个活动.

◆ 若 $a_k = a_m$, 结束 (a_m 在最大子集中).

◆ 否则, 令 $B_{ij} = A_{ij} - \{a_k\} \cup \{a_m\}$ (用 a_m 替换 a_k).

B_{ij} 中的活动互不相交. (A_{ij} 中的活动互不相交, a_k 是 A_{ij} 中第一个结束的活动. $f_m \leq f_k \Rightarrow a_m$ 不覆盖 B_{ij} 中的其他活动). 因此 $|B_{ij}| = |A_{ij}|$, A_{ij} 和 B_{ij} 都是最大子集.



16.1.3 把 DP 解法转换为贪心解法

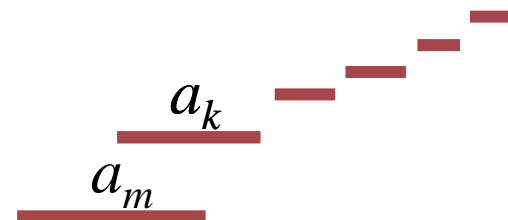
$$c[i, j] = \begin{cases} 0 & , \text{ if } S_{ij} = \emptyset, \\ \max_{i < k < j} \{c[i, k] + c[k, j] + 1\} & , \text{ if } S_{ij} \neq \emptyset. \end{cases} \quad (16.3)$$

- **Theorem 16.1** 让 $S_{ij} \neq \emptyset$, 然后令 a_m 是 S_{ij} 最早结束的活动:
 $f_m = \min \{f_k : a_k \in S_{ij}\}$. 那么
- 1、 a_m 包含在 S_{ij} 某个最大相容活动子集中.
 - 2、 $S_{im} = \emptyset$, 选择 a_m 后 S_{mj} 成为唯一一个非空子问题.
- 这个定理很有意义:

	使用定理前	使用定理后
最优解法子问题数量	2	1
要考虑的选择的数量	$O(j-i-1)$	1

16.1.3 把 DP 解法转换为贪心解法

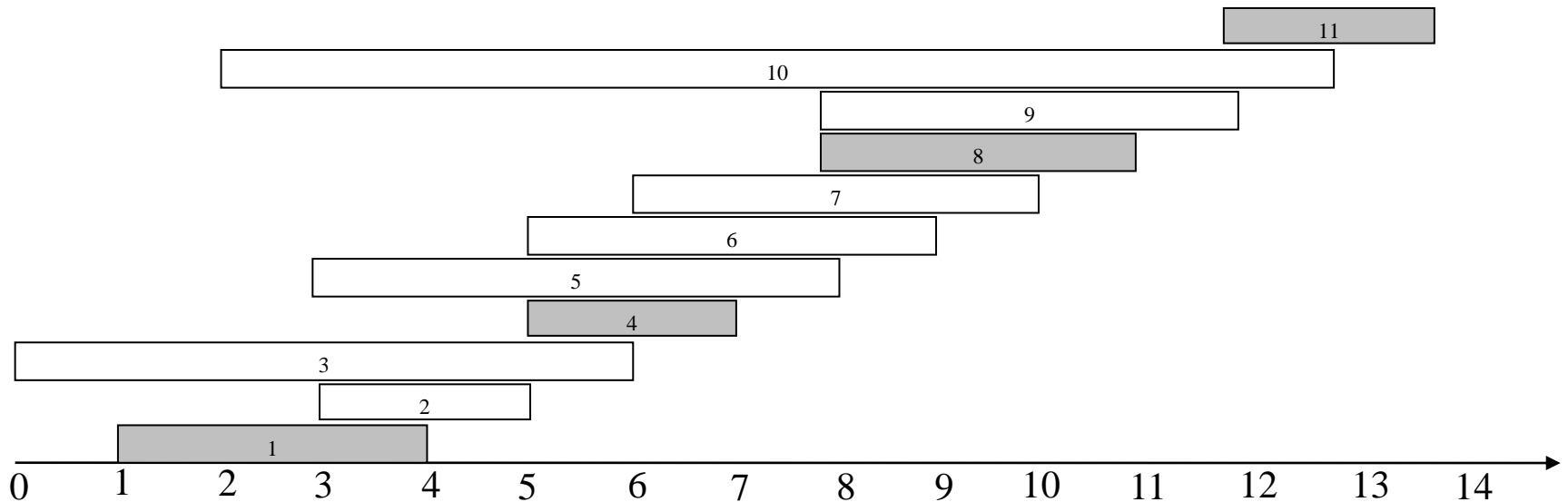
- Theorem 16.1 让 $S_{ij} \neq \emptyset$, 然后令 a_m 是 S_{ij} 最早结束的活动:
 $f_m = \min \{f_k : a_k \in S_{ij}\}$. 那么
 1. a_m 包含在 S_{ij} 某个最大相容活动子集中.
 2. $S_{im} = \emptyset$, 选择 a_m 后 S_{mj} 成为唯一一个非空子问题.
- 现在我们可以用 **自上而下** 的方式求解问题 S_{ij} (**DP 是什么样的方式?**)
 - ◆ 选择结束时间最早的 $a_m \in S_{ij}$: **贪心选择**. (留下尽可能多的时间来安排活动)
 - ◆ 求解 S_{mj} .



16.1.3 把 DP 解法转换为贪心解法

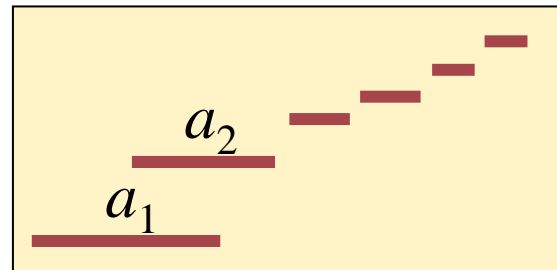
- 子问题是什么？

- ◆ 初始问题是 $S_{0, n+1}$ ($a_0 = [-\infty, 0)$; $a_{n+1} = [\infty, \infty+1)$)
- ◆ 假设我们第一个选择是 a_{m1} (实际上是 a_1)
- ◆ 那么下一个子问题是 $S_{m1, n+1}$
- ◆ 假设下一个选择是 a_{m2} (一定是 a_2 吗?)
- ◆ 下一个子问题 $S_{m2, n+1}$
- ◆ 这样继续下去



16.1.3 把 DP 解法转换为贪心解法

- 子问题是什么？
 - ◆ 初始问题是 $S_{0, n+1}$
 - ◆ 假设我们第一个选择是 a_{m1}
 - ◆ 那么下一个子问题是 $S_{m1, n+1}$
 - ◆ 假设下一个选择是 a_{m2}
 - ◆ 下一个子问题 $S_{m2, n+1}$
 - ◆ 这样继续下去
- 每一个子问题是 $S_{mi, n+1}$ 。
- 所选的子问题，其完成时间是增序排列
- 因此我们可以按完成时间单调递增的顺序只考虑每个活动一次。

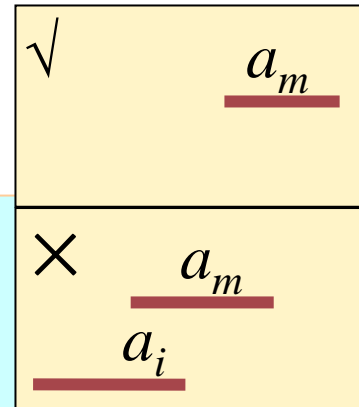


16.1.4 一种递归贪心算法

- 原问题是 $S_{0, n+1}$
- 子问题是 $S_{mi, n+1}$
- 假设活动已经按结束时间单调递增排序。(如果没有, 则在 $O(n \lg n)$ 时间内排序。) 返回 $S_{i, n+1}$ 的一个最优解。

REC-ACTIVITY-SELECTOR(s, f, i, n)

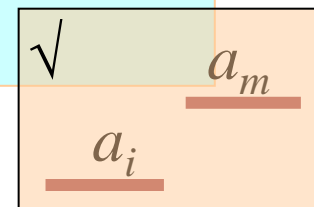
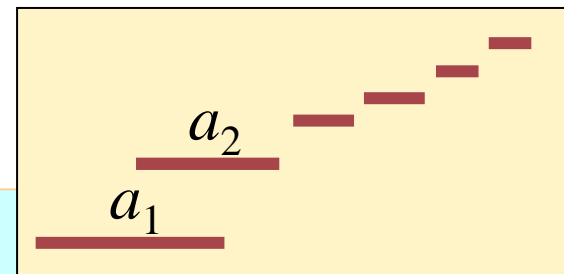
```
1  $m \leftarrow i+1$  // initially  $i = 0, m = 1$ 
2 while  $m \leq n$  and  $s_m < f_i$  // Find next activity in  $S_{i, n+1}$ .
3      $m \leftarrow m+1$ 
4 if  $m \leq n$ 
5     return  $\{a_m\} \cup \text{REC-ACTIVITY-SELECTOR}(s, f, m, n)$ 
6 else return  $\emptyset$ 
```



16.1.4 一种递归贪心算法

REC-ACTIVITY-SELECTOR(s, f, i, n)

```
1  $m \leftarrow i+1$ 
2 while  $m \leq n$  and  $s_m < f_i$  // Find next activity in  $S_{i, n+1}$ .
3    $m \leftarrow m+1$ 
4 if  $m \leq n$ 
5   return  $\{a_m\} \cup \text{REC-ACTIVITY-SELECTOR}(s, f, m, n)$ 
6 else return  $\emptyset$ 
```



- **Initial call:** REC-ACTIVITY-SELECTOR($s, f, 0, n$).
- **Idea:** **while** 循环检查 $a_{i+1}, a_{i+2}, \dots, a_n$ 直到找到一个和 a_i 相容的活动 a_m (need $s_m \geq f_i$).
 - ◆ 如果循环因找到 a_m 而终止 ($m \leq n$), 递归求解 $S_{m, n+1}$, and 返回它的解和 a_m .
 - ◆ 如果循环找不到一个相容的 a_m ($m > n$), 那就仅返回空集。

16.1.4 一种递归贪心算法

REC-ACTIVITY-SELECTOR(s, f, i, n)

1 $m \leftarrow i+1$

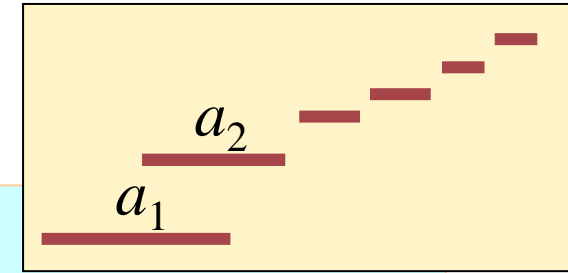
2 **while** $m \leq n$ and $s_m < f_i$ // Find next activity in $S_{i,n+1}$.

3 $m \leftarrow m+1$

4 **if** $m \leq n$

5 **return** $\{a_m\} \cup \text{REC-ACTIVITY-SELECTOR}(s, f, m, n)$

6 **else return** \emptyset



- **Time:** $\Theta(n)$ —每个活动只检查一次。

$$T(n) = m_1 + T(n - m_1) = m_1 + m_2 + T(n - m_1 - m_2)$$

$$= m_1 + m_2 + m_3 + T(n - m_1 - m_2 - m_3) = \dots$$

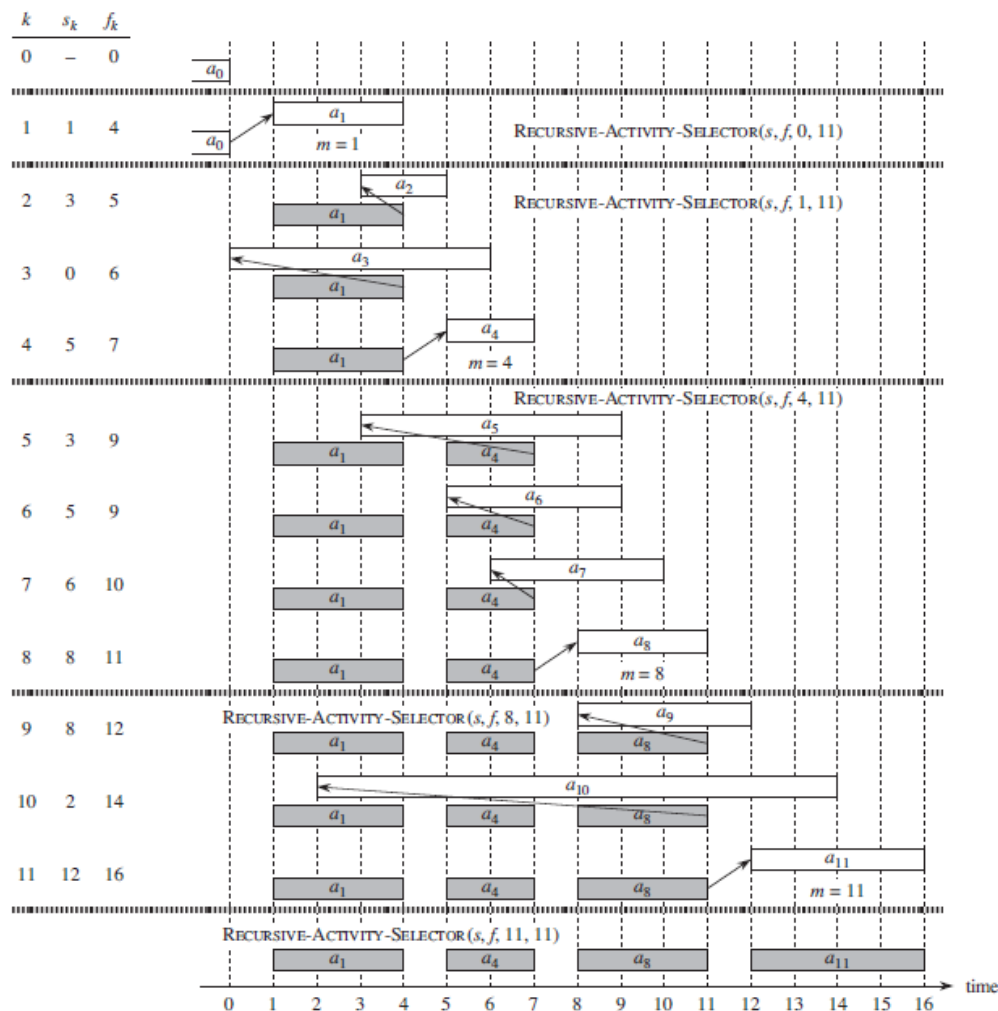
$$= \sum m_k + T(n - \sum m_k)$$

$$\text{basecase: } n - \sum m_k = 1, \text{ then } \sum m_k = n - 1, \sum m_k + T(1) = \Theta(n)$$

16.1.4 一种递归贪心算法

- **Initial call:** REC-ACTIVITY-SELECTOR($s, f, 0, n$).

- **Idea:** while 循环检查 a_{i+1} , a_{i+2}, \dots, a_n 直到找到一个和 a_i 相容的活动 a_m (need $s_m \geq f_i$).
 - ◆ 如果循环因找到 a_m 而终止 ($m \leq n$), 递归求解 $S_{m, n+1}$, 返回它的解和 a_m .
 - ◆ 如果循环找不到一个相容的 a_m ($m > n$), 那就仅返回空集。

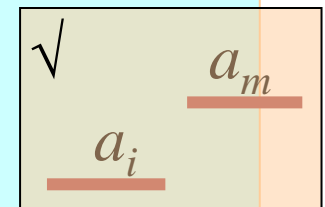
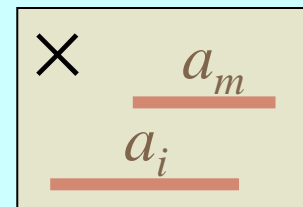
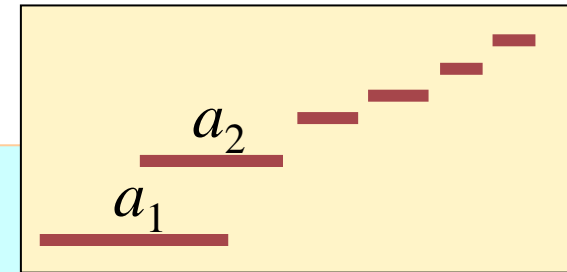


16.1.5 迭代贪心算法

- **REC-ACTIVITY-SELECTOR** 几乎是“尾递归”的。
- 我们可以很容易的将递归过程转换为迭代过程。（一些编译器自动完成这项工作）

GREEDY-ACTIVITY-SELECTOR(s, f, n)

```
1  $A \leftarrow \{a_1\}$ 
2  $i \leftarrow 1$ 
3 for  $m \leftarrow 2$  to  $n$ 
4   if  $s_m \geq f_i$ 
5      $A \leftarrow A \cup \{a_m\}$ 
6      $i \leftarrow m$  //  $a_i$  is most recent addition to  $A$ 
7 return  $A$ 
```



Review

- 贪心算法的关键: 当我们要做选择时, 选择当前情况下的最优解。 希望局部最优解能产生全局最优解。
- 贪心算法: 更简单, 更有效

16 贪心算法

- 16.1, 活动安排问题

- 16.2

贪心算法的基本特征：
背包问题

- 16.3, 一个重要应用：数据压缩编码设计（哈夫曼编码）

16.2 贪心策略的要素

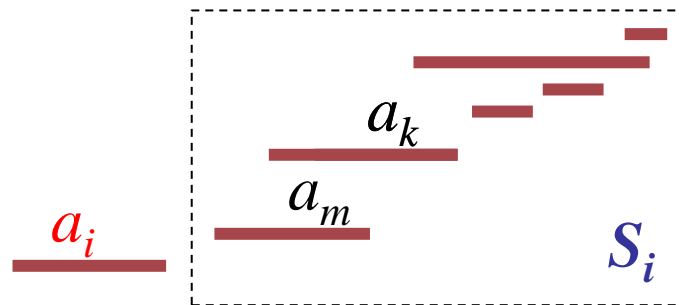
- 每次决策时，当前所做的选择看起来是“最好”的
- 我们在活动选择上做了什么？
 1. 确定最优子结构。
 2. 开发一个递归解法。
 3. 证明在递归的任何阶段，贪心选择是最优选择之一。
 4. 说明通过贪心选择，只有一个子问题非空。
 5. 开发一个递归贪心算法。
 6. 把它转换成迭代算法。

16.2 贪心策略的要素

- 这些步骤看起来像动态规划。

- 开发子结构时要注意

- ◆ 做贪心选择,
- ◆ 只留一个子问题。



- 对于活动选择，我们证明了贪心选择意味着在 S_{ij} 中，只有 i 变化， j 固定在 $n+1$

- 所以，我们可以从贪心算法开始：

- ◆ 定义 $S_i = \{a_k \in S : f_i \leq s_k\}$, (所有在 a_i 结束之后开始的活动)

- ◆ 给出贪心选择, S_i 中第一个结束的 a_m
结合 S_m 的最优解

} $\Rightarrow S_i$ 的最优解

16.2 贪心策略的要素

典型的流线型步骤

1. Cast the optimization problem as one in which we make a choice and are left with one subproblem to solve.

快速做选择，且留下尽可能少的子问题，且子问题包括的信息尽可能多

2. Prove that there's always an optimal solution that makes the greedy choice, so that the greedy choice is always safe.

选择是解的一部分【贪婪】，因此贪婪选择是安全的

3. Show that greedy choice and optimal solution to subproblem \Rightarrow optimal solution to the problem.

贪婪选择 + 子问题的最优解 \Rightarrow 原问题的最优解

16.2 贪心策略的要素

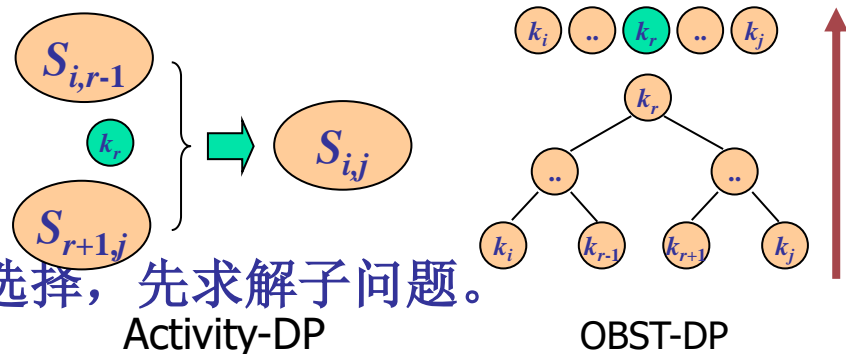
- 没有一般化的规则来说明贪婪算法是否最优，但有两个基本要点
 1. 贪心选择属性
 2. 最优子结构

16.2.1 贪心选择属性

- 全局最优解可以通过局部最优(贪心)的选择得到。

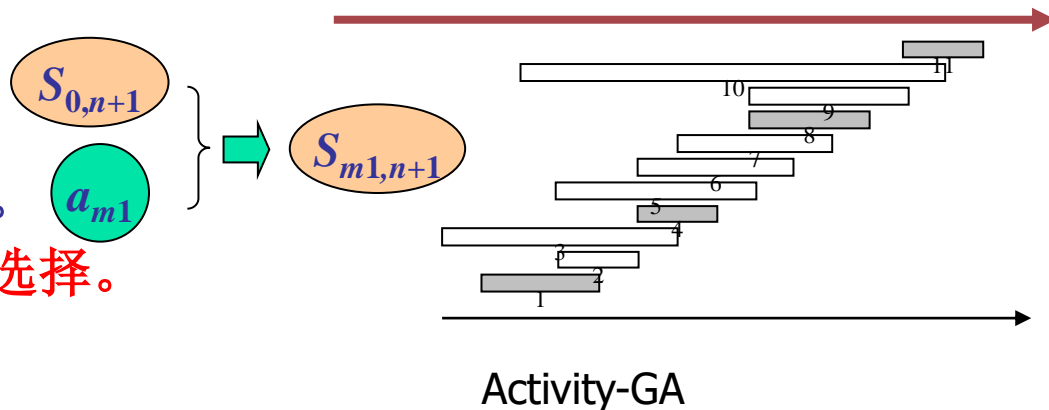
- 动态规划

- 每一步都要作出选择。
- 根据已知子问题的最优解作出选择，先求解子问题。
- 自底向上求解。



- 贪心

- 每一步都要作出选择。
- 在求解子问题前作出选择。
- 自顶向下求解。



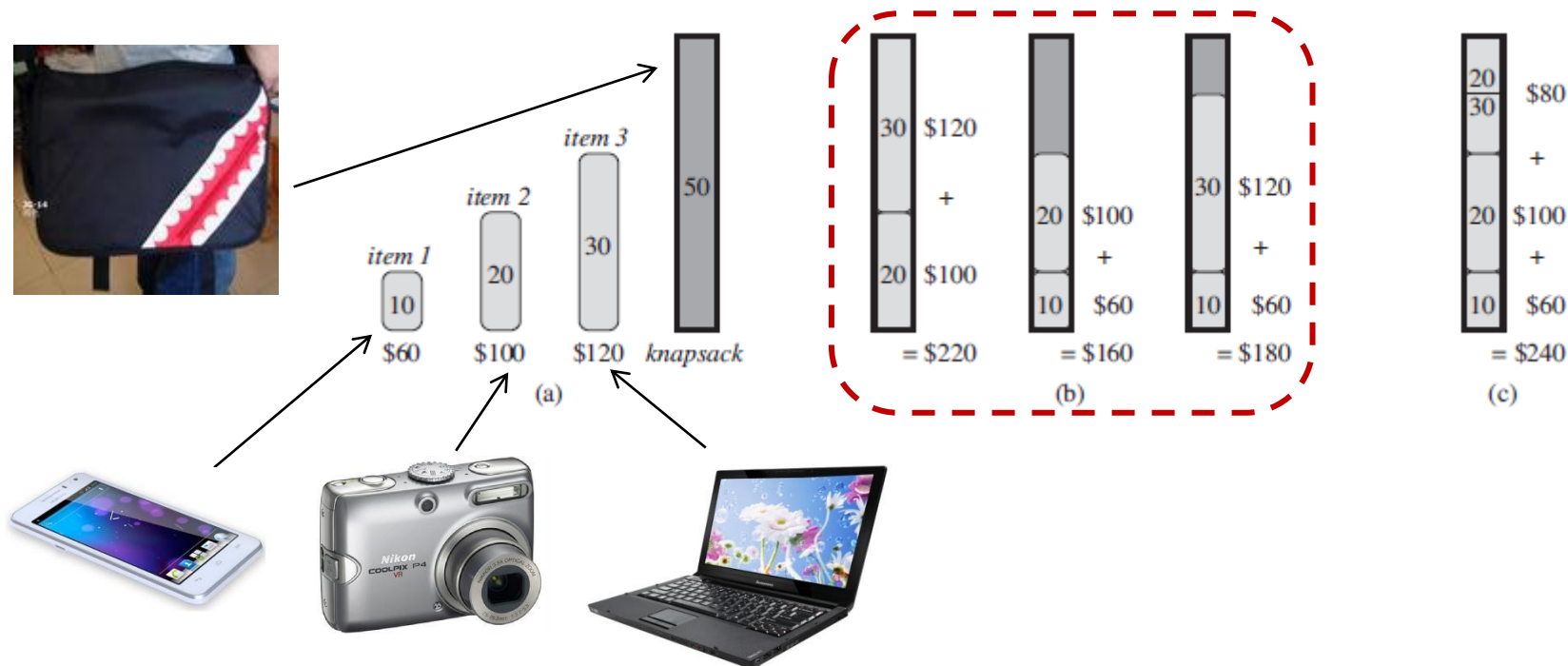
16.2.1 贪心选择属性

- 我们必须证明每一步的贪婪选择都能产生全局最优解。很难!也许需要智慧!
- 定理 16.1,证明解 (A_{ij}) 可以改为使用贪心选择 (a_m) , 留下一个类似但更小的子问题 (A_{mj}) 。
- 我们可以从贪婪选择属性中获得效率收益。(例如在活动选择中, 把活动按结束时间单增排序, 只需要检查每个活动一次)
 - ◆ 对输入进行预处理来将其放进弹性序列
 - ◆ 从技术上讲, 最大值或最小值通常是最佳选择。

16.2.2 最优子结构

- **最优子结构**:问题的最优解包含子问题的最优解。
- 只需要说明**子问题的最优解和贪心选择** \Rightarrow 原问题的最优解。

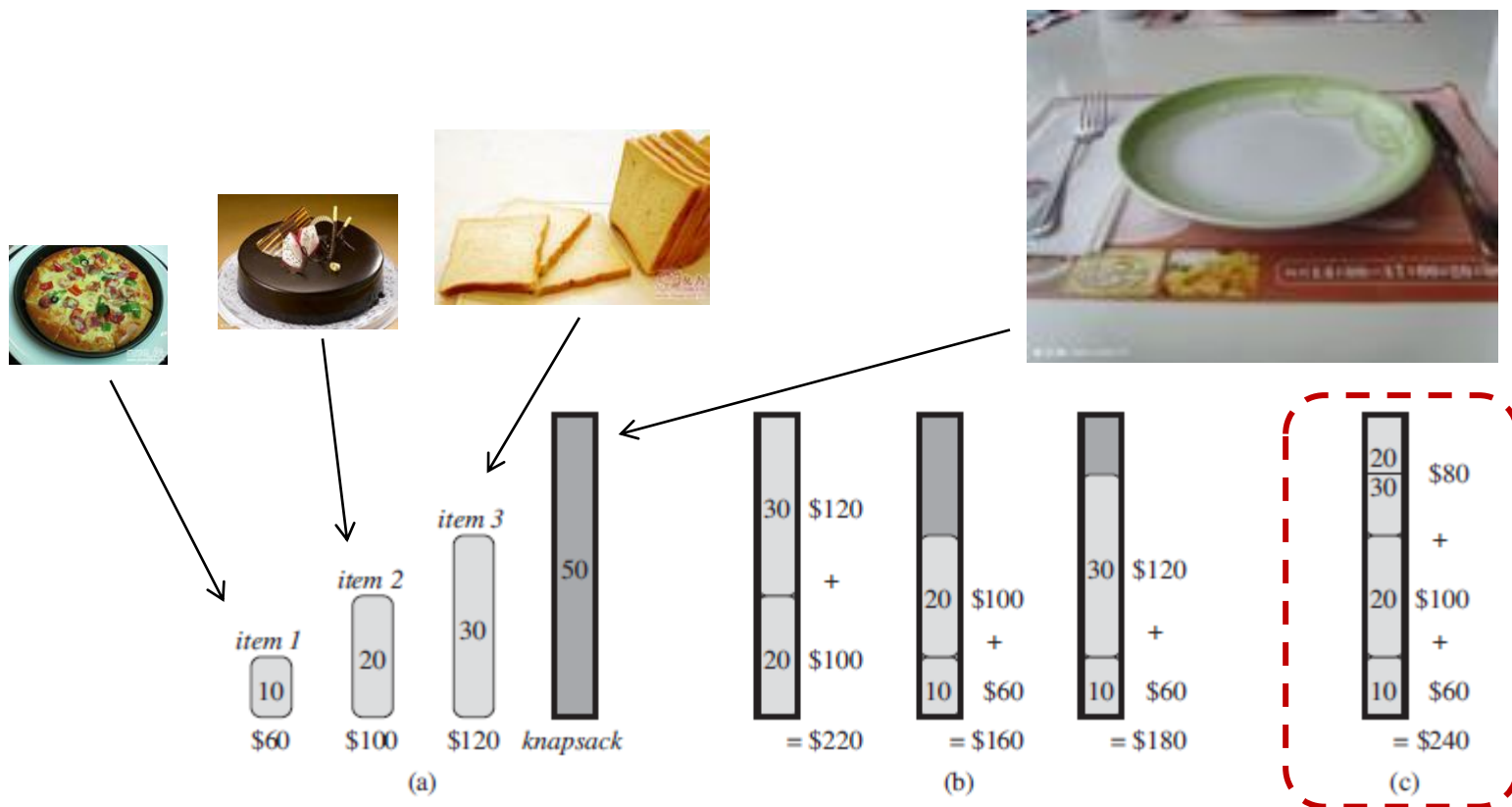
16.2.3 背包问题: 贪心 vs. 动态规划



• 0-1 背包

- ◆ n 个物品
- ◆ 物品 i 价值 $\$v_i$, 重量 w_i P
- ◆ 找到总重量 $\leq W$ 的最有价值的物品子集
- ◆ 要么拿要么不拿, 不能只拿一部分

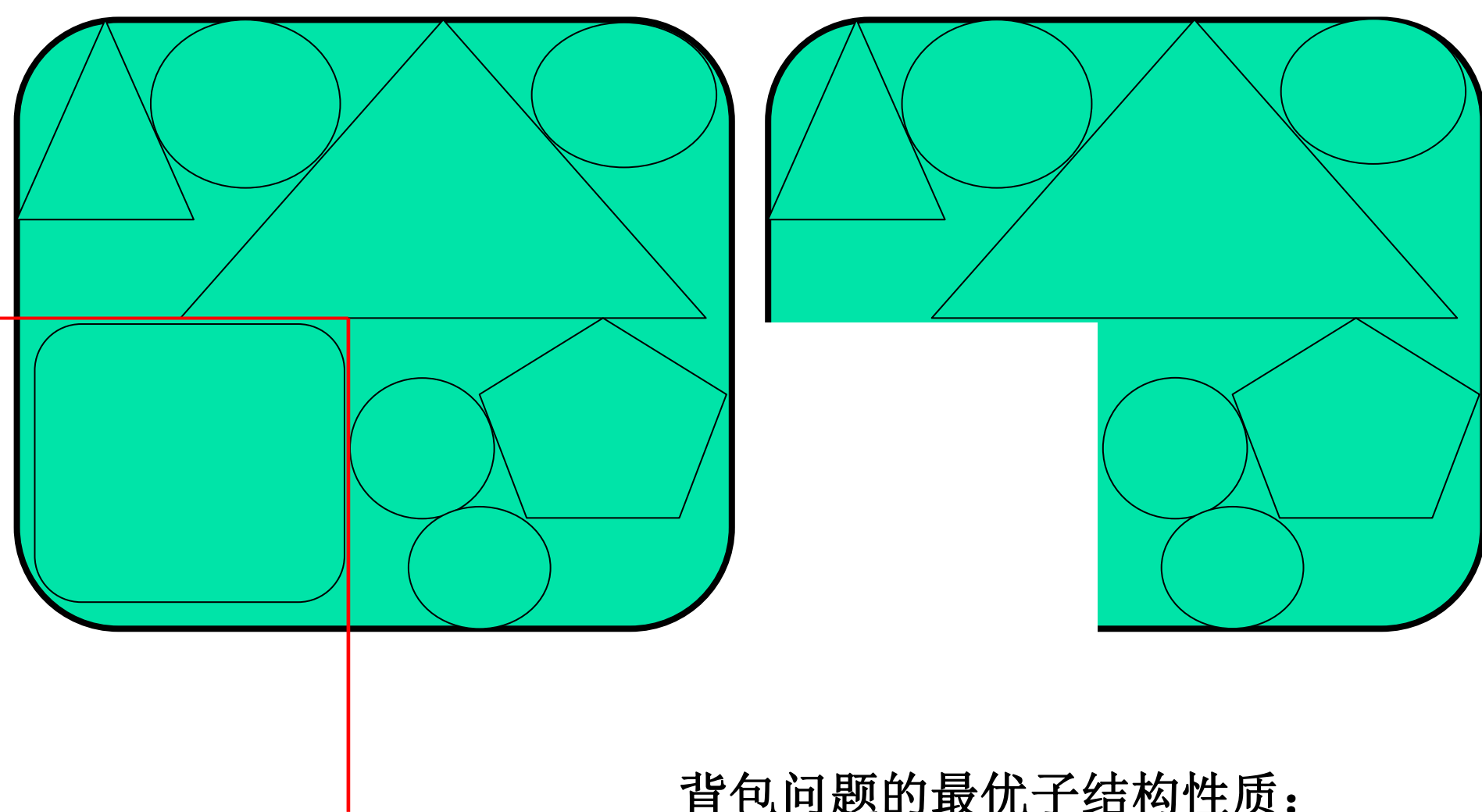
16.2.3 背包问题：贪心 vs. 动态规划



- 部分背包问题，小偷问题
 - 和 0-1 背包类似，但是可以只拿走物品的一部分。

16.2.3 背包问题：贪心 vs. 动态规划

- 0-1 背包问题
- 部分背包问题
- 两者都具有最优子结构
 - ◆ 0-1 背包: ?
 - ◆ 部分背包: ?



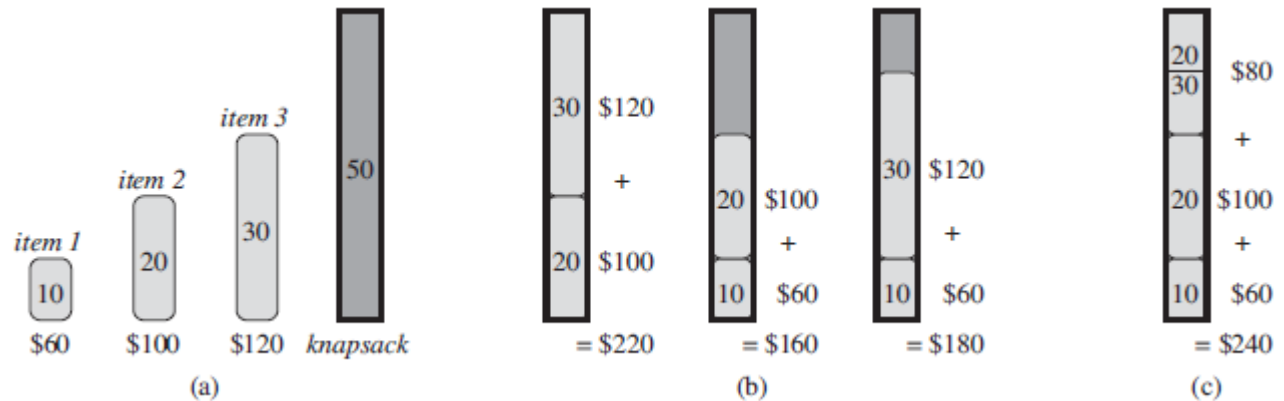
背包问题的最优子结构性质：

完整的圆角矩形框是一个最优背包，去掉右下角的红色部分剩下的部分是一个子背包，则该子背包也是一个最优背包。

16.2.3 背包问题：贪心 vs. 动态规划

- 0-1 背包问题
 - 部分背包问题
-
- 但是部分背包有贪心选择的性质，而 0-1 背包没有。

16.2.3 背包问题：贪心 vs. 动态规划



- 部分背包有贪心选择的性质，而 **0-1 背包** 没有。

- 为了求解部分背包问题，把物品按 v_i/w_i 降序排列

- 对于所有 i ，让 $v_i/w_i \geq v_{i+1}/w_{i+1}$

- Time:** $O(n \lg n)$ 进行排序, $O(n)$ 用来作贪心选择

```
FRACTIONAL-KNAPSACK( $v, w, W$ )
1   $load \leftarrow 0$ 
2   $i \leftarrow 1$ 
3  while  $load < W$  and  $i \leq n$ 
4      if  $w_i \leq W - load$ 
5          take all of item  $i$ 
6      else take  $W - load$  of  $w_i$  from item  $i$ 
7      add what was taken to  $load$ 
8       $i \leftarrow i + 1$ 
```


16.2.3 背包问题：贪心 vs. 动态规划

0-1 背包问题没有贪心选择的性质。

下面的例子取 $W = 50$ 。

- 贪心解：

- ◆ 取物品 1 和 2

- ◆ value = 160, weight = 30

剩余20磅容量。

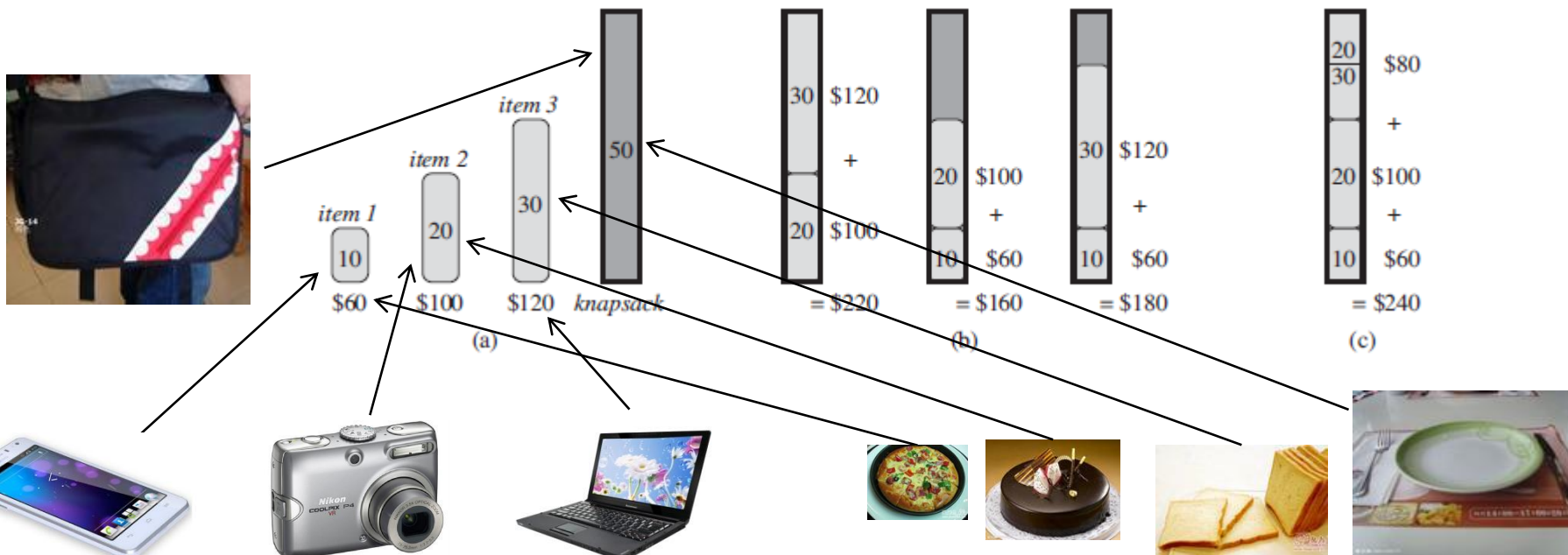
- 最优解：

- ◆ 取物品 2 和 3

- ◆ value=220, weight=50

没有剩余空间。

i	1	2	3
v_i	60	100	120
w_i	10	20	30
v_i/w_i	6	5	4



16 贪心算法

- 16.1, 活动安排问题
- 16.2, 贪婪算法的基本特征; 背包问题
- 16.3, 一个重要应用: 数据压缩编码设计 (哈夫曼编码)

A method for the construction of minimum-redundancy codes

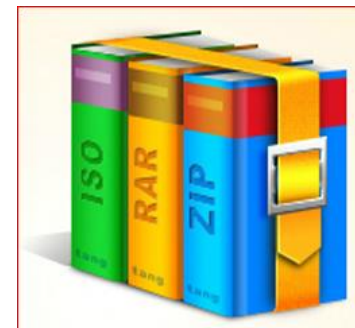
DA Huffman - Proceedings of the IRE, 1952 - ieeexplore.ieee.org

INTRODUCTION ONE IMPORTANT METHOD of transmitting messages is to transmit in their place sequences of symbols. If there are more messages which might be sent than there are kinds of symbols available, then some of the messages must use more than one symbol. If it is assumed that each symbol requires the same time for transmission, then the time for transmission (length) of a message is directly proportional to the number of symbols associated with it. In this paper, the symbol or sequence of symbols associated with a given message will be called the ...

☆ 被引用次数: 7783 相关文章 所有 7 个版本

16.3 哈夫曼编码

- 哈夫曼编码: 用于文件编码或数据压缩的一种广泛而有效的技术。
 - ◆ 节省 20% 到 90%



- 将数据视为一个字符序列

abaaaabbbdcffeaeeec...abaadefe



- 哈夫曼的贪心策略:

依据字符出现的频率表, 使用二进串来建立一种表示字符的最佳方法。

作业: 每人写
一个压缩软件

16.3 哈夫曼编码

- 希望储存100000个字符的数据文件。
只有六个不同字符出现。

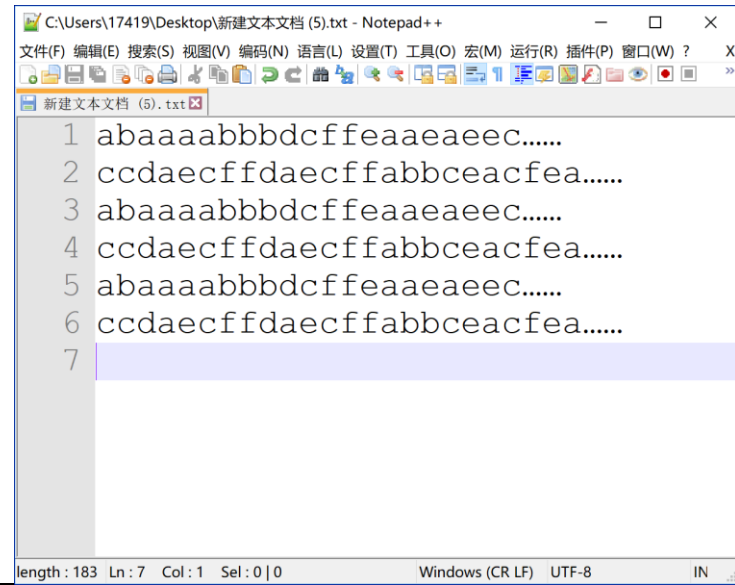
abaaaabbbdcffeaaeaeec.....

ccdaecffdaecffabbceacfea.....

频率表:

	a	b	c	d	e	f
频率 (千)	45	13	12	16	9	5
定长码	000	001	010	011	100	101
变长码	0	101	100	111	1101	1100

- 有许多方法(编码)来表示这样的信息文件
- binary character code* (or *code* for short): 每个字符由一个唯一的二进制字符串表示。
 - 定长码**: 如果使用3位定长码, 则文件可编码为30万比特。 我们能做得更好吗?



```
C:\Users\17419\Desktop\新建文本文档 (5).txt - Notepad++
文件(F) 编辑(E) 搜索(S) 视图(V) 编码(N) 语言(L) 设置(T) 工具(O) 宏(M) 运行(R) 插件(P) 窗口(W) ? X
新建文本文档 (5).txt
1 abaaaabbbdcffeaaeaeec.....
2 ccdaecffdaecffabbceacfea.....
3 abaaaabbbdcffeaaeaeec.....
4 ccdaecffdaecffabbceacfea.....
5 abaaaabbbdcffeaaeaeec.....
6 ccdaecffdaecffabbceacfea.....
7
length: 183 Ln: 7 Col: 1 Sel: 0 | 0 Windows (CR LF) UTF-8 IN
```

16.3 哈夫曼编码

- 100,000字符数据文件

	a	b	c	d	e	f
频率(千)	45	13	12	16	9	5
定长码	000	001	010	011	100	101
变长码	0	101	100	111	1101	1100

- binary character code (or code for short)*

- ◆ **变长码**: 高频出现的字符以短字码表示; 低频→长字码

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1,000 = 224,000 \text{ bits}$$

16.3 哈夫曼编码

- 100,000字符数据文件

	a	b	c	d	e	f
频率(千)	45	13	12	16	9	5
定长码	000	001	010	011	100	101
变长码	0	101	100	111	1101	1100

- binary character code (or code for short)*

- ◆ **定长码: 300,000 bits**
- ◆ **变长码: 224,000 bits**, 节省了25.3%. 这是这个文件的一种最佳编码方式。

16.3.1 前缀码

- 前缀码〔前缀无关码〕：没有字码是其他字码的前缀

	a	b	c	d	e	f
频率(千)	45	13	12	16	9	5
定长码	000	001	010	011	100	101
变长码	0	101	100	111	1101	1100

- 任何二进制字符码的编码都很简单
 - 连接表示每个字符的编码，如“abc”，它的变长前缀码是 $0 \cdot 101 \cdot 100 = 0101100$ ，我们使用 ‘ \cdot ’ 表示连接。
- 前缀码简化了译码

16.3.1 前缀码

- **前缀码〔前缀无关码〕**：没有字码是其他字码的前缀

	a	b	c	d	e	f
变长码	0	101	100	111	1101	1100

- 任何二进制字符码的编码都很简单
- 前缀码**简化了译码**
 - ◆ 由于没有任何码字是任何其他码字的前缀，因此编码文件开头的码字是明确的。
 - ◆ 我们可以轻易地识别初始码字，将其翻译回原始字符，并对编码文件的其余部分重复解码过程。
 - ◆ Exam: 001011101 仅能识别为 0·0·101·1101,解码为“aabe”.

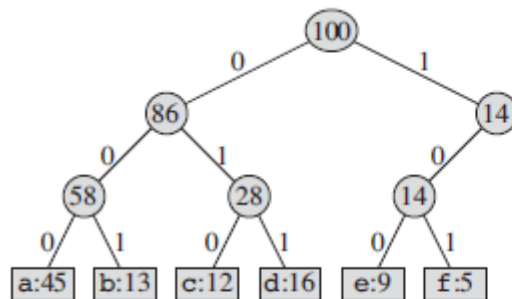
16.3.1 前缀码

a	b	c	d	e	f
0	101	100	111	1101	1100

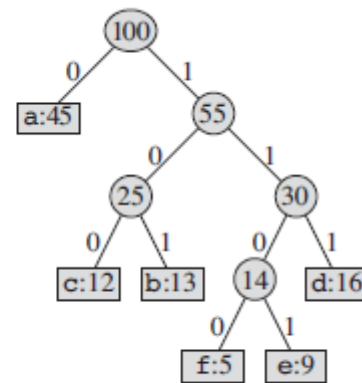
001011101

uniquely as 0·0·101·1101,

which decodes to “aabe”.



(a)



(b)

解码

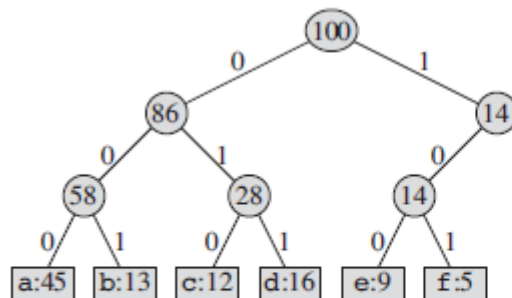
- 解码过程需要一种前缀码的简便表示来方便解码。
- 二叉树是一种方便的表示方法，树叶为给定字符，从树根到树叶的过程就是解码过程。

16.3.1 前缀码

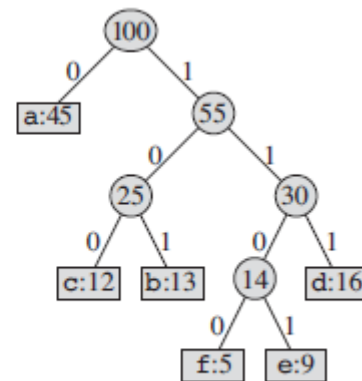
a	b	c	d	e	f
0	101	100	111	1101	1100

001011101

uniquely as 0·0·101·1101,
which decodes to “aabe”.



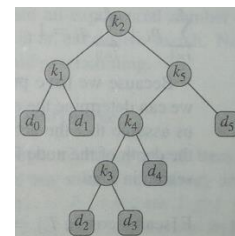
(a)



(b)

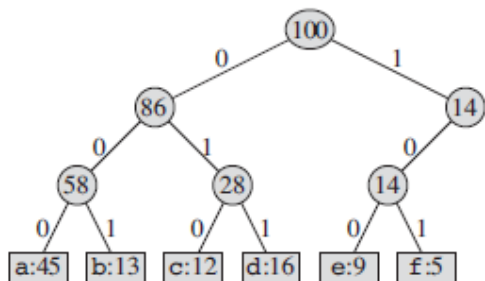
解码

- 字符的编码为一条从树根到树叶路径。
- 不是二叉搜索树, 叶子不需要按顺序出现, 中间节点不包含字符。

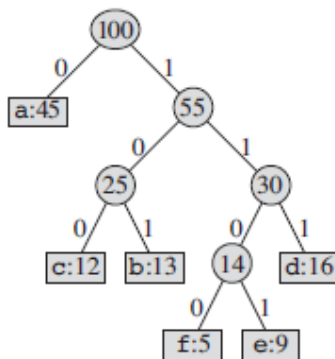


注意：不要混淆各种二叉树，如，最大（小）堆、二叉搜索树，哈夫曼树

16.3.1 前缀码



(a)



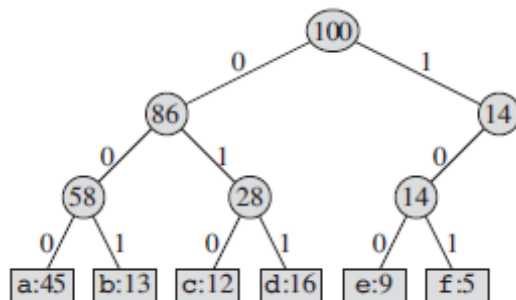
(b)

full binary tree 满二叉树

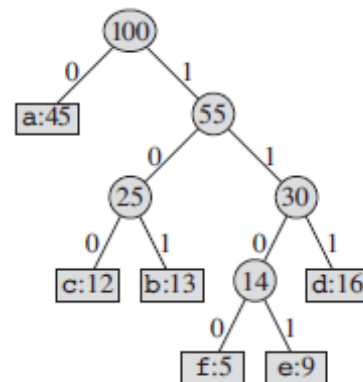
- 国际定义：除叶子节点外，所有节点都有两个孩子。
- 国内定义：除了满足如上定义，所有叶子节点还需要在同一层上。

- 文件的最佳编码总是用**满**二叉树来表示, 每个非叶节点都有两个子节点(Ex16.3-1). 定长码在我们的例子中并不是最优的。
- 我们可以把注意力放在满二叉树上
 - ◆ C 是字母表,
 - ◆ 所有字符频率 > 0
 - ◆ 一个最佳前缀码树有 $|C|$ 片叶子, C 中每个字母都是一片, 正好 $|C|-1$ 个中间节点。

16.3.1 前缀码



(a)



(b)

计算编码文件所需的比特数

- 对于C中的每个字符 c 给定与前缀码对应的树 T ,
 - ◆ $f(c)$: 文件中 c 的频率
 - ◆ $d_T(c)$: c 的叶在树中的深度 (c 的编码长度)。编码所需的比特数

$$B(T) = \sum_{c \in C} f(c) d_T(c) \quad (16.5)$$

我们定义为树 T 的开销。

16.3.2 构造哈夫曼编码

哈夫曼编码:
构造最优前缀码的
贪心策略

```
HUFFMAN(C)
1   $n \leftarrow |C|$ 
2   $Q \leftarrow C$ 
3  for  $i \leftarrow 1$  to  $n - 1$ 
4      allocate(分配) a new node  $z$ 
5       $left[z] \leftarrow x \leftarrow \text{EXTRACT-MIN}(Q)$ 
6       $right[z] \leftarrow y \leftarrow \text{EXTRACT-MIN}(Q)$ 
7       $f[z] \leftarrow f[x] + f[y]$ 
8      INSERT( $Q, z$ )
9  return EXTRACT-MIN( $Q$ ) //return the root of the tree.
```

- C : n 个字符的集合, $c \in C$: 频率为 $f[c]$ 的对象。
 - ◆ 建立与最优码对应的树 T 。
 - ◆ 从 $|C|$ 个叶开始, 执行 $|C|-1$ 次“合并”操作。
 - ◆ 以 f 为键的**最小优先级队列** Q 用于识别两个频率最低的对象来合并。合并结果为一个新对象, 频率为合并的两个对象之和。

16.3.2 构造哈夫曼编码

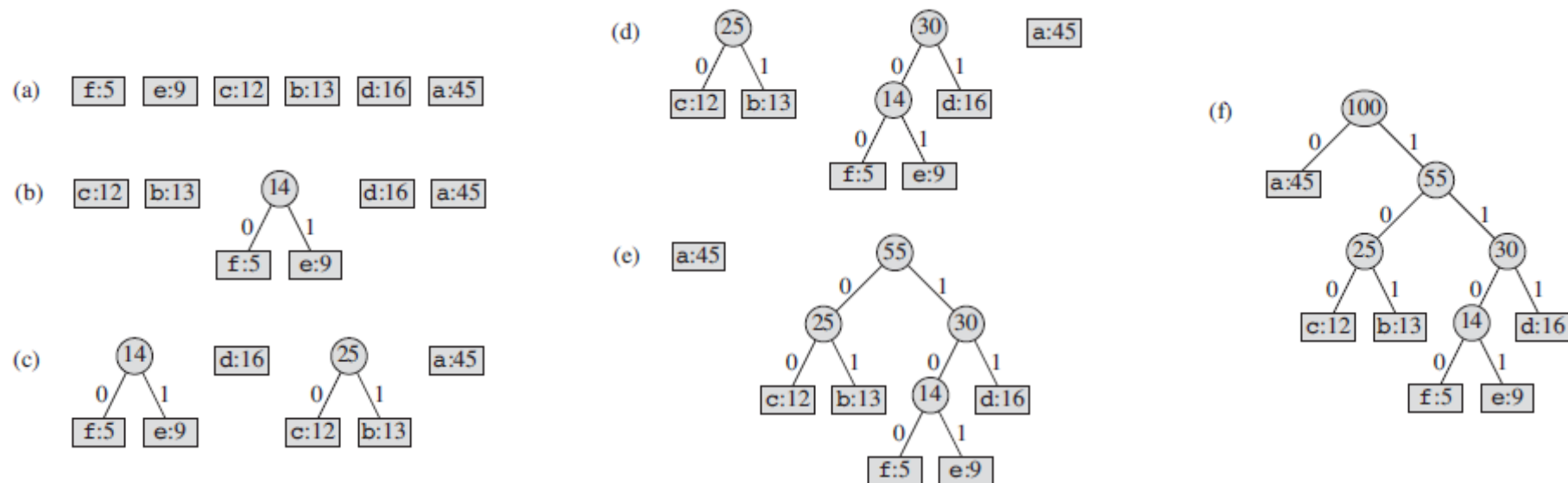
Example:

霍夫曼算法执行，
6 个字母,5 步合并。
最终生成的树
代表最优前缀码。

HUFFMAN(C)

```
1  $n \leftarrow |C|$ 
2  $Q \leftarrow C$ 
3 for  $i \leftarrow 1$  to  $n - 1$ 
4   allocate(分配) a new node  $z$ 
5    $left[z] \leftarrow x \leftarrow \text{EXTRACT-MIN}(Q)$ 
6    $right[z] \leftarrow y \leftarrow \text{EXTRACT-MIN}(Q)$ 
7    $f[z] \leftarrow f[x] + f[y]$ 
8    $\text{INSERT}(Q, z)$ 
9 return  $\text{EXTRACT-MIN}(Q)$  //return the root of the tree.
```

Running time ?



16.3.3 哈夫曼算法的正确性*

- 最佳前缀码的确定问题具有贪心选择和最佳子结构的性质。

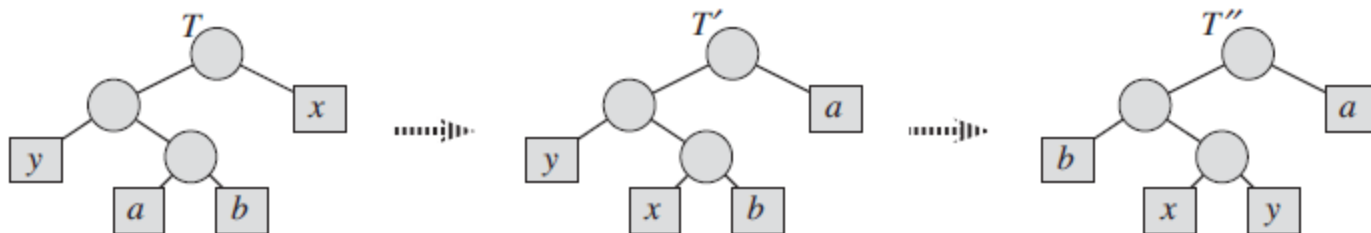
- 引理 16.2 (贪心选择性质)

C 是字母表, 每个字符 $c \in C$ 频率为 $f[c]$ 。

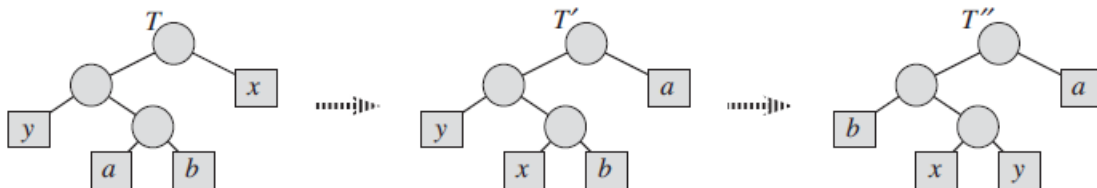
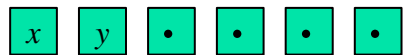
$x, y \in C$, 频率最低。那就存在 C 的最佳前缀码, x 和 y 编码长度相同, 只有最后一位不同。

Proof idea: 取表示任意最优前缀码的树 T , 修改它使其成为表示另一个最优前缀码的树, 使 x 和 y 作为新树中深度最大的姐妹叶出现。

x y \bullet \bullet \bullet \bullet



16.3.3 哈夫曼算法的正确性*



● 引理 16.2

$c \in C$ 频率为 $f[c]$ 。 $\mathbf{x, y} \in C$, 频率最低。存在 C 的最佳前缀码, \mathbf{x} 和 \mathbf{y} 编码长度相同, 只有最后一位不同。

$$B(T) = \sum_{c \in C} f(c) d_T(c) \quad (16.5)$$

Proof: 令 $\mathbf{a, b}$ 为最优 T 中最深的姐妹叶。假设 $f[a] \leq f[b], f[x] \leq f[y]$ 。 $f[x]$ 和 $f[y]$ 是两个最低的叶频率, $f[a], f[b]$ 是两个任意频率, 依次为, $\Rightarrow f[x] \leq f[a], f[y] \leq f[b]$. 交换 \mathbf{a} 和 \mathbf{x} 在 T 中的位置来生成树 T' , 然后交换 \mathbf{b} 和 \mathbf{y} 在 T' 中的位置来生成树 T'' . 通过(16.5), 我们得到

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} f(c) d_T(c) - \sum_{c \in C} f(c) d_{T'}(c) \\ &= f[x] d_T(x) + f[a] d_T(a) - f[x] d_{T'}(x) - f[a] d_{T'}(a) \\ &= f[x] d_T(x) + f[a] d_T(a) - f[x] d_T(a) - f[a] d_T(x) \\ &= (f[x] - f[a]) d_T(x) + (f[a] - f[x]) d_T(a) \\ &= (f[a] - f[x]) (d_T(a) - d_T(x)) \geq 0 \end{aligned}$$

类似, $B(T') - B(T'') \geq 0$,

那么, $B(T'') \leq B(T)$.

因为 T 最优, $B(T) \leq B(T'')$.

故 $B(T'') = B(T)$.

因此, T'' 是最优树。

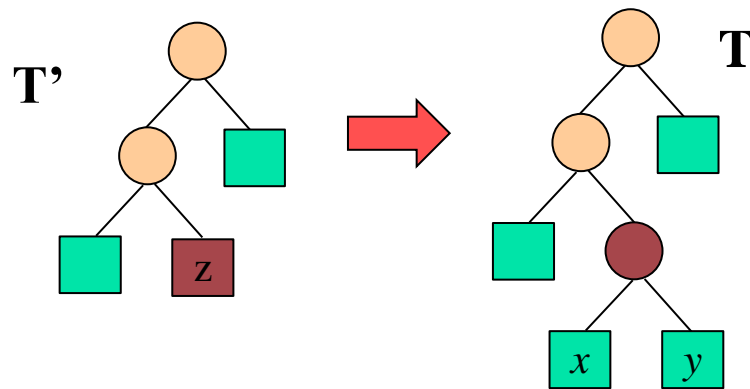
16.3.3 哈夫曼算法的正确性*

- 引理 16.3 (最优子结构性质) ?

给定字母表集 C ，每一个字符 $c \in C$ 的频率为 $f[c]$ 。 x 和 $y \in C$ 有最小频率。从 C 中抽取字符 x 和 y ，但增加新字符 z 到 C 中，得到新的字符集 C' ，即 $C' = C - \{x, y\} \cup \{z\}$ 。除 $f[z] = f[x] + f[y]$ 以外， f 在 C' 中的定义与在 C 中相同。若 T' 为 C' 的最优前缀无关编码，则 T 为关于 C 的最优前缀无关编码，其中， T 为把 T' 的叶节点 z 代替为以 x 和 y 作为叶节点的内点变换而来。

$C : \{c_1, \dots, c_m, x, y\}, C' : \{c_1, \dots, c_m, z\},$

由optimal T' 能构成 optimal T



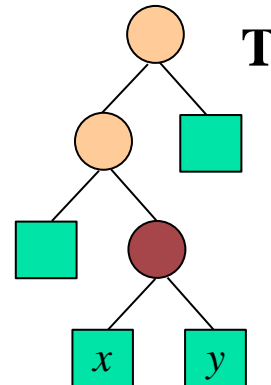
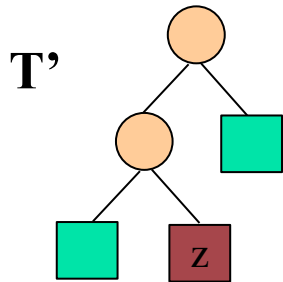
16.3.3 哈夫曼算法的正确性*

- 引理 16.3 (最优子结构性质)

Proof : 对每个 $c \in C - \{x, y\}$, 我们有 $d_T(c) = d_{T'}(c)$, 那么 $f[c]d_T(c) = f[c]d_{T'}(c)$. 因为 $d_T(x) = d_T(y) = d_{T'}(z) + 1$, 我们得到

$f[x]d_T(x) + f[y]d_T(y) = (f[x] + f[y])(d_{T'}(z) + 1) = f[z]d_{T'}(z) + (f[x] + f[y])$,
可以推断出 $B(T) = B(T') + f[x] + f[y]$.

($B(T) = f[x]d_T(x) + f[y]d_T(y) + f[c]d_T(c)$, $B(T') = f[z]d_{T'}(z) + f[c]d_{T'}(c)$)

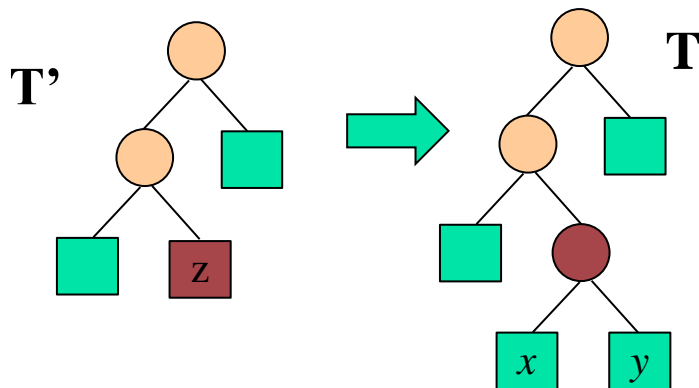


Detail of proof? Next slides.

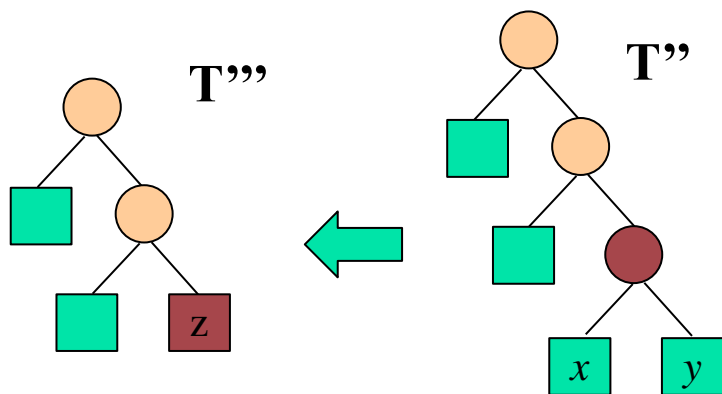
16.3.3 哈夫曼算法的正确性*

- 引理 16.3 (最优子结构性质)

$C : \{c_1, \dots, c_m, x, y\}, C' : \{c_1, \dots, c_m, z\}$, 由 optimal T' 能构成 optimal T



Here, $B(T) = B(T') + f[x] + f[y]$



假设 T 不是最优的, T'' 是。那么 $B(T'') < B(T)$. 由引理 16.2, T'' 中 x 和 y 是姐妹叶。令 T''' 为树 T'' 中将 x 和 y 的父节点用频率 $f[z]=f[x]+f[y]$ 的叶 z 替代。那么

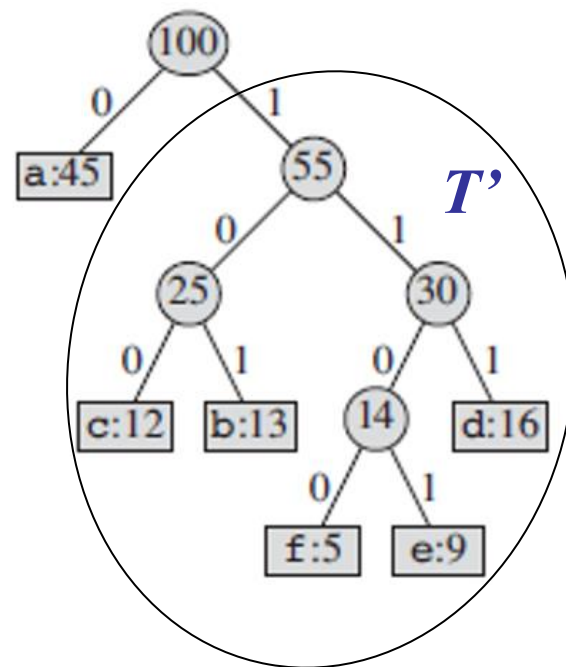
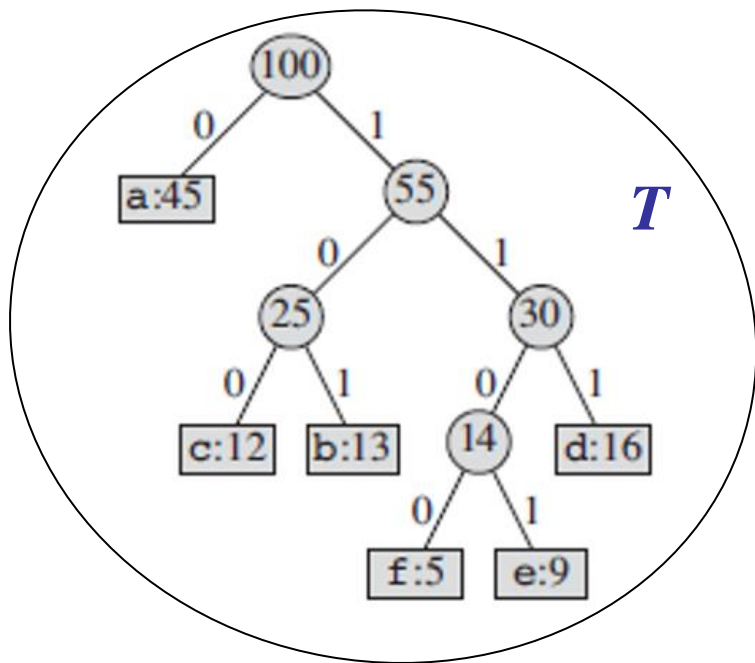
$$B(T''') = B(T'') - f[x] - f[y]$$

$$< B(T) - f[x] - f[y] = B(T'),$$

与 T' 表示 C' 的最优前缀码矛盾. 因此, T 必须表示字母表 C 的最优前缀码。

16.3.3 哈夫曼算法的正确性*

- 引理16.3 (最优子结构性质) 另一种解释



若 T 是最优解, 那么 T' 也是最优的

16.3.3 哈夫曼算法的正确性*

□ Theorem 16.4

HUFFMAN 过程产生最优前缀码。

Proof 引理 16.2（每一次选择是贪婪的、是正确的）和引理 16.3（确保由子问题的最优解能导出原问题的最优解）。

Exercises

$$c[i, j] = \begin{cases} 0 & , \text{ if } S_{ij} = \emptyset, \\ \max_{i < k < j} \{c[i, k] + c[k, j] + 1\} & , \text{ if } S_{ij} \neq \emptyset. \end{cases} \quad (16.3)$$

- 能很轻易的给问题16.3设计一个基于递归的算法。
 - (1) 直接递归算法? 复杂度?
 - (2) 动态规划算法? 复杂度?
- 对于问题16.3:
 - ◆ 有多少种选择?
 - ◆ 一个选择有多少子问题?
- 能简化解法吗?

16.1.3 把 DP 解法转换为贪心解法

Exercises

16.1-1,

16.1-2 (最晚开始优先原则)

16.2-2

给出零一背包的动态规划解法，时间复杂度为 $O(n W)$ ， n 是物品数量， W 小偷能拿取物品的最大重量。

16.3-2 (课堂作业)

基于前八个斐波那契数，对于下面的频率集合，最佳的霍夫曼编码是什么？

a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

又一些大作业或讨论题提示:

- 活动安排、Huffman code等，是否都能描述为背包问题？
- 本书有多少算法是用的贪心策略（小论文：贪心算法十个经典问题？）
- 哈夫曼编码（用哈夫曼压缩方法，设计一个压缩软件：测试一下，算法导论这本书的压缩率能到多少？）
- **OBST**（最优二叉搜索树构建（以某本书里的词汇为基础？））
- 活动安排、分数背包
- **0-1**背包、钢管切割、ALS、MCM、LCS、最短路径
- 雇佣（雇佣多少人）、取帽子、相同生日
- **RSA**加密解密、**FFT**、串匹配、计算几何、最大流
- 算法实验室（问题求解工具、算法效果展示平台、多种算法时间复杂度对比分析、多种算法空间复杂度对比分析、**IO**导入、.....。支持穷举、递归、回溯、分治、**DP**、贪心；排序、查找；随机；图、树；等等若干方法（算法）的仿真演示。）