



HITWH  
SE

# 第五章

## 贪心算法



5.1 贪心算法的基本原理

5.2 活动选择问题

5.3 Huffman 编码

5.4 最小生成树问题

5.5 Minimum Makespan Scheduling



**HITWH**  
**SE**

**参考资料**



# **Introduction to Algorithms**

**Chapter 16 Pages 370-405**

**Chapter 23 Pages 561-579**



## 5.1 贪心算法基本原理

- Greedy算法的基本概念
- Greedy算法与动态规划方法的比较
- Greedy算法的设计步骤



# Greedy算法的基本概念

- 顾名思义，贪心算法总是作出在当前看来最好的选择
- 也就是说贪心算法并不从整体最优考虑，它所作出的选择只是在某种意义上的局部最优选择
- 当然，希望贪心算法得到的最终结果也是整体最优的

— 兑换硬币问题



# Greedy算法的基本概念

## • 应用实例1

### - 兑换硬币问题

已知有5种不同面值的硬币：1元、2角5分、1角、5分、1分

欲兑换钱数：6角7分

目标：用于兑换的硬币个数最少

如何兑换？

1. 穷举所有可能性：代价高！
2. 贪心策略：按照面值从大到小选择硬币兑换

2角5分 : 2枚  
1角 : 1枚  
5分 : 1枚  
1分 : 2枚

贪心策略

每次尽可能选择面额最大的硬币

即：当前看来最优的选择

得到的兑换结果是最优解么？

是否总能得到最优解呢？



# Greedy算法的基本概念

- 应用实例1

- 兑换硬币问题

若不同面值的硬币为：1角1分、5分、1分

欲兑换钱数：1角5分

目标：用于兑换的硬币个数最少

贪心策略：每次尽可能选择面额最大的硬币

即：当前看来最优的选择

1角1分 : 1枚

1分 : 4枚

得到的兑换结果是最优解么？ No!



# Greedy算法的基本概念

- 贪心算法的用途
  - 求解最优化问题
- 贪心算法的基本思想
  - 求解最优化问题的算法包含一系列步骤
  - 每一步都有一组选择
  - 作出在当前看来最好的选择
  - 希望通过作出局部优化选择达到全局优化选择

最终不一定得到全局最优解！

可能存在多种贪心策略！



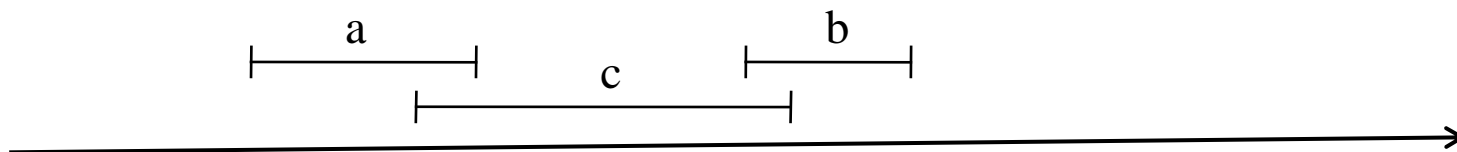


- 应用实例2

- 区间调度(活动选择)问题

输入:  $n$ 个活动区间的集合 $\{[s_1, f_1], [s_2, f_2], \dots, [s_n, f_n]\}$ ,  
 $s_i$ 是区间 $i$ 的起始时间,  $f_i$ 是终止时间,

输出: 具有最多相容区间(活动)的调度



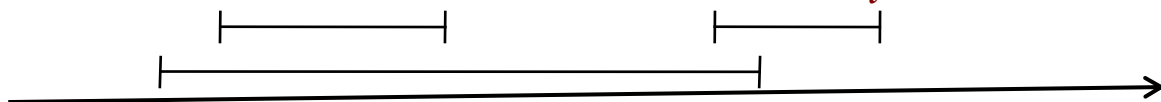


# Greedy算法的基本概念

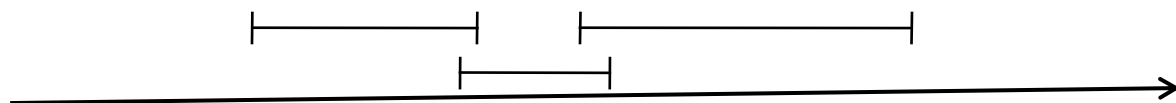
## • Greedy算法的实例

### — 如何贪心选择？

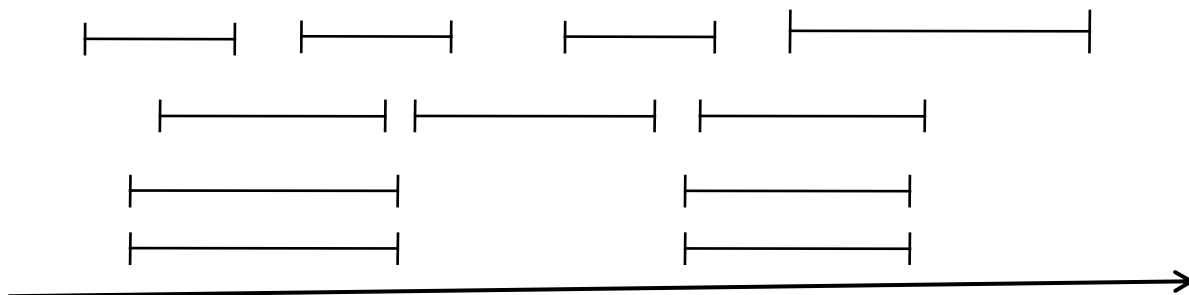
- 选择具有最小开始时间 $s_i$ 的区间？



- 选择具有最短时长 $f_i - s_i$ 的区间？



- 具有最少冲突的区间？



Greedy算法不一定  
产生最优解



# Greedy算法的基本概念

- 贪心算法不能对所有问题都得到整体最优解，但对许多问题它能产生整体最优解，如：单源最短路径问题、最小生成树问题等
- 在一些情况下，即使贪心算法不能得到整体最优解，其最终结果却是最优解的很好近似

什么情况下可以产生最优解呢？？



- Greedy 算法产生优化解的条件

- 优化子结构

- 若一个优化问题的优化解包含它的(剩余)子问题的优化解，则称其具有优化子结构

- Greedy 选择性 (Greedy-choice property)

- 一个优化问题的全局优化解可以通过局部优化选择得到



- Greedy选择性的证明

- 归纳法

- 对算法步数归纳或问题规模归纳

- 证明在每一步做得都比其它算法好，从而最终产生了一个最优解

- 交换论证法

- 在保证最优性不变前提下，从一个最优解逐步替换，最终得到贪心算法的解

- 证明贪心算法一定能够找到一个至少与其它最优解一样优化的解
      - 贪心选择性做出的选择至少包含于一个优化解中



# 与动态规划方法的比较

- 动态规划方法

- 以自底向上方式，先解小子问题，再求解大子问题
- 在每一步所做的选择通常依赖于子问题的解

- Greedy方法

- 以自顶向下方式，逐步进行贪心选择，不断减少子问题规模
- 在每一步先做出当前看起来最好的选择
- 然后再求解本次选择后产生的剩余子问题
- 每次选择既不依赖于子问题的解，也不依赖于未来的选择



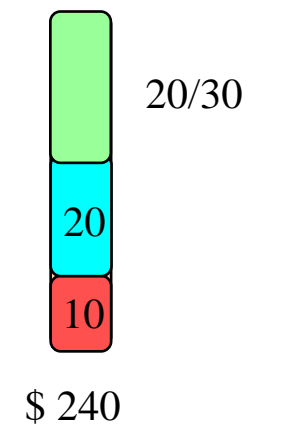
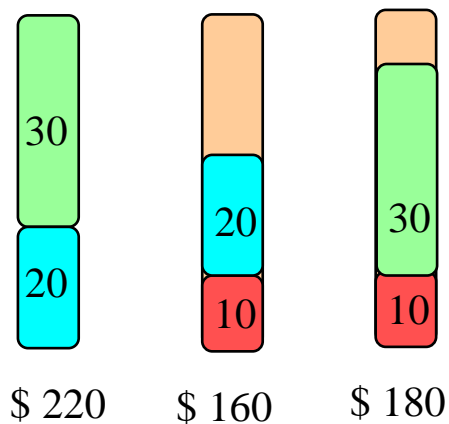
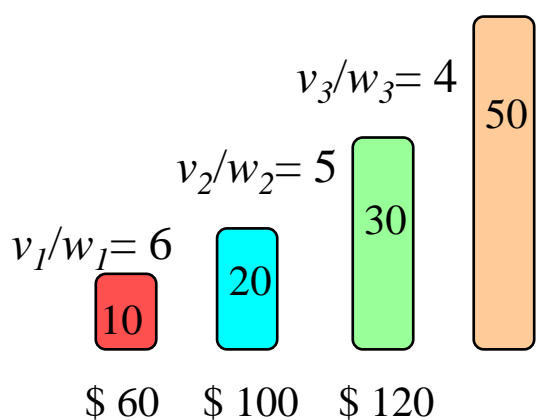
# 与动态规划方法的比较

- 动态规划方法可用的条件
  - 优化子结构
  - 子问题重叠性
- Greedy方法可用的条件
  - 优化子结构
  - Greedy选择性
- 可用Greedy方法时，动态规划方法可能不适用
- 可用动态规划方法时，Greedy方法可能不适用



# 与动态规划方法的比较

- 例如：0-1背包问题与部分背包问题
  - 都具有优化子结构
  - 但是，部分背包问题可用贪心策略解决，而0-1背包问题却不行！
    - 计算每个物品每磅价值 $v_i/w_i$ ，并按照每磅价值由大到小顺序取物品



0-1背包问题

部分背包问题





# 准确Greedy算法的设计步骤

## 1. 设计贪心选择方法：

- 贪心选择方法
- 剩余子问题

很重要！  
决定能否得到  
全局最优解

2. 证明：对于1中贪心选择来说，所求解的问题具有优化子结构

3. 证明：对于1中贪心选择来说，所求解的问题具有Greedy选择性

4. 按照1中设计的贪心选择方法设计算法



## 5.2 An activity-selection problem

- 问题定义
- 问题求解
  - 设计贪心选择方法
  - 优化解的结构分析
  - Greedy选择性证明
  - 算法设计
  - 算法复杂性分析

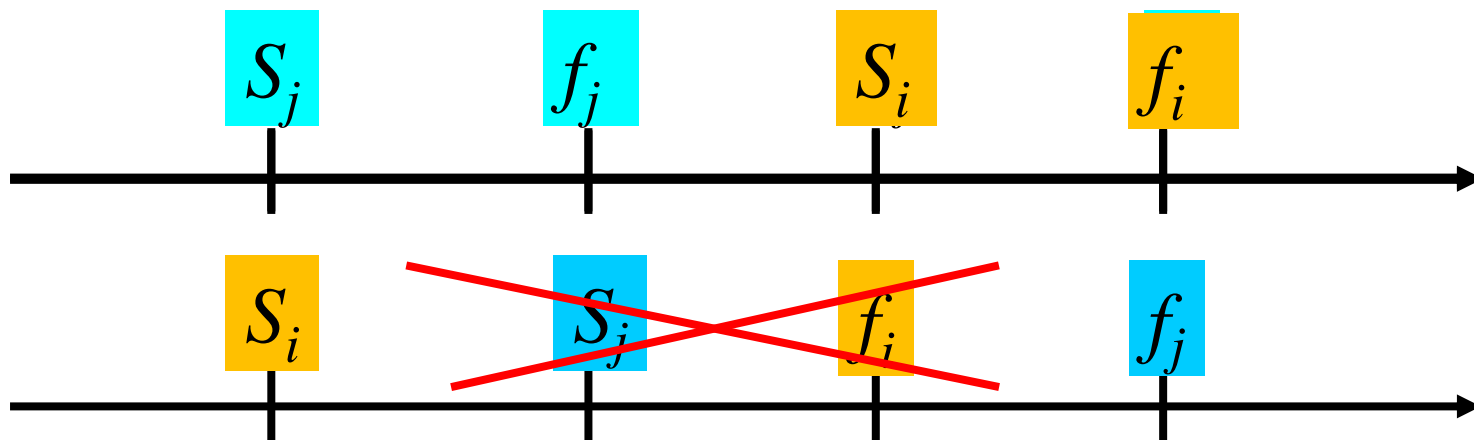


## • 活动

- 设  $S = \{1, 2, \dots, n\}$  是  $n$  个活动的集合，所有活动共享一个资源，该资源同时只能为一个活动使用
- 每个活动  $i$  有起始时间  $s_i$ ，终止时间  $f_i$ ， $s_i \leq f_i$

## • 相容活动

- 活动  $i$  和  $j$  是相容的，若  $s_j \geq f_i$  或  $s_i \geq f_j$ ，即





- 活动选择问题

- 输入:  $S = \{1, 2, \dots, n\}$ ,

$$F = \{ [s_i, f_i] \}, \quad n \geq i \geq 1$$

- 输出:  $S$  中的最大相容活动集合



- 贪心思想

为了选择最多的相容活动，每次选 $f_i$ 最小的活动，使我们能够选更多的活动

剩余子问题：

$$S_i = \{ j \in S \mid s_j \geq f_i \}$$

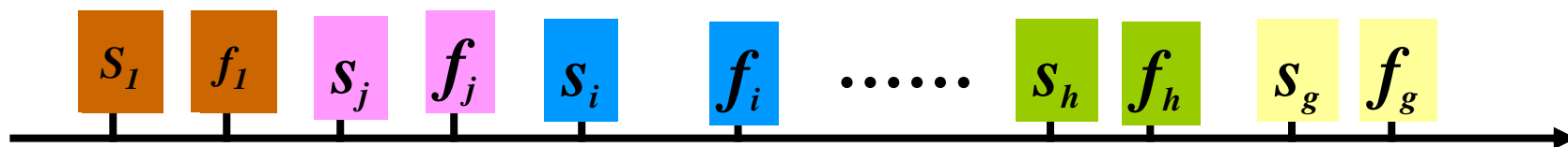


## 引理1说明贪心选择包含在某个优化解中

**引理1** 设 $S=\{1,2,\dots,n\}$ 是 $n$ 个活动集合,  $[s_i, f_i]$ 是活动 $i$ 的起始终止时间, 且 $f_1 \leq f_2 \leq \dots \leq f_n$ .

则 $S$ 的活动选择问题的某个优化解包括活动1.

**证** 设 $A$ 是一个优化解, 按结束时间排序 $A$ 中活动, 设其第一个活动为 $k$ , 第二个活动为 $j$ . .....



如果 $k=1$ , 引理成立.

如果 $k \neq 1$ , 令 $B=A-\{k\} \cup \{1\}$ ,

由于 $A$ 中活动相容,  $f_1 \leq f_k \leq s_j$ ,  $B$ 中活动相容.

因为 $|B|=|A|$ , 所以 $B$ 是一个优化解, 且包括活动1.

**引理2.** 设  $S = \{1, 2, \dots, n\}$  是  $n$  个活动集合,  $[s_i, f_i]$  是活动  $i$  的起始终止时间, 且  $f_1 \leq f_2 \leq \dots \leq f_n$ , 设  $A$  是  $S$  的调度问题的一个优化解且包括活动  $1$ , 则  $A' = A - \{1\}$  是  $S' = \{i \in S / s_i \geq f_1\}$  的调度问题的优化解.

**证.** 显然,  $A'$  中的活动是相容的.

我们仅需要证明  $A'$  是最大的.

设不然, 存在一个  $S'$  的活动选择问题的优化解  $B'$ ,  $|B'| > |A'|$ .

令  $B = \{1\} \cup B'$ . 对于  $\forall i \in S'$ ,  $s_i \geq f_1$ ,  $B$  中活动相容.

$B$  是  $S$  的一个解.

由于  $|A| = |A'| + 1$ ,  $|B| = |B'| + 1 > |A'| + 1 = |A|$ , 与  $A$  最大矛盾.

**引理2说明活动选择问题具有优化子结构**



**引理3.** 设  $S=\{1, 2, \dots, n\}$  是  $n$  个活动集合,  $f_1 \leq f_2 \leq \dots \leq f_n$ ,  $f_{l_0}=0$ ,  $l_i$  是  $S_i=\{j \in S \mid s_j \geq f_{l_{i-1}}\}$  中具有最小结束时间  $f_{l_i}$  的活动, 令  $k = \min\{i \mid S_{i+1} = \emptyset\}$ . 则  $A = \cup_{i=1}^k \{l_i\}$  是  $S$  的优化解

$$S_1=\{1,2,3,4,5,6,7,8,9,10,11\}, l_0=0$$

$$S_2=\{4,6,7,8,9,11\}, l_1=1$$

$$S_3=\{8,9,11\}, l_2=4$$

$$S_4=\{11\}, l_3=8$$

$$S_5=\emptyset, l_4=11$$

$$l_1=1$$

$$l_2=4$$

$$l_3=8$$

$$l_4=11$$

$$S_1=\{j \in S \mid s_j \geq f_{l_0}=0\}$$

$$S_2=\{j \in S \mid s_j \geq f_{l_1}=f_1\}$$

$$S_3=\{j \in S \mid s_j \geq f_{l_2}\}$$

.....

$$S_k=\{j \in S \mid s_j \geq f_{l_{k-1}}\}$$

$$S_{k+1}=\{j \in S \mid s_j \geq f_{l_k}\}=\emptyset$$

$i$	1	2	3	4	5	6	7	8	9	10	11
$s_i$	1	3	3	5	3	5	6	8	8	2	12
$f_i$	4	5	6	7	8	9	10	11	12	13	14





**引理3.** 设  $S = \{1, 2, \dots, n\}$  是  $n$  个活动集合,  $f_1 \leq f_2 \leq \dots \leq f_n$ ,  $f_{l_0} = 0$ ,  $l_i$  是  $S_i = \{j \in S \mid s_j \geq f_{l_{i-1}}\}$  中具有最小结束时间  $f_{l_i}$  的活动, 令  $k = \min\{i \mid S_{i+1} = \emptyset\}$ . 则  $A = \bigcup_{i=1}^k \{l_i\}$  是  $S$  的优化解

最优解由引理3中的构造方法得到的  $l_i$  构成

**证. 思路:** 证明存在一个优化解  $A$  等于  $\bigcup_{i=1}^k \{l_i\}$

对  $|\bigcup_{i=1}^k \{l_i\}|$  作归纳法.

当  $|\bigcup_{i=1}^k \{l_i\}| = 1$  时, 由 **引理1**, 命题成立.

设  $|\bigcup_{i=1}^k \{l_i\}| \leq m-1$  时, 命题成立,  $1 \leq m \leq k$ .

当  $|\bigcup_{i=1}^k \{l_i\}| = m$  时, 由 **引理2**, 存在优化解  $A = \{l_1 = I\} \cup A_1$ .

其中,  $A_1$  是  $S_2 = \{j \in S \mid s_j \geq f_{l_1} = f_1\}$  的优化解,

而且  $|\bigcup_{i=2}^m \{l_i\}| = m-1$ . 由归纳假设, 存在优化解  $A_1 = \bigcup_{i=2}^m \{l_i\}$ .

于是,  $A = \bigcup_{i=1}^m \{l_i\}$  是  $S$  的优化解,  $m=k$  时, 引理3得证.

$$S_1 = \{j \in S \mid s_j \geq f_{l_0} = 0\}$$

$$S_2 = \{j \in S \mid s_j \geq f_{l_1} = f_1\}$$

$$S_3 = \{j \in S \mid s_j \geq f_{l_2}\}$$

.....

$$S_k = \{j \in S \mid s_j \geq f_{l_{k-1}}\}$$

$$S_{k+1} = \{j \in S \mid s_j \geq f_{l_k}\} = \emptyset$$



- 贪心选择方法

- 选择:

- 每次选择具有最小结束时间的活动 $f_i$

- 剩余子问题:

- $S_i = \{ j \in S \mid s_j \geq f_i \}$

# • 算法

## 算法及复杂性分析



(设  $f_1 \leq f_2 \leq \dots \leq f_n$  已排序)

Greedy-Activity-Selector( $S, F$ )

$n \leftarrow \text{length}(S);$

$A \leftarrow \{1\}$

$j \leftarrow 1$

For  $i \leftarrow 2$  To  $n$  Do

    If  $s_i \geq f_j$

        Then  $A \leftarrow A \cup \{i\}; j \leftarrow i;$

Return  $A$

- 如果结束时间已排序

$$T(n) = \theta(n)$$

- 如果 结束时间未排序

$$\begin{aligned} T(n) &= \theta(n) + \theta(n \log n) \\ &= \theta(n \log n) \end{aligned}$$



**定理1.** Greedy-Activity-Selector 算法能够产生最优解.

**证.**

- (1) 由引理2可知活动选择问题具有**优化子结构**
- (2) 由引理1知贪心选择方法具有**Greedy选择性**
- (3) Greedy-Activity-Selector 算法**确实按照**引理3的**Greedy选择性**进行局部优化选择.



## 5.3 Huffman codes

- 问题定义
- 问题求解
  - 设计贪心选择方法
  - 优化解的结构分析
  - Greedy选择性证明
  - 算法设计
  - 算法复杂性分析

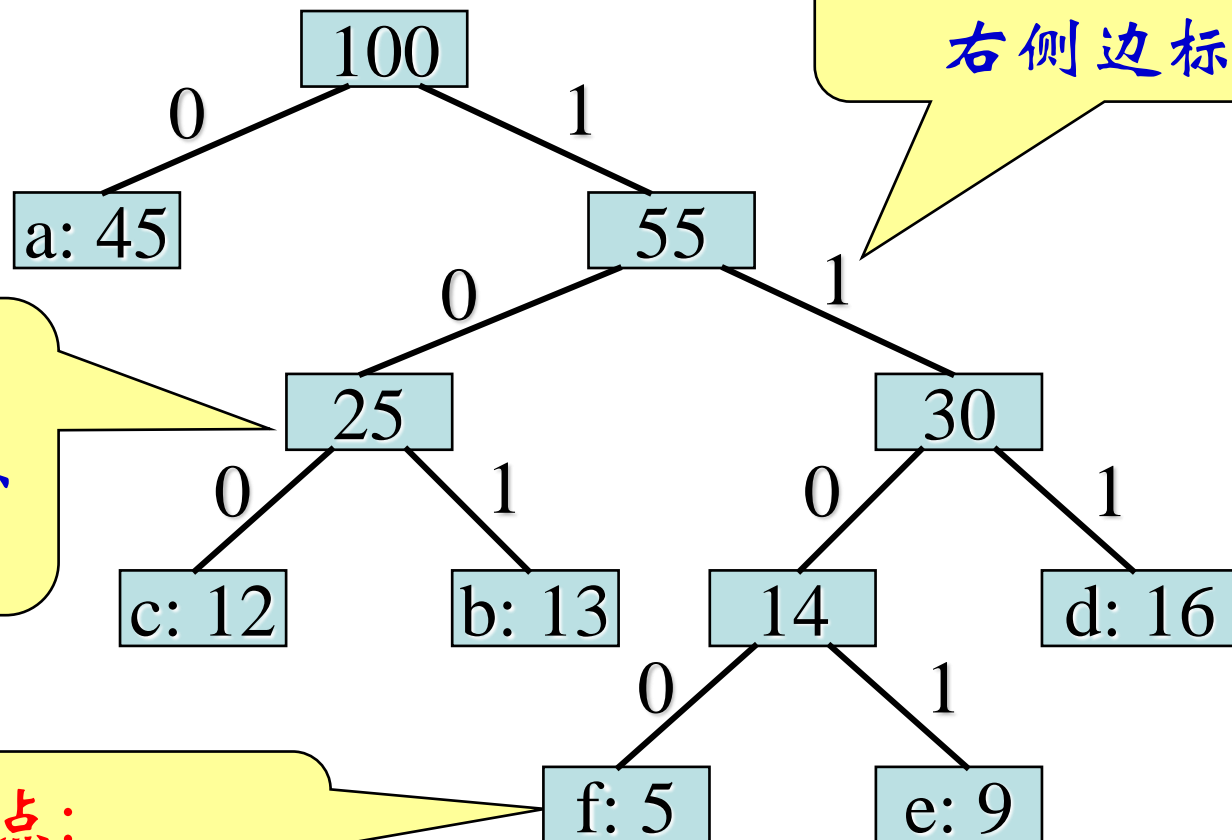


- 二进制字符编码
  - 每个字符用一个二进制0、1串来表示.
- 固定长编码
  - 每个字符都用相同长度的0、1串表示.
- 可变长编码
  - 经常出现的字符用短码，不经常出现的用长码
- 前缀编码
  - 无任何字符的编码是另一个字符编码的前缀



# 一颗编码树对应一个编码方案

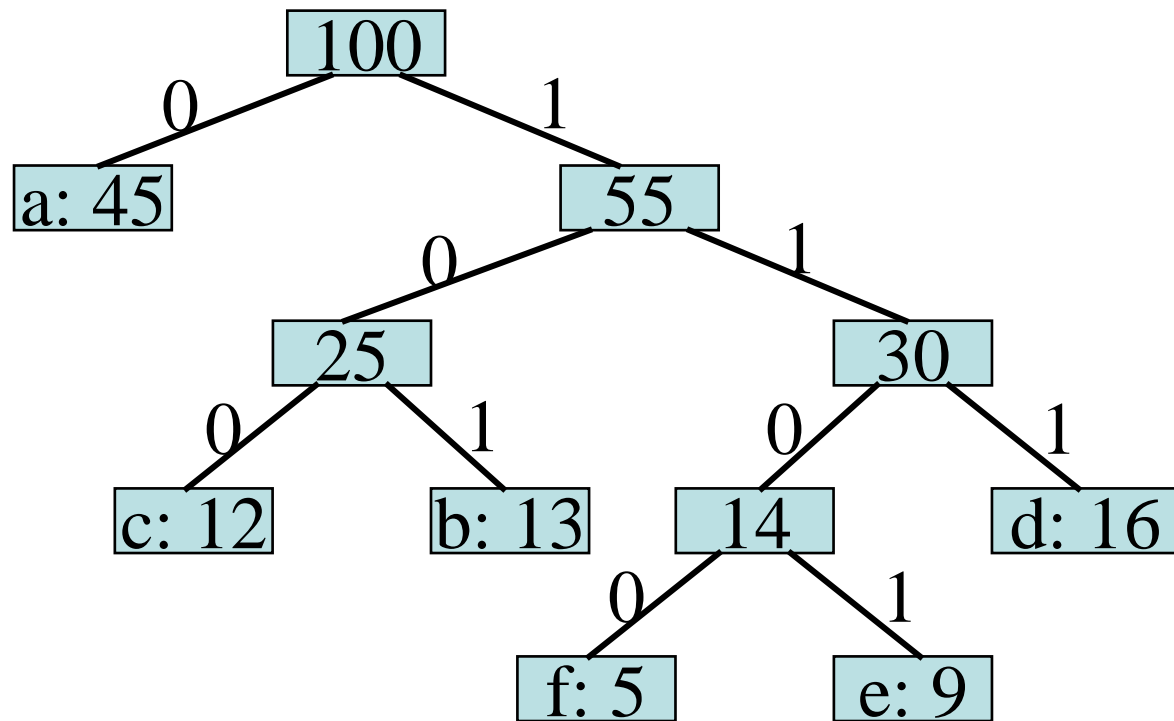
## • 编码树



边标记: 左边标记0,  
右侧边标记1

内结点:  
子树的叶结点的  
频数和

叶结点:  
字符及其出现的频数



- 编码树 $T$ 的代价

- 设 $C$ 是字母表(给定文件中的字母集合),  $\forall c \in C$
- $f(c)$ 是 $c$ 在文件中出现的频数
- $d_T(c)$ 是叶子 $c$ 在树 $T$ 中的深度, 即 $c$ 的编码长度
- $T$ 的代价是编码一个文件的所有字符的代码长度(位数):

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





- 优化编码树问题

输入: 字母表  $C = \{c_1, c_2, \dots, c_n\}$ ,

频数表  $F = \{f(c_1), f(c_2), \dots, f(c_n)\}$

输出: 具有最小  $B(T)$  的  $C$  的前缀编码树

贪心思想:

循环地选择具有最低频数的两个结点,  
生成一棵子树, 直至形成树

剩余子问题: ???



HITWH  
SE

贪心思想:

循环地选择具有最低频数的两个结点,  
生成一棵子树, 直至形成树

f: 5

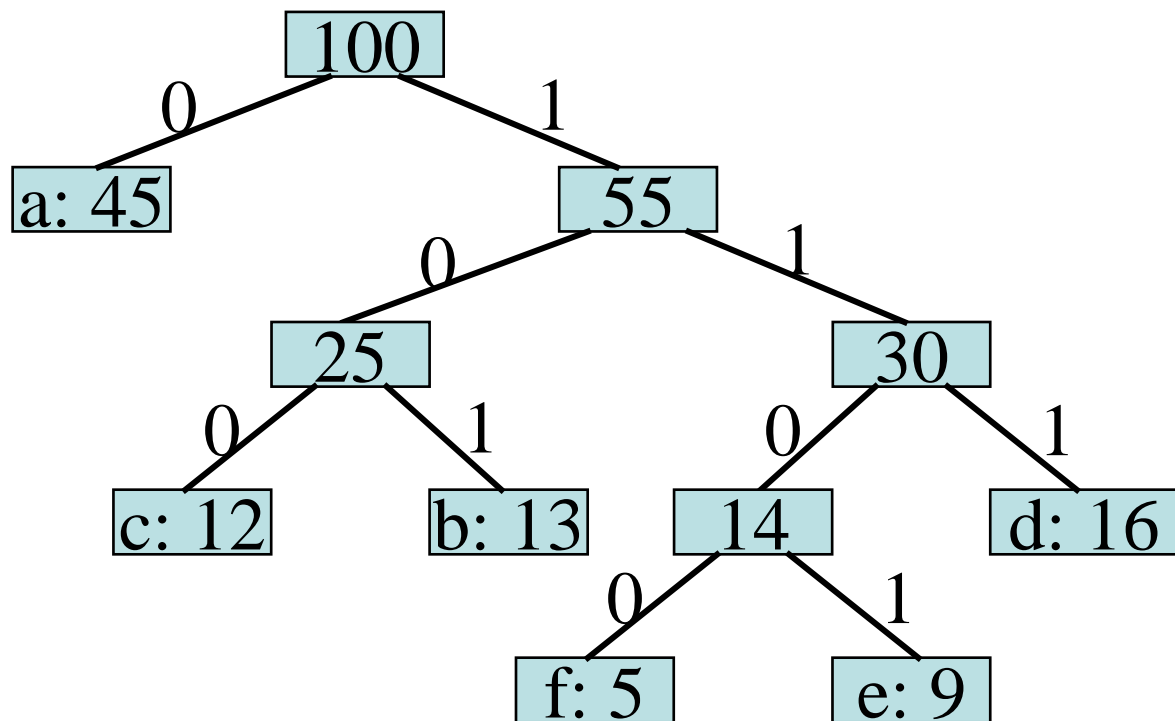
e: 9

c: 12

b: 13

d: 16

a: 45

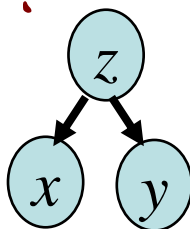




- 贪心选择方法

- 选择方法:

- 每次选择具有最低频数的两个节点 $x$ 和 $y$ ,  
构造一个子树:

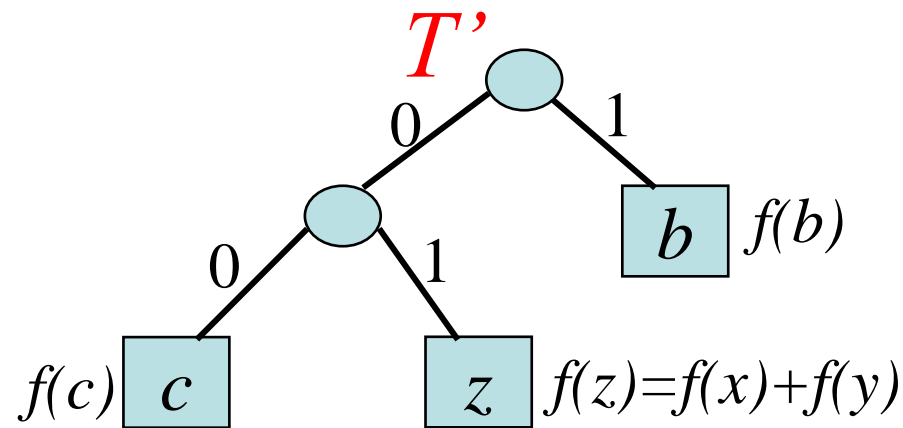
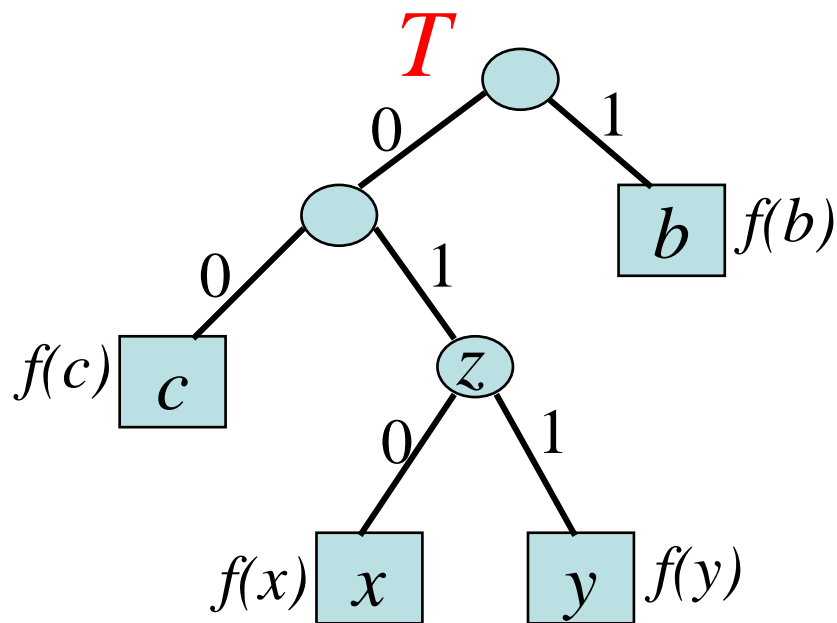


- 剩余子问题的结构:

- $C' = C - \{x, y\} \cup \{z\}$
    - $F' = F - \{f(x), f(y)\} \cup \{f(z)\}$ ,  $f(z) = f(x) + f(y)$

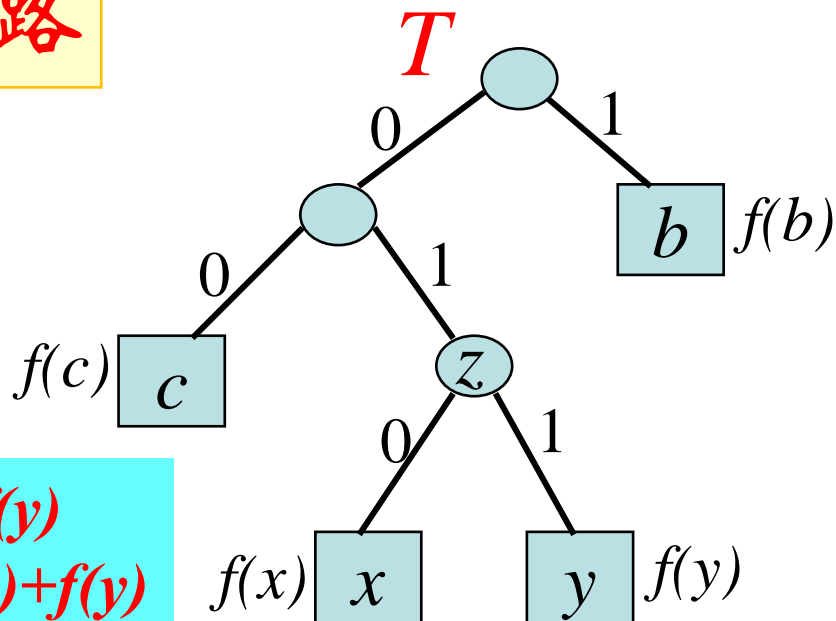
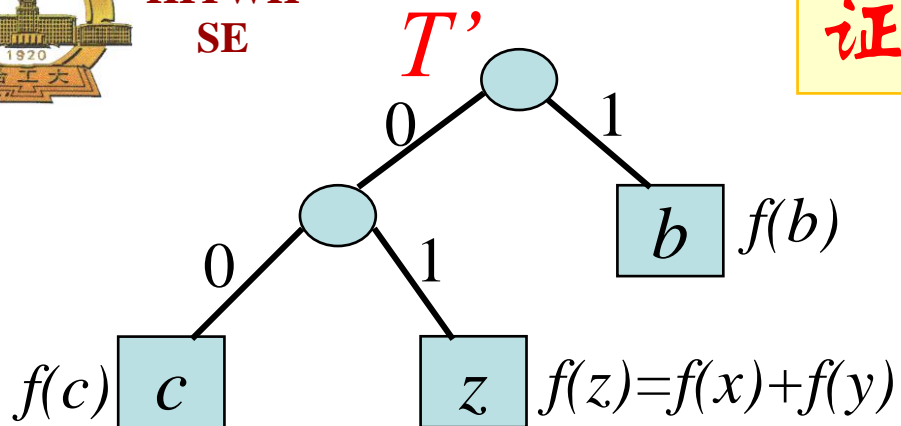


**引理1.** 设 $T$ 是字母表 $C$ 的优化前缀树,  $\forall c \in C$ ,  $f(c)$ 是 $c$ 在文件中出现的频数. 设 $x$ 、 $y$ 是 $T$ 中任意两个相邻叶结点,  $z$ 是它们的父结点, 则 $z$ 作为频数是 $f(z)=f(x)+f(y)$ 的字符,  $T'=T-\{x,y\}$ 是字母表 $C'=C-\{x,y\} \cup \{z\}$ 的优化前缀编码树.

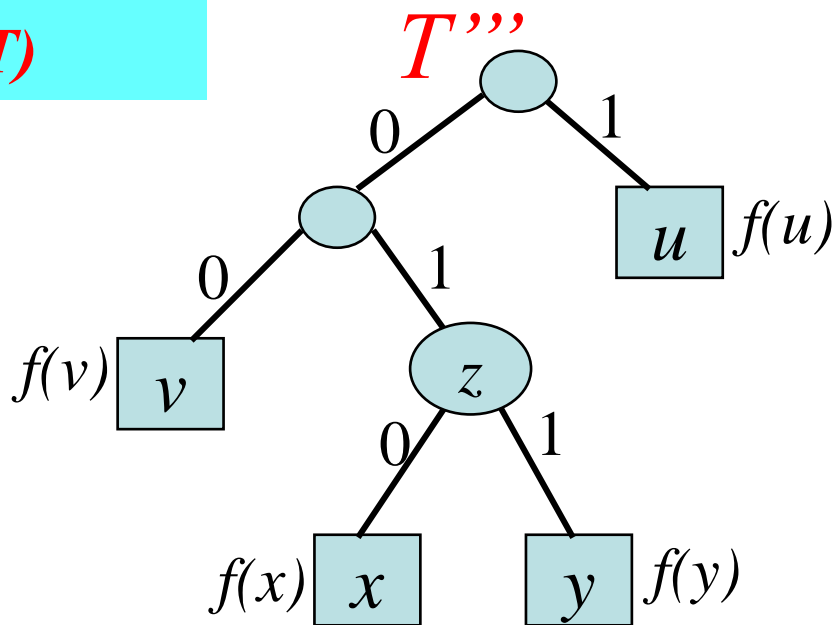
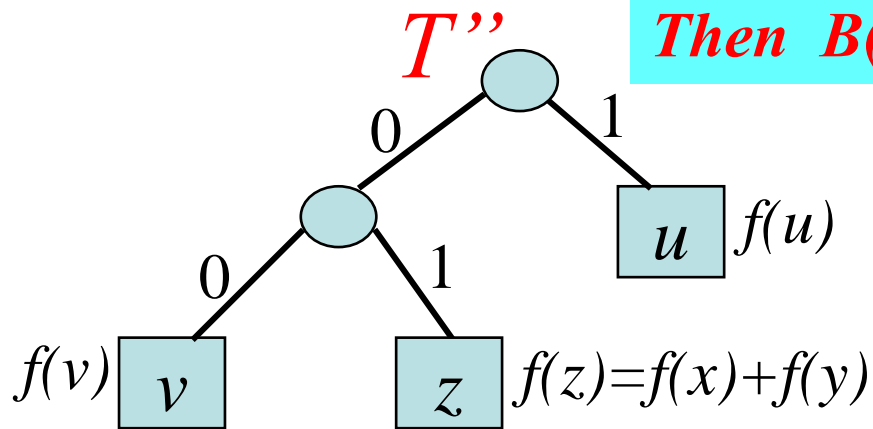




## 证明思路



$B(T) = B(T') + f(x) + f(y)$   
 $B(T'') = B(T'') + f(x) + f(y)$   
If  $B(T'') < B(T')$   
Then  $B(T''') < B(T)$



H

证. 往证  $B(T) = B(T') + f(x) + f(y)$ .

$$B(T) = f(x)d_T(x) + f(y)d_T(y) + \sum_{k \in C - \{x, y\}} f(k)d_T(k)$$

$$B(T') = f(z)d_{T'}(z) + \sum_{k \in C' - \{z\}} f(k)d_{T'}(k)$$

$$B(T) - B(T') = (f(x)d_T(x) + f(y)d_T(y)) - f(z)d_{T'}(z)$$

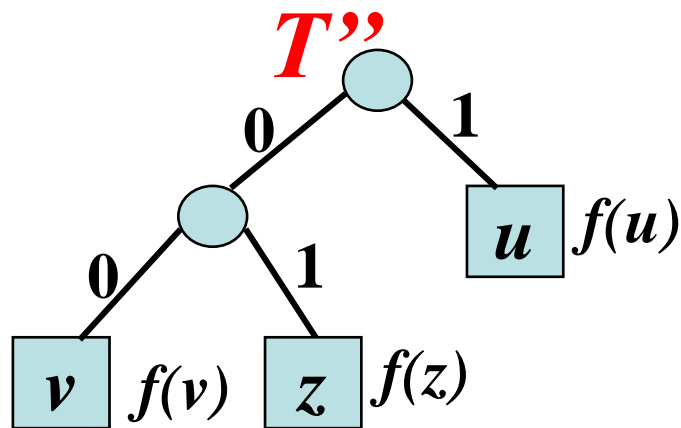
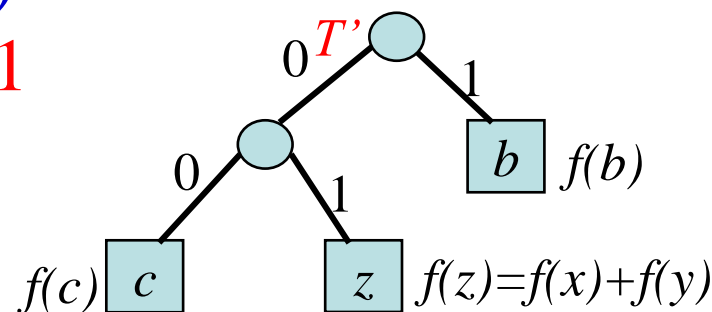
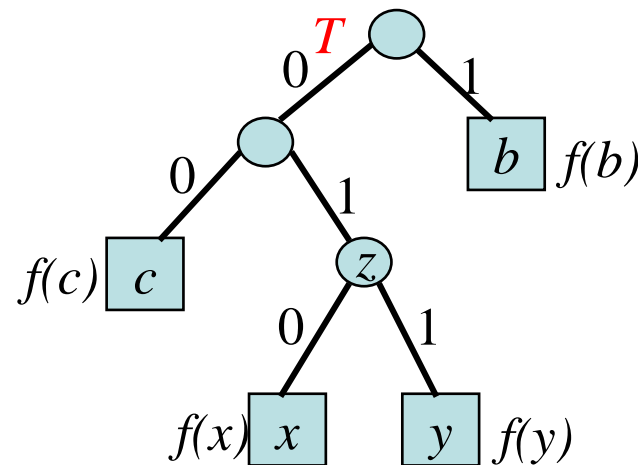
由于  $f(z) = f(x) + f(y)$ ,  $d_T(x) = d_T(y) = d_{T'}(z) + 1$

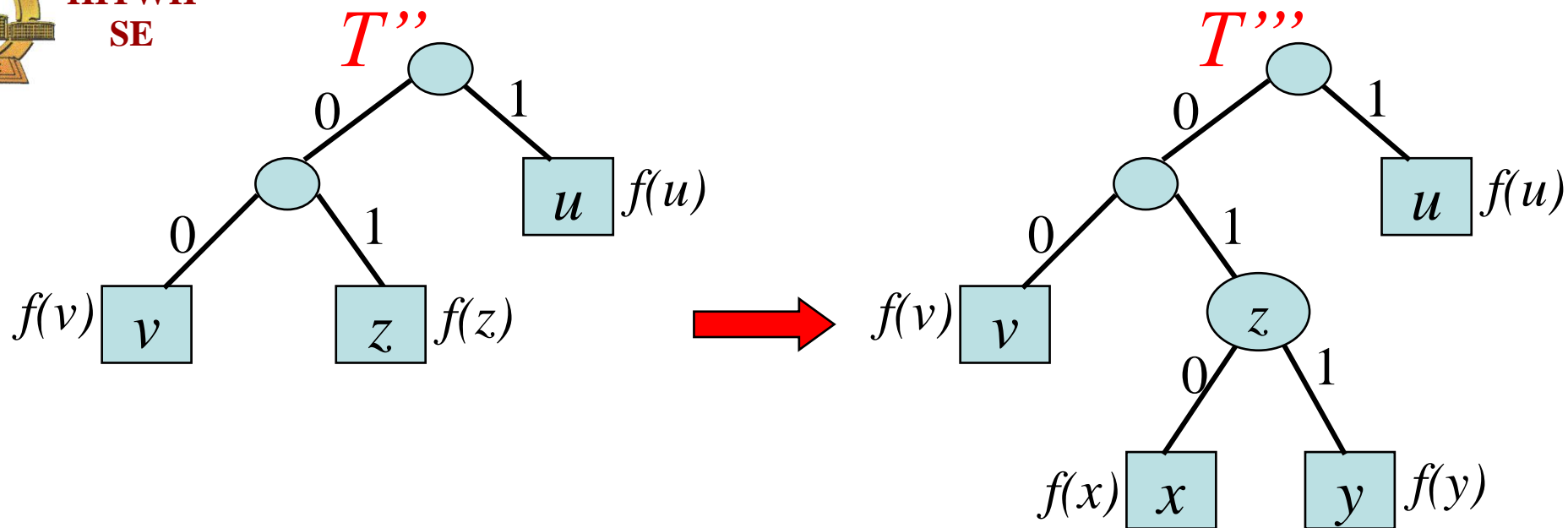
$$\begin{aligned} B(T) - B(T') &= (f(x) + f(y))(d_{T'}(z) + 1) \\ &\quad - (f(x) + f(y))d_{T'}(z) \\ &= f(x) + f(y). \end{aligned}$$

若  $T'$  不是  $C'$  的优化前缀编码树,

则必存在  $T''$ , 使  $B(T'') < B(T')$ .

因为  $z$  是  $C'$  中字符, 它必为  $T''$  中的叶子.  
把结点  $x$  与  $y$  加入  $T''$ , 作为  $z$  的子结点,  
则得到  $C$  的一个如下前缀编码树  $T'''$ :





如上可证：

$$B(T''') = B(T'') + f(x) + f(y)。$$

由于  $B(T'') < B(T')$ ,

$$B(T''') = B(T'') + f(x) + f(y) < B(T') + f(x) + f(y) = B(T)$$

与  $T$  是优化的矛盾，故  $T'$  是  $C'$  的优化编码树。



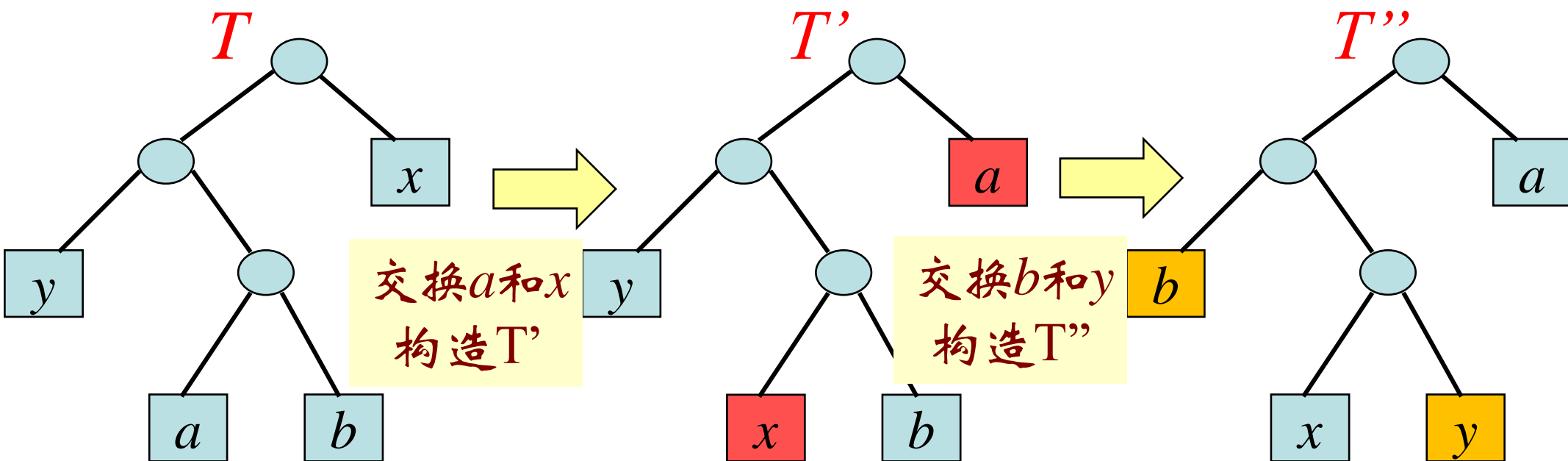
**引理2.** 设  $C$  是字母表,  $\forall c \in C$ ,  $c$  具有频数  $f(c)$ ,  $x, y$  是  $C$  中具有最小频数的两个字符, 则存在一个  $C$  的优化前缀树,  $x$  与  $y$  的编码具有相同最大长度, 且仅在最末一位不同.

优化前缀树问题具有 Greedy 选择性.



**证：**若 $T$ 是 $C$ 的优化前缀树，如果 $x$ 和 $y$ 是具有最大深度的两个兄弟字符，则命题得证。

**若不然，**设 $a$ 和 $b$ 是具有最大深度的两个兄弟字符：



不失一般性，设 $f(a) \leq f(b)$ ,  $f(x) \leq f(y)$ .

因 $x$ 与 $y$ 是具有最低频数的字符， $f(x) \leq f(y) \leq f(a) \leq f(b)$ .

交换 $T$ 的 $a$ 和 $x$ ，从 $T$ 构造 $T'$ ；

交换 $T'$ 的 $b$ 和 $y$ ，从 $T'$ 构造 $T''$

往证 $T''$ 是最优化前缀树.

$$B(T)-B(T')$$

$$= \sum_{c \in C} f(c)d_T(c) - \sum_{c \in C} f(c)d_{T'}(c)$$

$$= \underline{f(x)d_T(x) + f(a)d_T(a)} - \underline{f(x)d_{T'}(x) + f(a)d_{T'}(a)}$$

$$= f(x)d_T(x) + f(a)d_T(a) - f(x)d_{T'}(a) - f(a)d_{T'}(x)$$

$$= (f(a)-f(x))(d_T(a)-d_T(x)).$$

$\because f(a) \geq f(x), d_T(a) \geq d_T(x)$  (因为 $a$ 的深度最大)

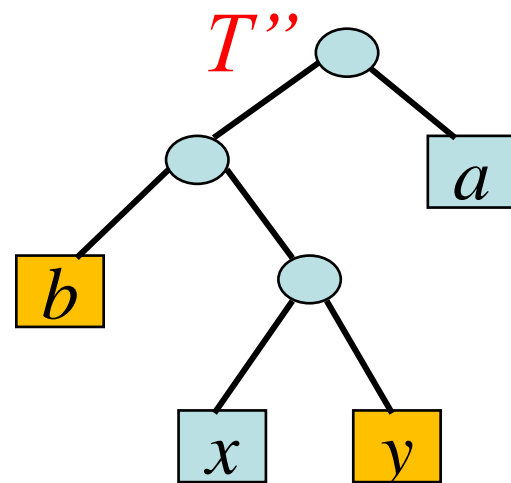
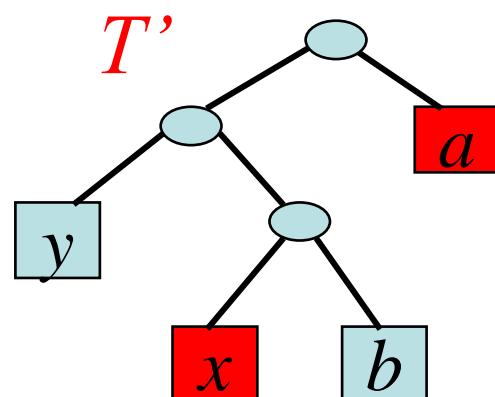
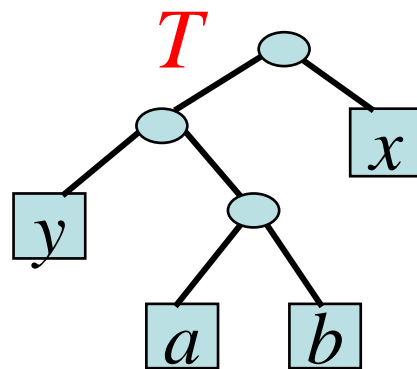
$$\therefore B(T)-B(T') \geq 0, B(T) \geq B(T')$$

同理可证 $B(T') \geq B(T'')$ . 于是 $B(T) \geq B(T'')$ .

由于 $T$ 是最优化的, 所以 $B(T) \leq B(T'')$ .

于是,  $B(T) = B(T'')$ ,  $T''$ 是 $C$ 的最优化前缀编码树.

在 $T''$ 中,  $x$ 和 $y$ 具有相同最大长度编码, 且仅最后位不同.





- 基本思想

- 循环地选择具有最低频数的两个结点，生成一棵子树，直至形成树

# 算法及复杂性分析

- Greedy 算法 ( $Q$  是 min-heap)

Huffman( $C, F$ )

1.  $n \leftarrow |C|$ ;  $Q \leftarrow$  根据  $F$  排序  $C$ ;  $T$  为一个空树; 第1步: 建堆  $O(n)$
  2. FOR  $i \leftarrow 1$  To  $n-1$  Do
  3.      $z \leftarrow \text{Allocate-Node}()$ ;
  4.      $\text{left}[z] \leftarrow x \leftarrow \text{Extract-min}(Q)$  /\* 从  $Q$  删除  $x$  \*/;      $O(\log n)$
  5.      $\text{right}[z] \leftarrow y \leftarrow \text{Extract-min}(Q)$  /\* 从  $Q$  删除  $y$  \*/;      $O(\log n)$
  6.      $f(z) \leftarrow f(x) + f(y)$ ;
  7.      $\text{Insert}(Q, z, f(z))$ ;      $O(\log n)$
  8. Return  $\text{Extract-min}(Q)$  /\* 返回树的根 \*/
- 循环  $n-1$  次, 故:  $T(n) = O(n \log n)$



## 关于算法行为

引理2. 设 $C$ 是字母表,  $\forall c \in C$ ,  $c$ 具有频数 $f(c)$ ,  $x, y$ 是 $C$ 中具有最小频数的两个字符, 则存在一个 $C$ 的优化前缀树,  $x$ 与 $y$ 的编码具有相同最大长度, 且仅在最末一位不同.

引理3. 设 $C$ 是字母表, 存在 $x, y$ 是 $C$ 中具有最小频数的两个字符, 则Huffman算法输出的编码树 $T$ 将 $x, y$ 作为相邻叶子结点, 而且有:

$$d_T(x) = d_T(y) = \max_{a \in T} \{d_T(a)\}.$$

Huffman算法中的贪心选择方法能够达到贪心选择性所需的目标!



算法生成的编码树中，存在最大深度的两个兄弟字符具有最小的两个频率。

往证：在合并过程中，存在具有最大深度的两个兄弟字符具有最小的两个频率。

证明要点：

(1) 如果 $r$ 和 $r'$ 是两个内节点，而且 $r$ 先被生成，则 $f(r) \leq f(r')$ ；

(2) 如果 $r$ 的儿子节点为 $x$ 和 $y$ ， $r'$ 的儿子节点是 $x'$ 和 $y'$ ，而且 $f(r) = f(r')$ ，则有 $f(x) = f(y) = f(x') = f(y')$ ，而且 $r$ 和 $r'$ 的后代中同深度节点的频率都相等；

(3) 往证的命题在第一次合并之后成立；

(4) 如果合并过程中，往证命题不再成立，关注固定的一次关键的合并，这次合并节点 $r'$ 和 $r''$ ，这次合并之前，往证命题成立，合并之后， $r'$ 中的叶子具有最大深度，这些叶子中不存在兄弟节点具有最小的频率，继而导致往证命题不成立。

(5) 由于合并 $r'$ 和 $r''$ 之前， $r'$ 被刚刚合成的时候，存在具有最大深度的两个兄弟字符具有最小的两个频率，设这两个节点所在子树的根为 $r$ ， $r$ 比 $r'$ 先被合成， $f(r) \leq f(r')$ ， $r$ 的高度不小于 $r'$ 的高度。又由于往证命题在合并 $r'$ 和 $r''$ 之后不成立， $r'$ 的高度加1超过 $r$ 的高度，可证 $r$ 和 $r'$ 具有相同的高度，而且该次合成使用 $r'$ 而未使用 $r$ ，因此 $f(r') \leq f(r)$ ，因此 $f(r) = f(r')$ 。

(6) 继而可证 $r$ 和 $r'$ 中最深的叶子也具有相同的频率，与 $r'$ 中没有兄弟叶子具有最小频率产生矛盾。



**定理3.** Huffman算法产生一个优化前缀编码树

**证.** 由于引理1（优化子结构）、引理2成立，

根据引理3，Huffman算法按照引理2的Greedy选择性确定的规则进行局部优化选择，所以Huffman算法产生一个优化前缀编码树。



## 5.4 Minimal spanning tree problem

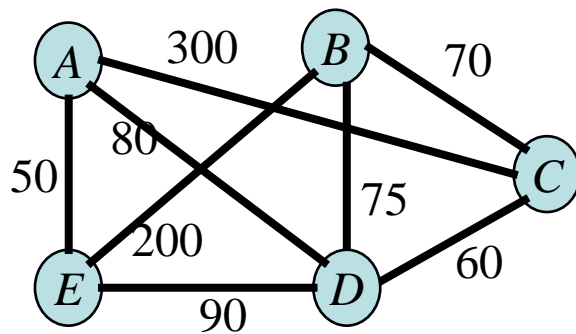
- 问题定义
- Kruskal 算法
- Prim 算法
- Generic 算法





## • 生成树

- 设  $G=(V, E)$  是一个边加权无向连通图.  $G$  的生成树是无向树  $T=(V, E_T)$ ,  $E_T \subseteq E$ .
- 如果  $W: E \rightarrow \{\text{实数}\}$  是  $G$  的权函数,  $T$  的权值定义为  $W(T) = \sum_{(u,v) \in T} W(u,v)$ .



## • 最小生成树

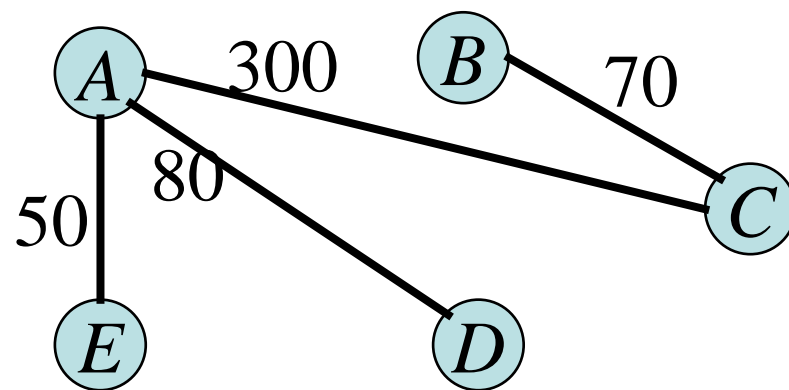
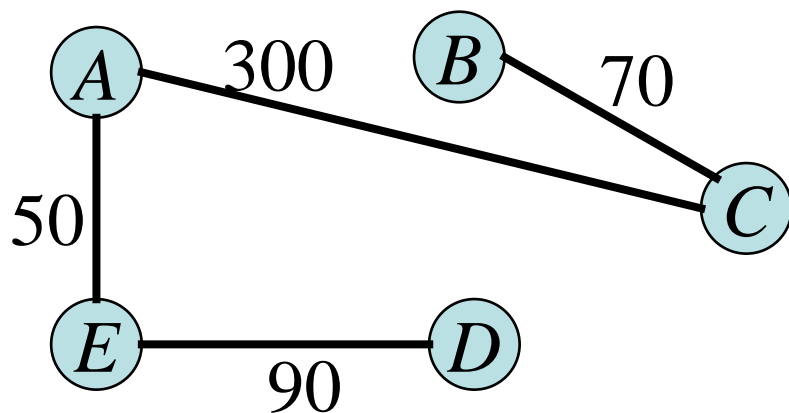
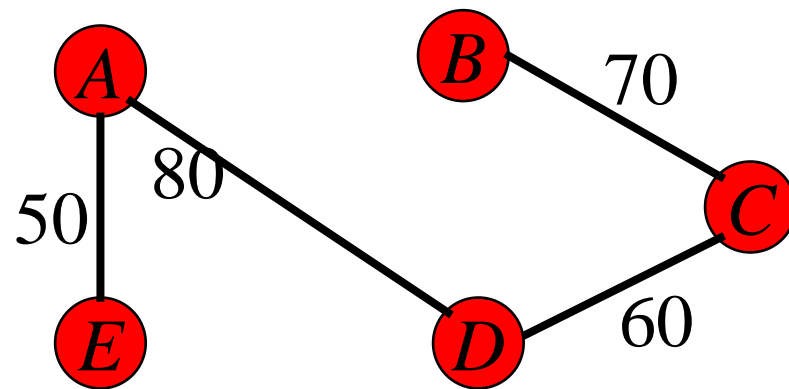
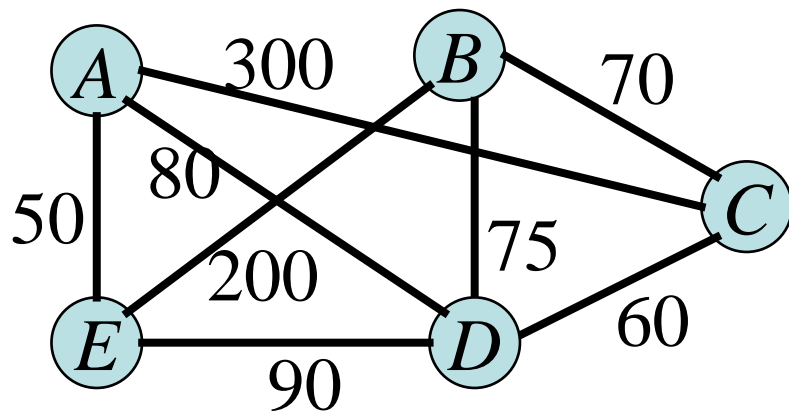
- $G$  的最小生成树是  $W(T)$  最小的  $G$  之生成树.

## • 问题的定义

**输入:** 无向连通图  $G=(V, E)$ , 权函数  $W$

**输出:**  $G$  的最小生成树

# • 实例

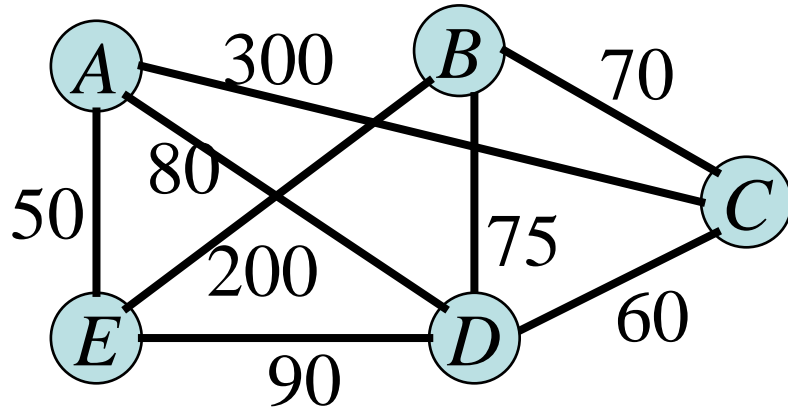




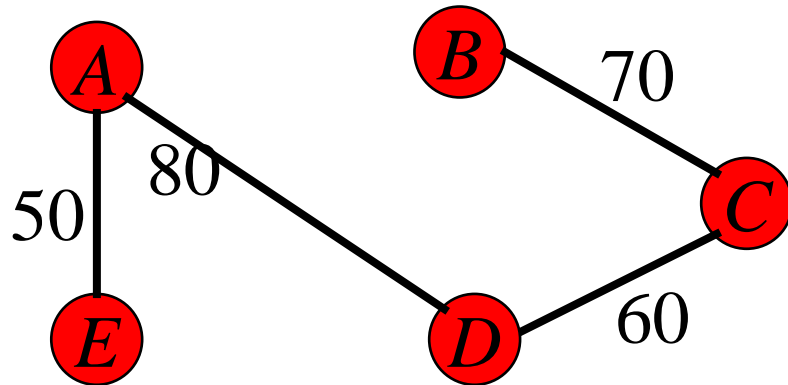
- **Kruskal算法**
  - 设计贪心选择方法
  - 优化解结构分析
  - Greedy选择性证明
  - 算法复杂性

# 设计贪心选择方法

## • 基本思想



- 初始:  $A = \text{空}$ ; 构造森林  $G_A = (V, A)$ ;



剩余子问题?

- 贪心策略: 选择连接  $G_A$  中两棵树的具有最小权值的边加入  $A$ .



# Greedy选择性

**定理1.** 设 $(u,v)$ 是 $G$ 中权值最小的边，则必有一棵最小生成树包含边 $(u,v)$ .

证明：设 $T$ 是 $G$ 的一棵MST

若 $(u,v) \in T$ , 结论成立；

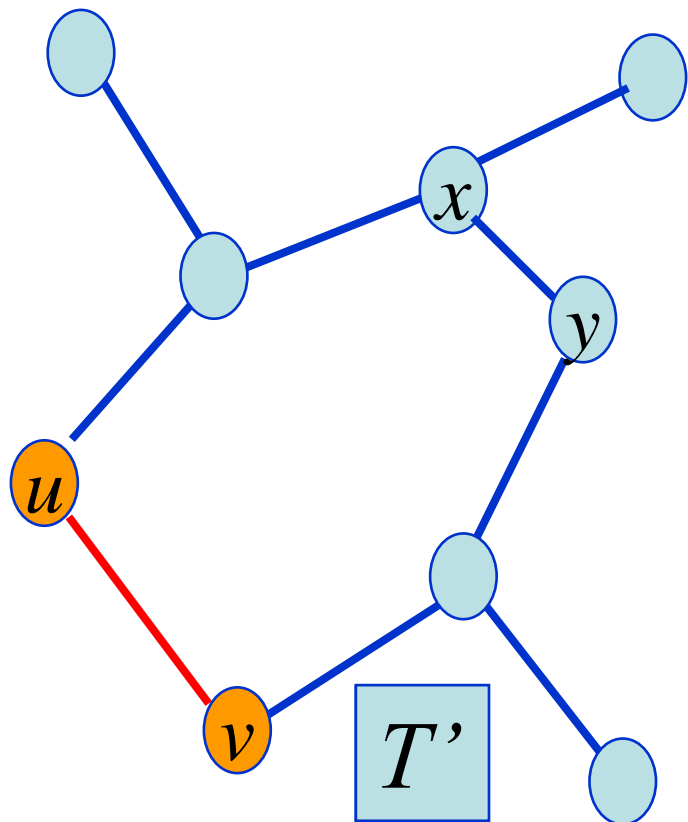
否则，如左图所示

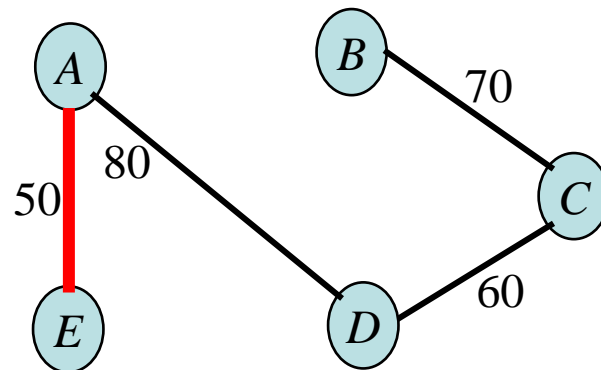
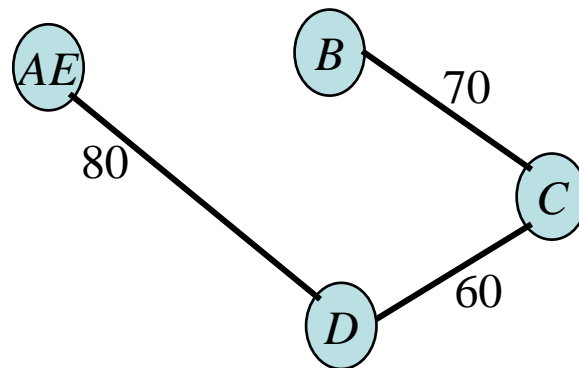
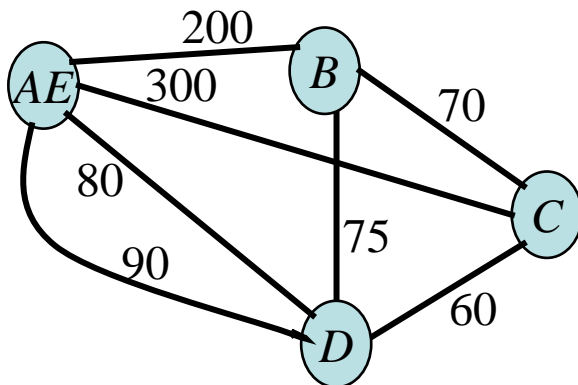
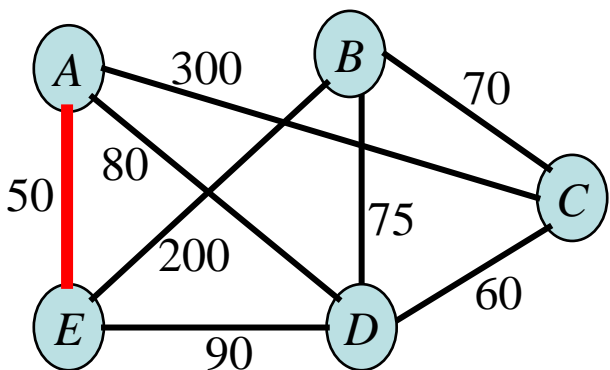
在 $T$ 中添加 $(u,v)$ 边，必产生环  
删除环中不同于 $(u,v)$ 的权值最小的边，设为 $(x,y)$ ，得到 $T'$ .

$$w(T') = w(T) - w(x,y) + w(u,v) \leq w(T)$$

又 $T$ 是最小生成树， $w(T) \leq w(T')$

则 $T'$ 也是一棵MST, 且包含边 $(u,v)$ .





- 图 $G$ 的边 $(x, y)$ 的收缩:  $G/(x, y)$ 
  - 用新顶点 $z$ 代替边 $(x, y)$
  - $\forall v \in V$ , 用边 $(z, v)$ 代替边 $(x, v)$ 或 $(y, v)$
  - 删除 $z$ 到其自身的边
  - $G$ 的其余部分保持不变
- 收缩操作的逆操作称为**扩张**, 表示为 $G/_z^{(x, y)}$



**定理2.** 给定加权无向连通图  $G=(V,E)$ , 权值函数为  $W:E \rightarrow R$ ,  $(u,v) \in E$  是  $G$  中权值最小的边。设  $T$  是  $G$  的包含  $(u,v)$  的一棵最小生成树, 则  $T/(u,v)$  是  $G/(u,v)$  的一棵最小生成树。

**证明.**

由于  $T/(u,v)$  是不含回路的连通图且包含了  $G/(u,v)$  的所有顶点, 因此,  $T/(u,v)$  是  $G/(u,v)$  的一棵生成树。

下面证明  $T/(u,v)$  是  $G/(u,v)$  的代价最小的生成树。

若不然, 存在  $G/(u,v)$  的生成树  $T'$  使得  $W(T') < W(T/(u,v))$ 。

显然,  $T'$  中包含顶点  $z=(u,v)$  且是连通的, 因此  $T''=T' \cup z$  包含  $G$  的所有顶点且不含回路, 故  $T''$  是  $G$  的一棵生成树。

但,  $W(T'') = W(T') + W(u,v) < W(T/(u,v)) + W(u,v) = W(T)$ , 这与  $T$  是  $G$  的最小生成树矛盾。



MST-Kruskal( $G(V, E), W$ )

1.  $A = \Phi$ ;
2. For  $\forall v \in V$  Do
3.     Make-Set( $v$ ); /\* 建立只有 $v$ 的集合 \*/
4. 按照 $W$ 值的递增顺序排序 $E$ 中的边;
5. For  $\forall (u, v) \in E$  (按 $W$ 值的递增顺序) Do
6.     If Find-Set( $u$ )  $\neq$  Find-Set( $v$ ) (判断是否出现回路)
7.     Then  $A = A \cup \{(u, v)\}$ ; Union( $u, v$ ); (合并 $u, v$ 集合)
8. Return  $A$





MST-Kruskal( $G(V, E)$ ,  $W$ )

1.  $A = \Phi$ ;
2. For  $\forall v \in V$  Do
3.     Make-Set( $v$ ); /\* 建立只有 $v$ 的集合 \*/
4.     按照 $W$ 值的递增顺序排序 $E$ ;
5.     For  $\forall (u, v) \in E$  (按 $W$ 值的递增顺序) Do
6.         If Find-Set( $u$ )  $\neq$  Find-Set( $v$ )
7.         Then  $A = A \cup \{(u, v)\}$ ;  
              Union( $u, v$ );
8.     Return  $A$

- 令  $n = |V|$ ,  $m = |E|$
  - 第2-3步执行  $O(n)$  个 Make-Set 操作
  - 第4步需要时间:  $O(m \log m)$
  - 第5-7步执行  $O(m)$  个 Find-Set 和 Union 操作
- 第2-3步和5-7步需要的时间为:
- $O((n+m)\alpha(n))$
- $m \geq n-1$  (因为  $G$  连通), 由  $\alpha(n) < \log n < \log m$
  - 总时间复杂性:  $O(m \log m)$

集合操作的复杂性见Intro. To Algo. 第21章 (498-509)

后续章节中将介绍这部分内容



**定理2.**  $\text{MST-Kruskal}(G, W)$  算法能够产生图  $G$  的最小生成树.

**证.**

因为算法按照 Greedy 选择性进行局部优化选择, 并且每次选择的都是权值最小的边.



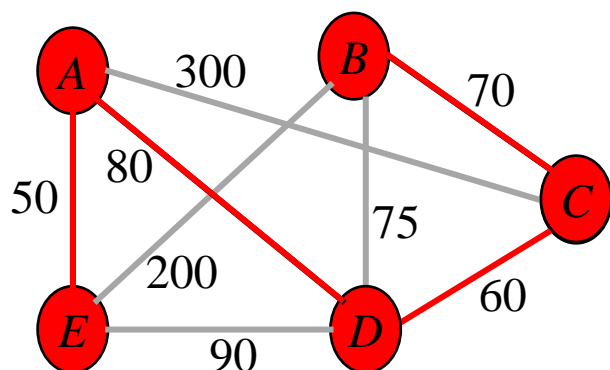
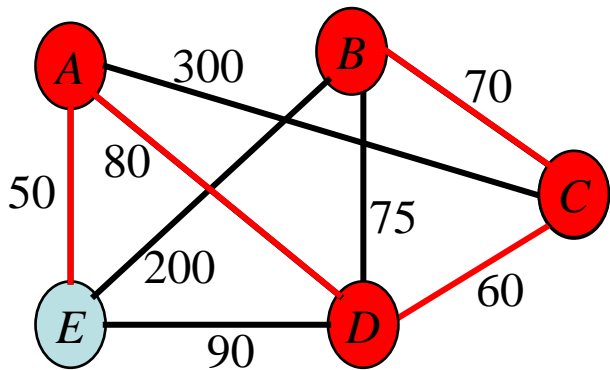
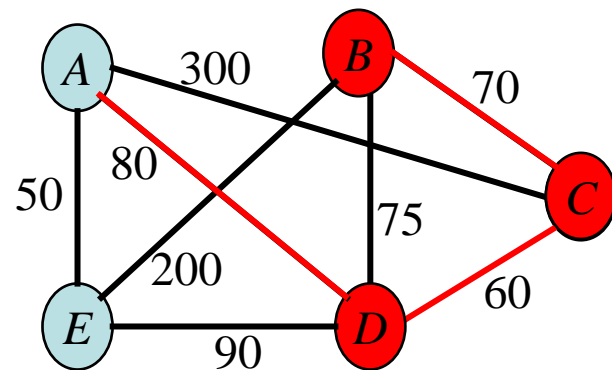
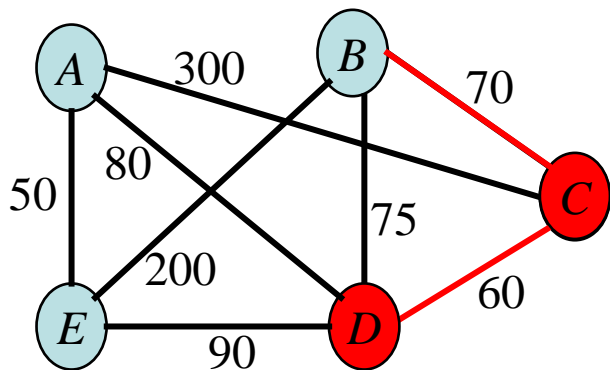
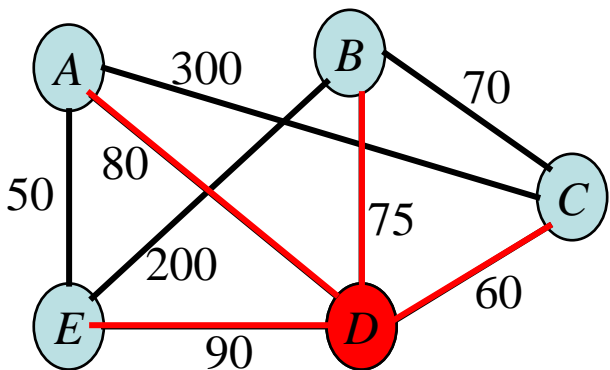
- Prim算法

- 设计贪心选择方法
- 优化解结构分析
- Greedy选择性证明
- 算法复杂性



## • 贪心策略

- 以任意顶点  $v_r$  作为树根，初始  $C = \{v_r\}$
- 选择  $C$  和  $V - C$  之间权值最小的边  $(x, y)$ ,  $x \in C$ ,  $y \in V - C$
- $C = C \cup \{y\}$





**定理1.** 设 $(u,v)$ 是 $G$ 中与节点 $u$ 相关联的权值最小的边，则必有一棵最小生成树包含边 $(u,v)$ .

证明：设 $T$ 是 $G$ 的一棵MST

若 $(u,v) \in T$ , 结论成立；

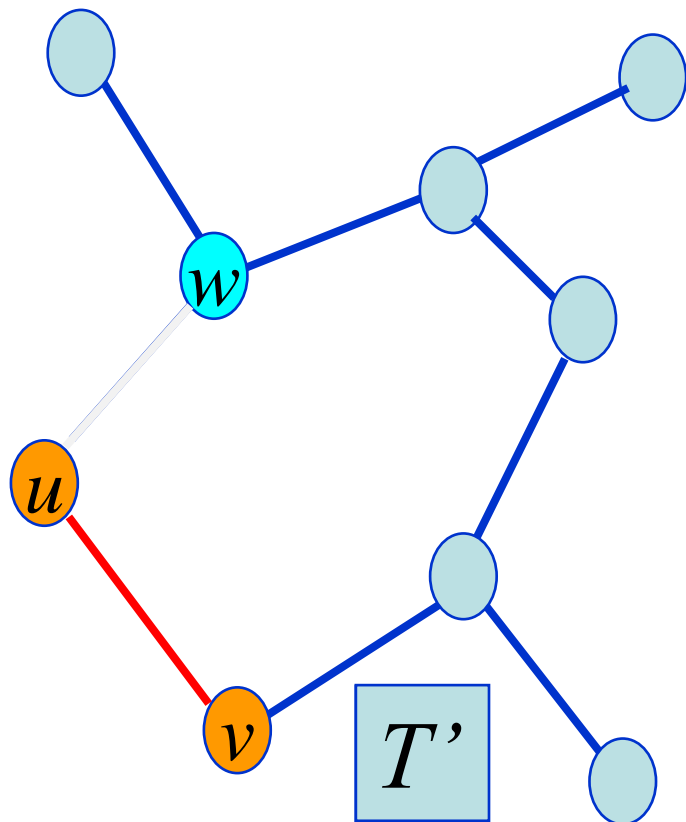
否则，如左图所示

在 $T$ 中添加 $(u,v)$ 边，必产生环  
删除环中与 $u$ 相关联的边 $(u,w)$ ，  
得到 $T'$ .

$$w(T') = w(T) - w(u,w) + w(u,v) \leq w(T)$$

又 $T$ 是最小生成树， $w(T) \leq w(T')$

则 $T'$ 也是一棵MST, 且包含边 $(u,v)$ .





**定理2.** 给定加权无向连通图  $G=(V,E)$ , 权值函数为  $W:E \rightarrow R$ ,  $(u,v) \in E$  是  $G$  中与节点  $u$  相关联的权值最小的边。设  $T$  是  $G$  的包含  $(u,v)$  的一棵最小生成树, 则  $T/(u,v)$  是  $G/(u,v)$  的一棵最小生成树。

证明. 略

与 Kruskal 算法类似



MST-Prim( $G, W, r$ )

Input 连通图 $G$ , 权值函数 $W$ , 树根 $r$

Output  $G$ 的一棵以 $r$ 为根的生成树

1. For  $\forall v \in V[G]$  Do
2.      $\text{key}[v] \leftarrow +\infty$      //所有与 $v$ 邻接的边的最小权值
3.      $\pi[v] \leftarrow \text{null}$      //与 $v$ 邻接的具有最小权值的边
4.  $\text{key}[r] \leftarrow 0$
5.  $Q \leftarrow V[G]$      // $Q$ 是一个最小堆
6. While  $Q \neq \emptyset$  do
7.      $u \leftarrow \text{Extract\_Min}(Q)$
8.     for  $\forall v \in \text{Adj}[u]$  do
9.         if  $v \in Q$  且  $w(u, v) < \text{key}[v]$  then
10.              $\pi[v] \leftarrow u$
11.              $\text{key}[v] \leftarrow w(u, v)$      //更新信息
12. Return  $A = \{(v, \pi[v]) \mid v \in V[G] - r\}$

Line 1-5:  $O(V)$

Line 6循环  $O(V)$  次

Line 7: 每次  $O(\log V)$

Line 8循环  $O(E)$  次

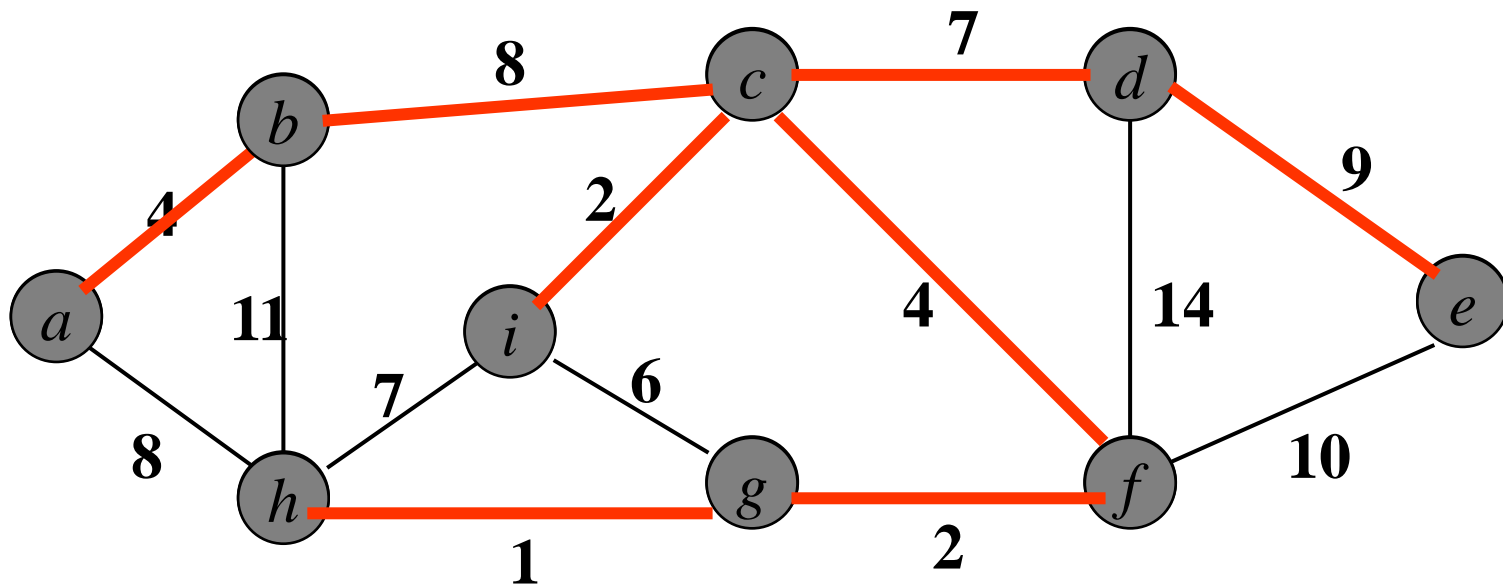
Line 10: 每次常数时间

Line 11: 每次  $O(\log V)$

共计:  $O(E \log V)$



# Prim算法实例



	$\pi[v]$	key[v]
<i>a</i>	null	0
<i>b</i>	<i>a</i>	4
<i>c</i>	<i>b</i>	8
<i>d</i>	<i>c</i>	7
<i>e</i>	<i>d</i>	9
<i>f</i>	<i>c</i>	4
<i>g</i>	<i>f</i>	2
<i>h</i>	<i>g</i>	1
<i>i</i>	<i>c</i>	2





- Generic 算法
  - 基本思想
  - 算法描述
  - 贪心选择性
  - 回顾 Kruskal 和 Prim 算法



- 扩展出一颗最小生成树

- A从空集开始，每次向A中添加一条边
- 添加 $|V|-1$ 条边后，算法终止，A为最小生成树的边集
- 在添加边的过程中，维护如下循环不变量

每一轮添加边之前，A是某个最小生成树的子集

- 每一轮迭代中，选择一条边 $(u, v)$ ， $A = A \cup \{(u, v)\}$

安全边：如果边集合A包含于某个最小生成树，而且 $A \cup \{(u, v)\}$ 也包含于某个最小生成树，则称边 $(u, v)$ 对A是安全的，即 $(u, v)$ 是A的安全边。



Generic-MST( $G(V, E), W$ )

1.  $A = \Phi$ ;
2. While A 没有形成  $G$  的生成树
3. 选择一条  $A$  的安全边  $(u, v)$ ;
4.  $A = A \cup \{(u, v)\}$ ;
5. Return  $A$ ;

**循环不变量:** 每一轮添加边之前,  $A$  是某个最小生成树的子集.

**初始化:** 执行第1行之后,  $A$  为空集满足循环不变量.

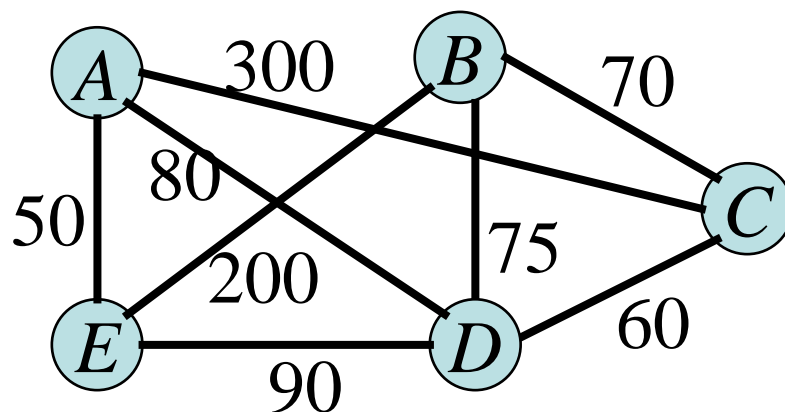
**维护:** 第2-4行添加  $A$  的安全边, 添加之后能维护循环不变量.

**终止:**  $A$  中的边形成  $G$  的生成树, 且包含于一个最小生成树, 因此  $A$  即为  $G$  的一个最小生成树.



- Cut  $(S, V-S)$

- $(S, V-S)$  是无向图  $G=(V, E)$  的顶点集合  $V$  的一个划分

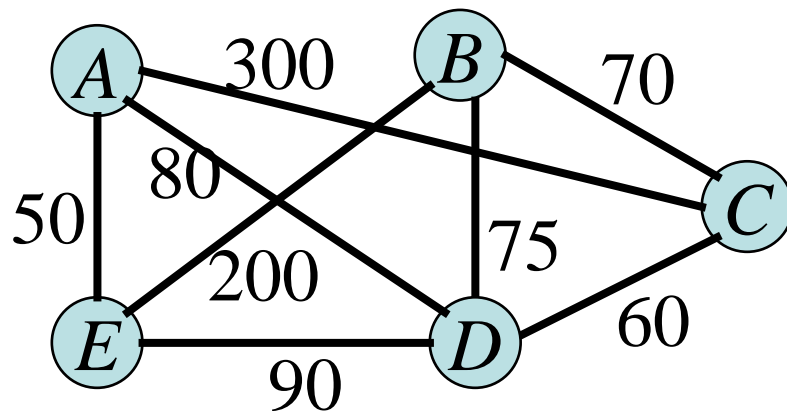


例如  $S=\{A, B, E\}$ ,  $V-S=\{C, D\}$ ,  $(S, V-S)$  是一个 Cut.



- 跨Cut  $(S, V-S)$  的边

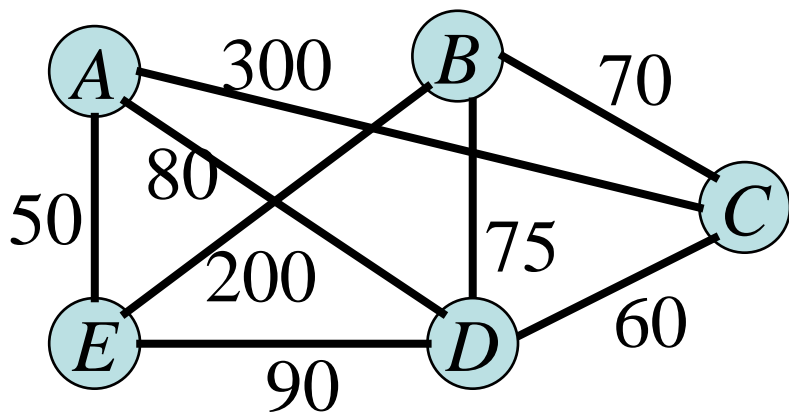
-  $(S, V-S)$  是无向图  $G=(V, E)$  的顶点集合  $V$  的一个划分, 如果边  $(u, v)$  的顶点满足  $u \in S, v \in V - S$ , 则称边  $(u, v)$  跨Cut  $(S, V-S)$ .



例如  $S=\{A, B, E\}$ ,  $V-S=\{C, D\}$ ,  $(S, V-S)$  是一个Cut.  
边  $(A, C)$ ,  $(A, D)$ ,  $(B, C)$ ,  $(B, D)$  和  $(E, D)$  跨  $(S, V-S)$ .



- 遵守边集  $A$  的 Cut  $(S, V-S)$ 
  - 如果边集  $A$  中没有跨  $(S, V-S)$  的边, 则称  $(S, V-S)$  遵守边集  $A$ .
- 跨  $(S, V-S)$  的轻边
  - 如果边  $(u, v)$  在所有跨  $(S, V-S)$  的边中具有最小权值, 则称边  $(u, v)$  是跨  $(S, V-S)$  的轻边.

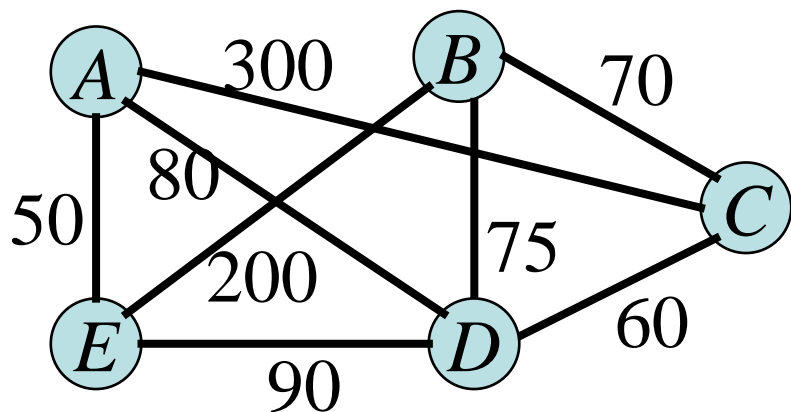


- 例如  $S = \{A, B, E\}$ ,  $V-S = \{C, D\}$ ,  $(S, V-S)$  是一个 Cut.
- 边  $(B, C)$  是跨  $(S, V-S)$  的轻边.
- 跨同一个 Cut 的轻边可能不唯一.



## • 定理1

设  $G=(V, E)$  是一个边加权无向连通图.  $W: E \rightarrow \{\text{实数}\}$  是  $G$  中边的权函数. 令  $A$  是  $E$  的子集并包含于  $G$  的某个最小生成树中,  $(S, V-S)$  为  $G$  中任意遵守  $A$  的 Cut. 如果边  $(u, v)$  是跨  $(S, V-S)$  的轻边, 则  $(u, v)$  必为  $A$  的安全边.



- 例如  $S=\{A, B, E\}, V-S=\{C, D\}$
- $(S, V-S)$  是一个 Cut.
- $A=\{(A, E), (C, D)\}$
- $(S, V-S)$  遵守  $A$
- 边  $(B, C)$  是跨  $(S, V-S)$  的轻边.
- 边  $(B, C)$  是  $A$  的安全边.



# 回顾Kruskal算法

## • 扩展出一颗最小生成树

-  $A$  为已选择的边集

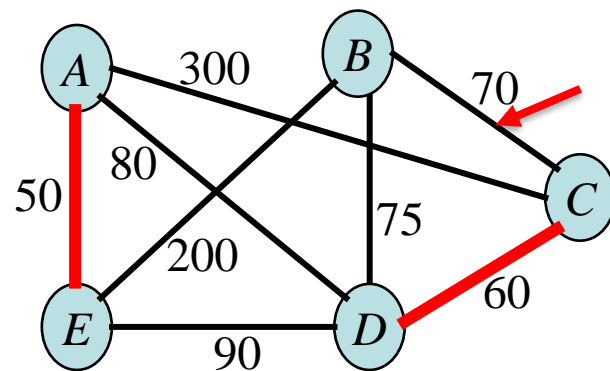
- 安全边的选择:

- $A$  中包含一些列连通分量
- Kruskal算法选择连通分量之间权值最小的边  $(u, v)$
- $(u, v)$  连接的连通分量为  $(V_1, E_1)$  和  $(V_2, E_2)$
- $\text{Cut}(V_1, V - V_1)$  遵守  $A$
- $(u, v)$  是跨  $\text{Cut}(V_1, V - V_1)$  的轻边
- $(u, v)$  是  $A$  的安全边

Kruskal算法使用遵守  $\{(A, E), (C, D)\}$  的  $\text{Cut}(\{B\}, \{A, C, D, E\})$  寻找  $\{(A, E), (C, D)\}$  的安全边.

Generic-MST( $G(V, E), W$ )

1.  $A = \Phi$ ;
2. While  $A$  没有形成  $G$  的生成树
3.     选择一条  $A$  的安全边  $(u, v)$ ;
4.      $A = A \cup \{(u, v)\}$ ;
5. Return  $A$ ;







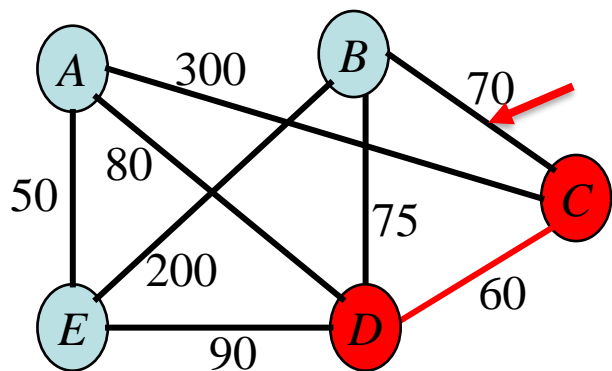
# 回顾Prim算法

## • 扩展出一颗最小生成树

-  $A = \{(v, v.\pi) | v \in V - \{r\} - Q\}$

- 安全边的选择:

## • 使用Cut $(V(A), V - V(A))$



- $A = \{(C, D)\}$
- 使用  $(\{C, D\}, \{A, B, E\})$
- 找到轻边  $(B, C)$

Generic-MST( $G(V, E), W$ )

1.  $A = \Phi$ ;
2. While A 没有形成  $G$  的生成树
3.       选择一条  $A$  的安全边  $(u, v)$ ;
4.        $A = A \cup \{(u, v)\}$ ;
5.   Return A;

MST-Prim( $G, W, r$ )

Input    连通图  $G$ , 权值函数  $W$ , 树根  $r$

Output    $G$  的一棵以  $r$  为根的生成树

1. For  $\forall v \in V[G]$  Do
2.      $\text{key}[v] \leftarrow +\infty$      // 所有与  $v$  邻接的边的最小权值
3.      $\pi[v] \leftarrow \text{null}$      // 与  $v$  邻接的具有最小权值的边
4.  $\text{key}[r] \leftarrow 0$
5.  $Q \leftarrow V[G]$
6. While  $Q \neq \emptyset$  do
7.      $u \leftarrow \text{Extract\_Min}(Q)$
8.     for  $\forall v \in \text{Adj}[u]$  do
9.         if  $v \in Q$  且  $w(u, v) < \text{key}[v]$  then
10.              $\pi[v] \leftarrow u$
11.              $\text{key}[v] \leftarrow w(u, v)$      // 更新信息
12. Return  $A = \{(v, \pi[v]) | v \in V[G] - r\}$



## 5.5 Minimum Makespan Scheduling

- 问题的定义
- 近似算法的设计
- 算法的性能分析



- 输入

计算时间分别为 $t_1, \dots, t_n$ 的 $n$ 个任务;  
 $m$ 台完全一样的机器

- 输出

计算任务在 $m$ 台机器上的一个调度策略  
使并行执行时间最短

**Minimum makespan scheduling on identical machines  
是一个NP完全问题.**



- 基本思想

- 贪心选择: 选择具有最短任务队列的机器

例如:  $m=3$ ,  $t_1 \sim t_6 = 1, 2, 3, 4, 5, 6$  (不限定任务序)

机器1



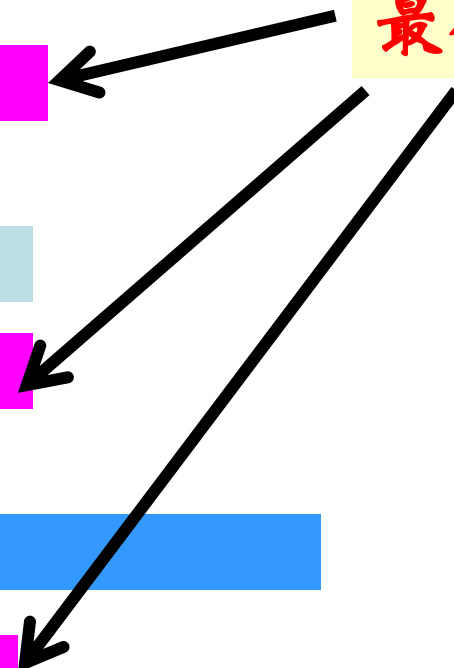
机器2



机器3



最优解



- 算法

## MakeSpanScheduling ()

1. 任意排定所有任务的一个顺序  $t_1, \dots, t_n$
2. For  $k \leftarrow 1$  To  $m$  Do
3.      $T_k \leftarrow 0, M_k \leftarrow \emptyset$
4. For  $i=1$  to  $n$  Do
5.     找出  $j$  使得  $T_j = \min_{1 \leq k \leq m} \{T_k\}$
6.      $T_j \leftarrow T_j + t_i; M_j \leftarrow M_j \cup \{i\}$
7. 输出  $M_1, \dots, M_m$

时间复杂性  $O(nm)$     用最小堆  $O(n \log m)$



**定理:** *MakeSpanScheduling* 算法的近似比为2

**证明:** 令  $T^*$  为最优解代价(并行时间)

有:  $T^* \geq (\sum_{1 \leq i \leq n} t_i)/m$  且  $T^* \geq t_i$  ( $1 \leq i \leq n$ )

令  $T$  为近似解的代价,

且近似解中最后处理任务为  $t_h$ , 其在第  $j$  台机器上执行,

则有  $T = T_{start} + t_h$  ( $T_{start}$  为第  $h$  个任务开始执行的时间)

因  $T_{start} \leq ((\sum_{1 \leq i \leq n} t_i) - t_h)/m$

则  $T \leq ((\sum_{1 \leq i \leq n} t_i) - t_h)/m + t_h$

$$= (\sum_{1 \leq i \leq n} t_i)/m + (1 - 1/m)t_h$$

$$\leq T^* + (1 - 1/m) T^*$$

$$= (2 - 1/m) T^* \leq 2T^*$$

