第五章

贪心算法

提要

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

HITWH
SE

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

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

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

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!

HITWH
SE

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！

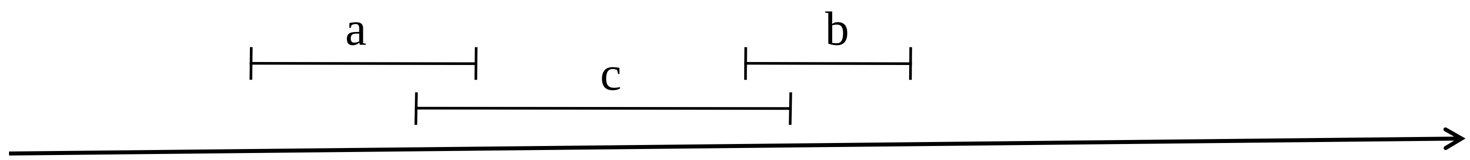
可能存在多种贪心策略！

Greedy算法的基本概念

- 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$\{[s_1, f_1],[s_2, f_2],\ldots[s_n, f_n]\}$，
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

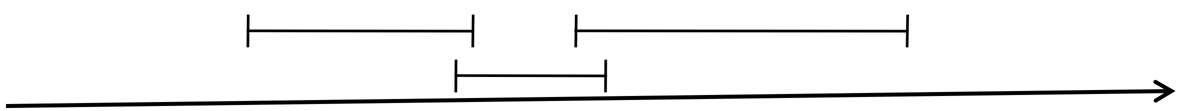


HITWH
SE

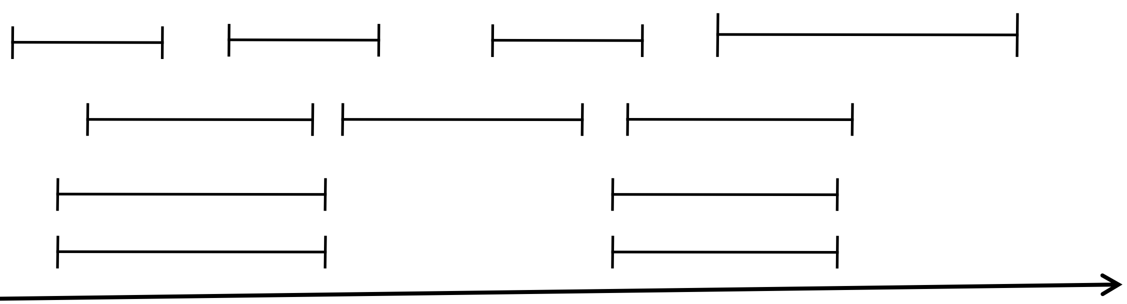
Greedy算法的基本概念

- 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：单源最短路经问题、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
      - 任一实例都至少有一个优化解包含在该实例上使用贪心方法做出的第一个选择

- 贪心算法的正确性证明
  - 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，证明贪心选择性和优化子结构+隐含的归纳法
  - (直接的)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

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

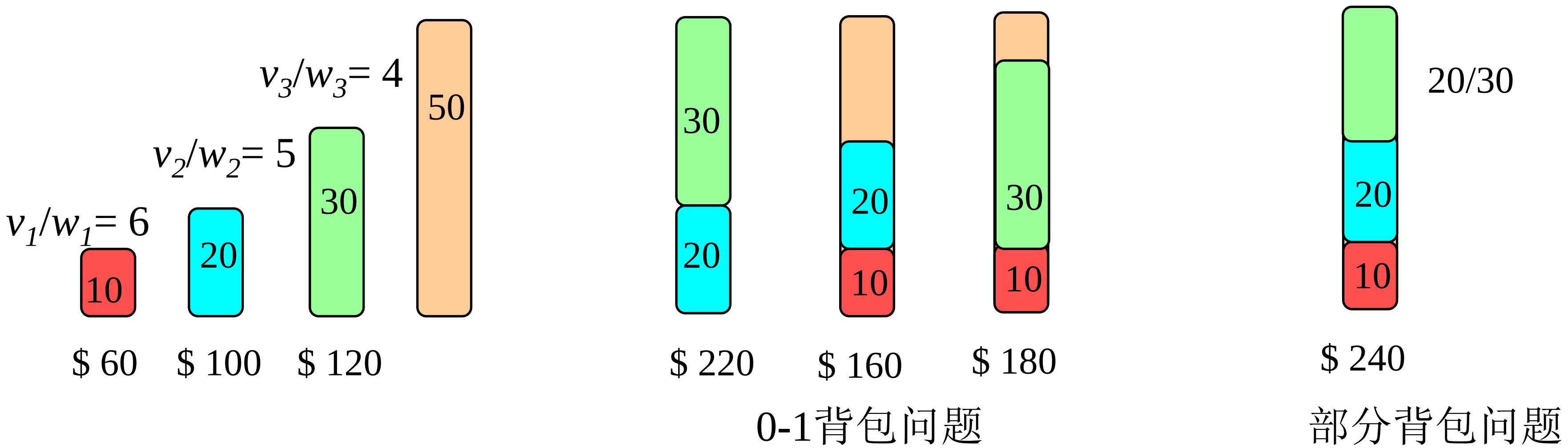
与动态规划方法的比较

- 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

HITWH
SE

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



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

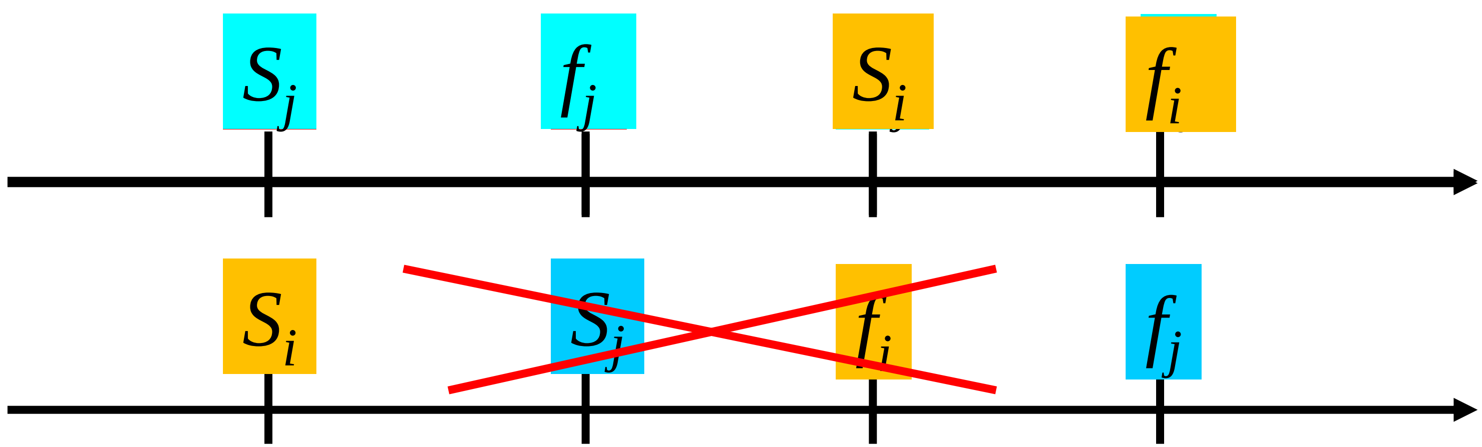
2. 证明：对于1中贪心选择来说，所求解的问题具有优化子结构
3. 证明：对于1中贪心选择来说，所求解的问题具有Greedy选择性(2和3可以交换顺序)
4. 按照1中设计的贪心选择方法设计算法

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

问题的定义

- 活动
  - 设$S=\{1,2,\ldots,n\}$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



问题的定义

- 活动选择问题
  - 输入：$S=\{1, 2, \ldots, n\}$,
    $F=\{ [s_i, f_i] \}$，$n \geq i \geq 1$
  - 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：

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

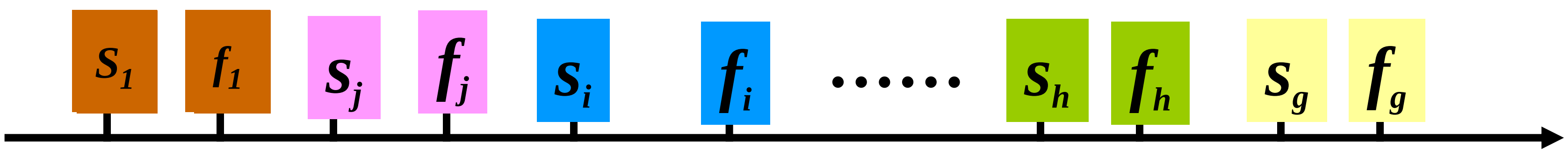
引理1说明活动选择问题具有贪心选择性

解结构分析

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

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$k \ne 1$，令$B=A-\{k\} \cup \{1\}$,

由于A中活动相容，$f_1 \le f_k \le s_j$, 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$S=\{1, 2, ..., n\}$是n个活动集合，$[s_i, f_i]$是活动i的起始终止时间，且$f_1 \leq f_2 \leq ... \leq f_n$，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

贪心选择方法的正确性

引理3. 设 $S=\{1, 2, ..., n\}$ 是 n 个活动集合, $f_1 \le f_2 \le \ldots \le f_n$, $f_{l_0}=0$, l_i 是 $S_i=\{j \in S \mid s_j \ge f_{l_{i-1}}\}$ 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

$S_1=\{1,2,3,4,5,6,7,8,9,10,11\}$, $l_0=0$	$l_1=1$
$S_2=\{4,6,7,8,9,11\}$, $l_1=1$	$l_2=4$
$S_3=\{8,9,11\}$, $l_2=4$	$l_3=8$
$S_4=\{11\}$, $l_3=8$	$l_4=11$
$S_5=\emptyset$, $l_4=11$	

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

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

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

……

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

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

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

贪心选择方法的正确性证明

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

$S_1=\{j\in S \mid s_j\ge f_{l_0}=0\}$
$S_2=\{j\in S \mid s_j\ge f_{l_1}=f_1\}$
$S_3=\{j\in S \mid s_j\ge f_{l_2}\}$
……
$S_k=\{j\in S \mid s_j\ge f_{l_{k-1}}\}$
$S_{k+1}=\{j\in S \mid s_j\ge f_{l_k}\}=\emptyset$

对 $|\cup_{i=1}^{k}\{l_i\}|$ 作归纳法.

当 $|\cup_{i=1}^{k}\{l_i\}|=1$ 时, $i=2...n$, 有 $s_i<f_1\le f_i$, 命题成立.

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

当 $|\cup_{i=1}^{k}\{l_i\}|=m$ 时, 由引理1、引理2, 存在优化解 $A=\{l_1=1\}\cup A_1$.

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

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

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

- 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

（设$f_1 \leq f_2 \leq \ldots . \leq f_n$已排序）

Greedy-Activity-Selector(S, F)

$n \leftarrow$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

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

$=\theta(nlogn)$

算法正确性分析

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) 继而可知, 引理3的贪心选择方法是正确的

(4) Greedy-Activity-Selector算法确实按照引理3的Greedy方法进行选择.

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

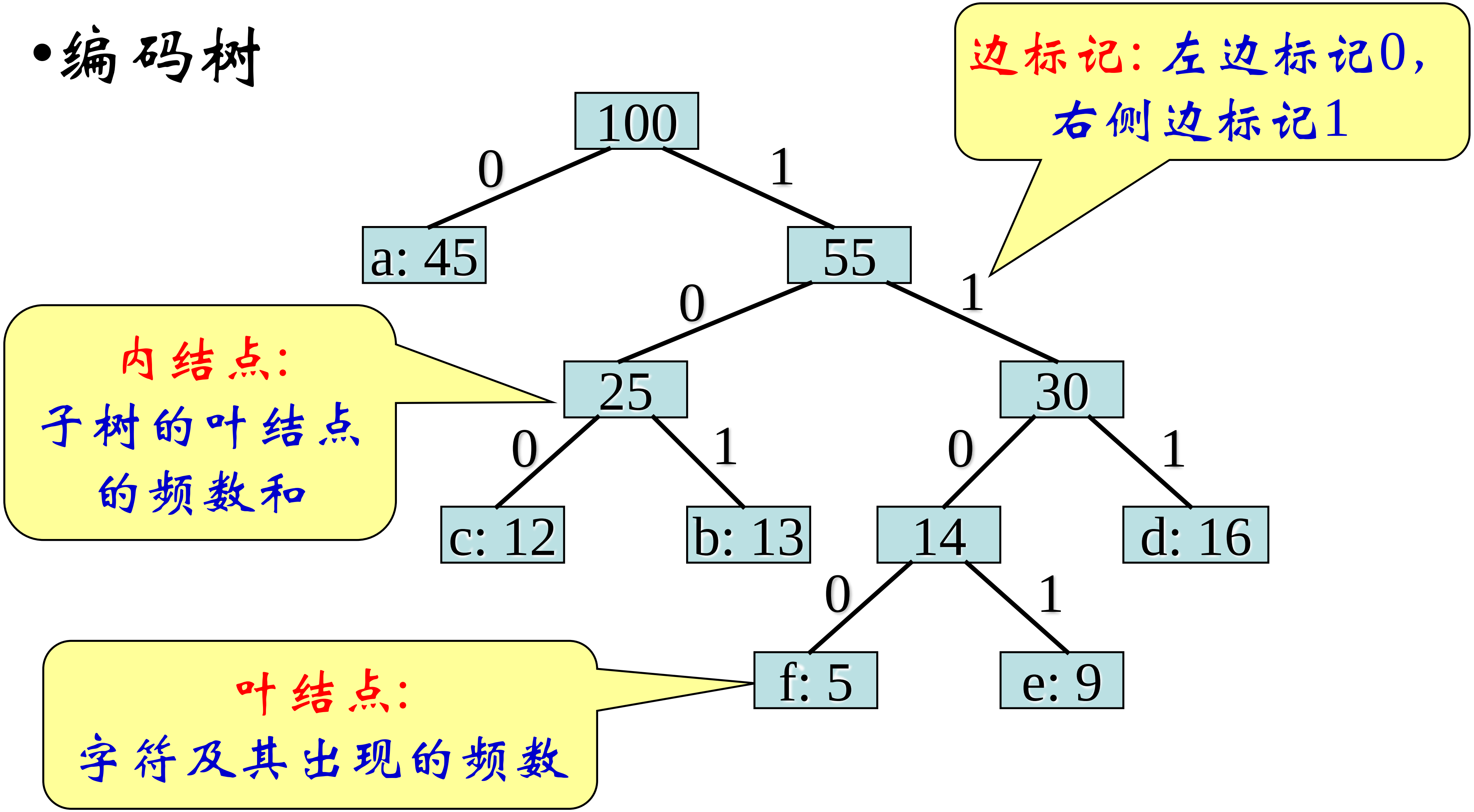
问题的定义

- 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

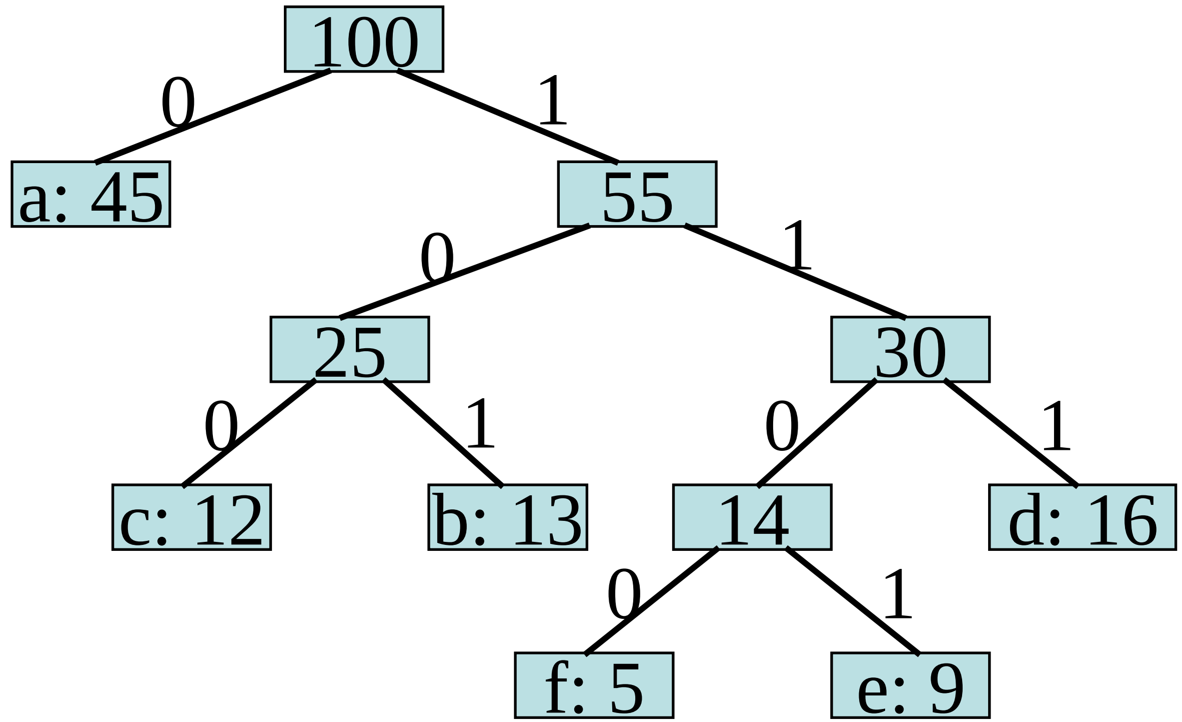
HITWH SE

一颗编码树对应一个编码方案

- 编码树



HITWH
SE



- 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 $C = \{c_1, c_2, \ldots, c_n\}$,

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

输出：具有最小$B(T)$的C的前缀编码树

贪心思想：

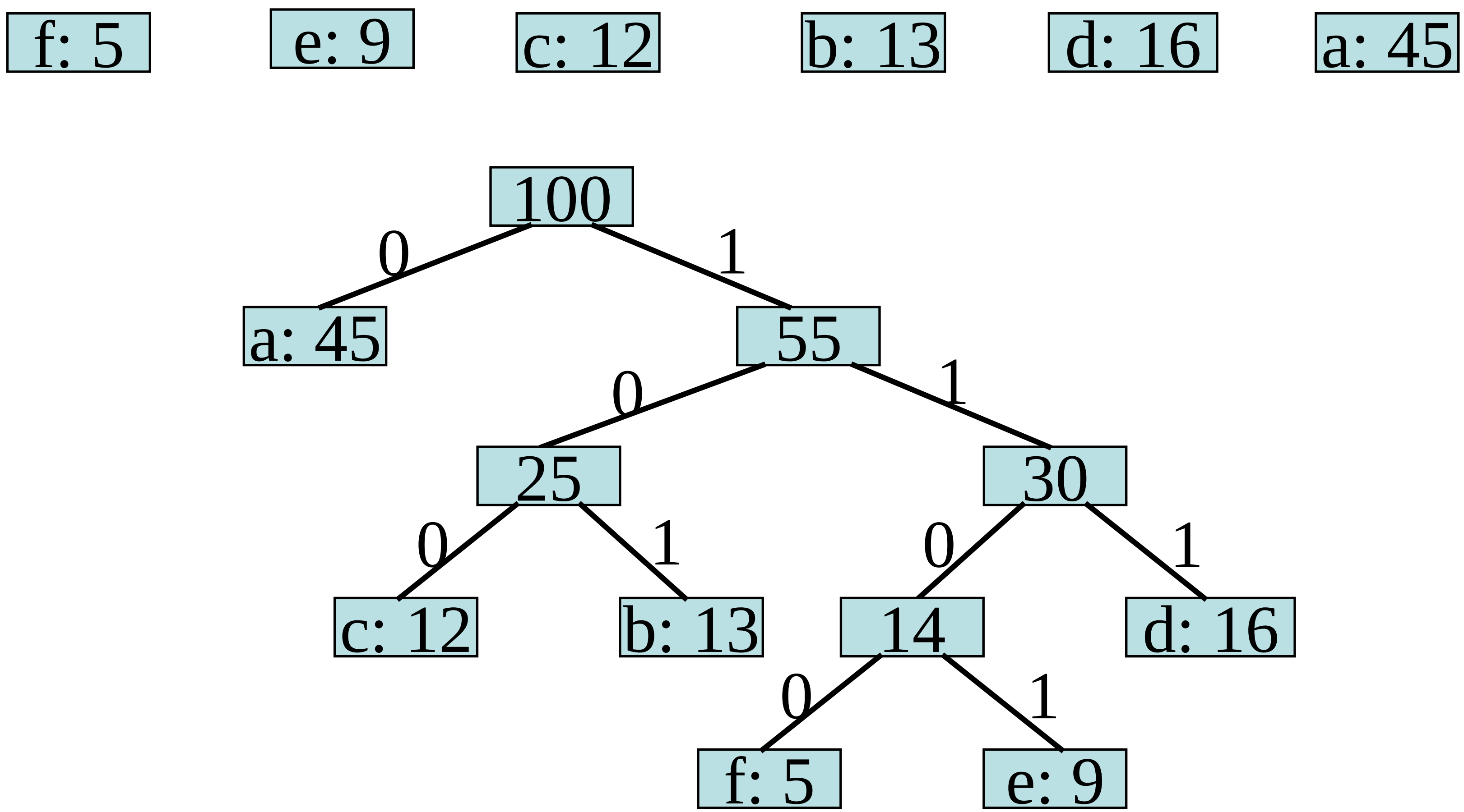
循环地选择具有最低频数的两个结点，生成一棵子树，直至形成树

剩余子问题：？？？

HITWH
SE

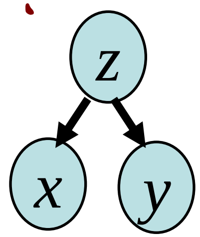
贪心思想：

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



设计贪心选择方法

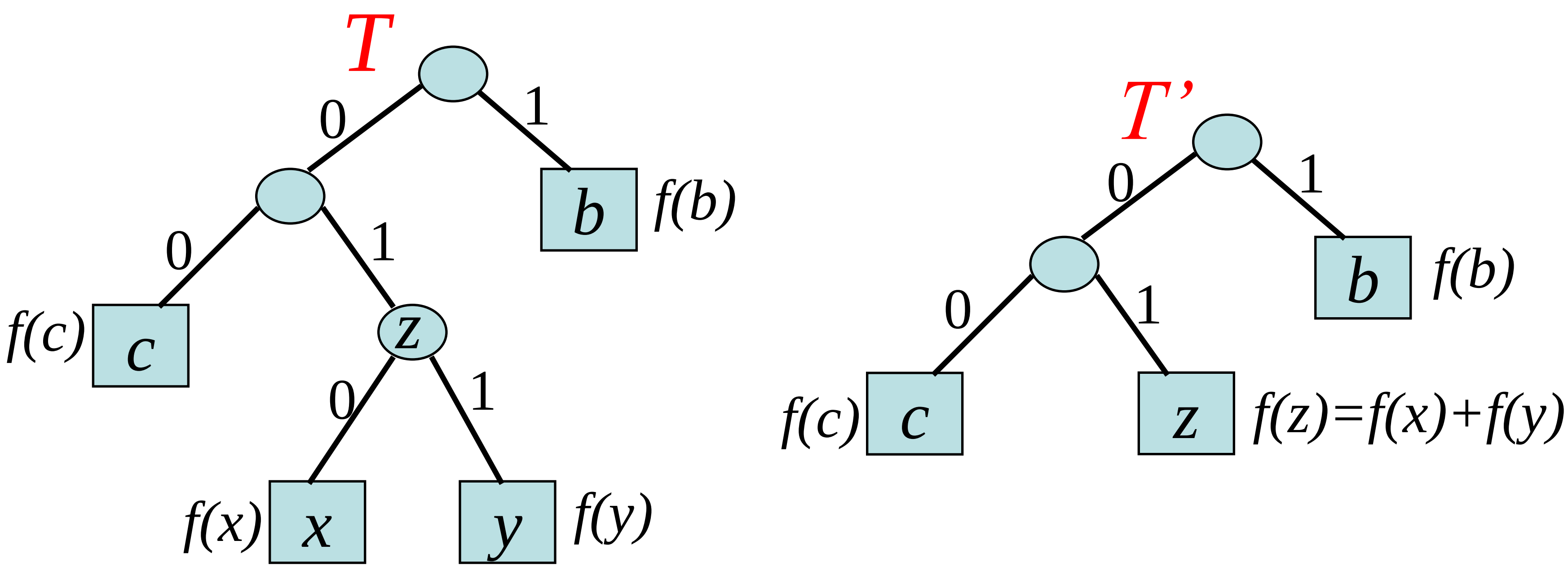
- 贪心选择方法
  - 选择方法：
    - 每次选择具有最低频数的两个节点x和y，构造一个子树：



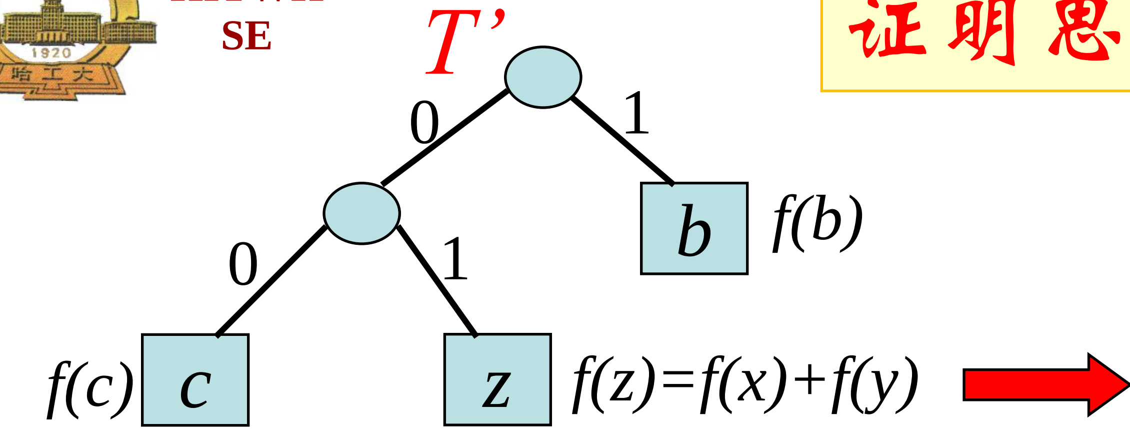
  - 剩余子问题的结构：
    - $C' = C - \{ x, y \} \cup \{ z \}$
    - $F' = F - \{f(x), f(y)\} \cup \{f(z)\}$, $f(z) = f(x) + f(y)$

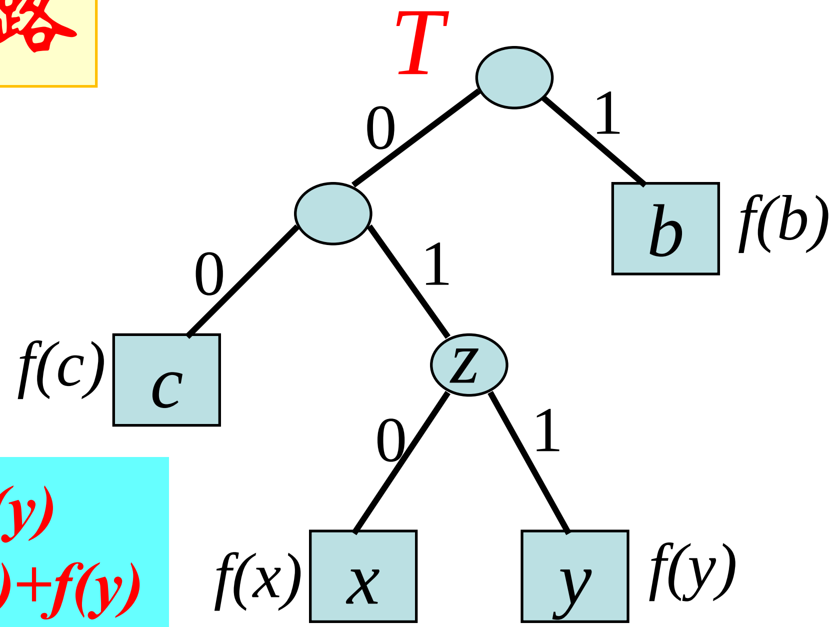
优化解的结构分析

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



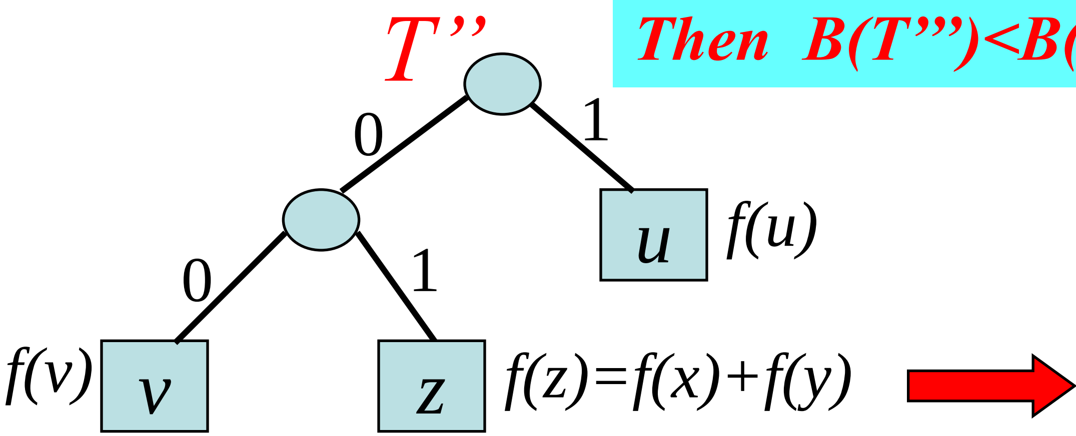


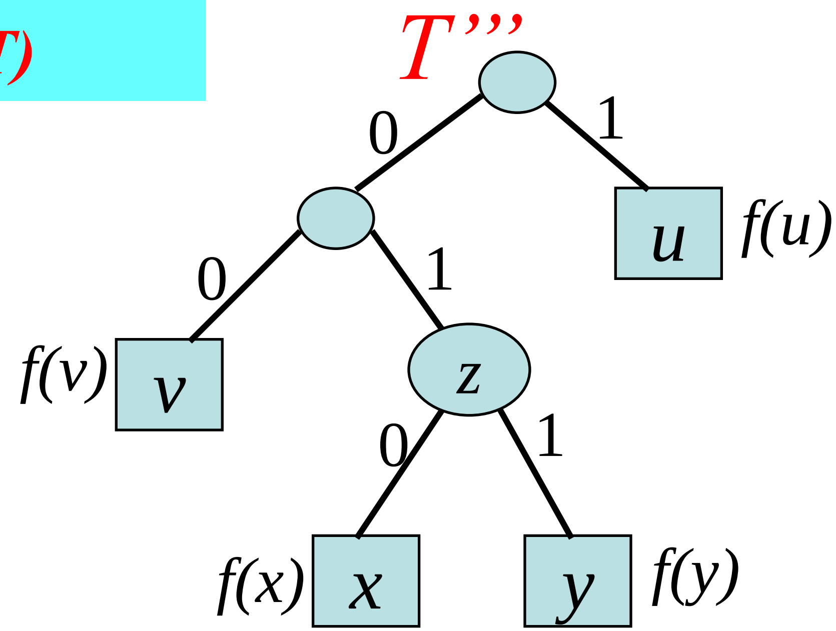
B(T)=B(T’)+f(x)+f(y)

B(T’’’)=B(T’’)+ f(x)+f(y)

If B(T’’)<B(T’)

Then B(T’’’)<B(T)

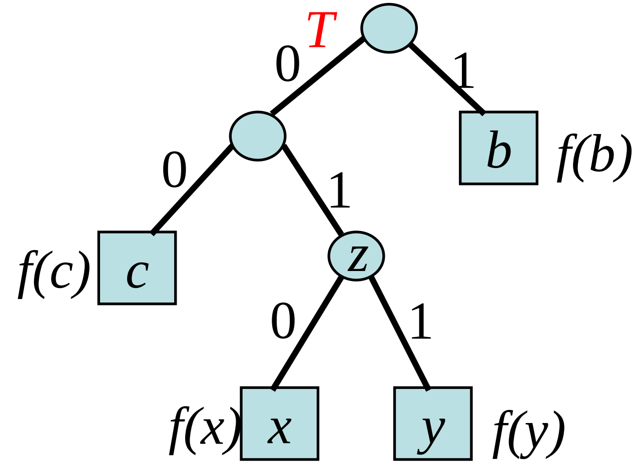




证．往证$B(T)=B(T')+f(x)+f(y)$.

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

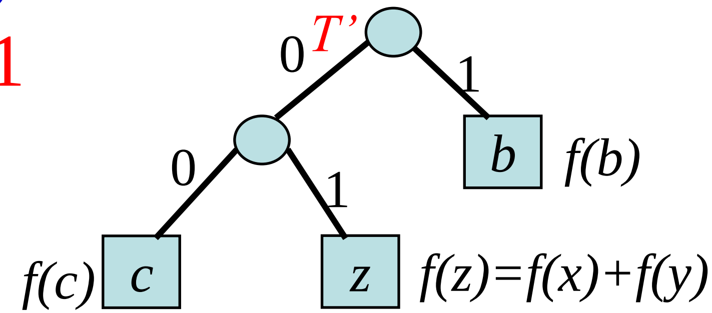
$$B(T')= f(z)d_{T'}(z) +\underline{\sum_{k\in C'-\{z\}}f(k)d_{T'}(k)}$$



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

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

$$B(T)-B(T') = (f(x)+f(y))(d_{T'}(z)+1) - (f(x)+f(y))d_{T'}(z) =f(x)+f(y).$$



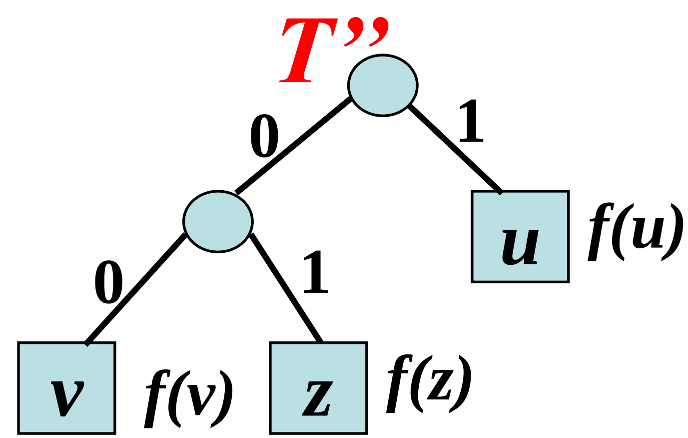
若T'不是C'的优化前缀编码树，

则必存在T''，使$B(T'')<B(T')$.

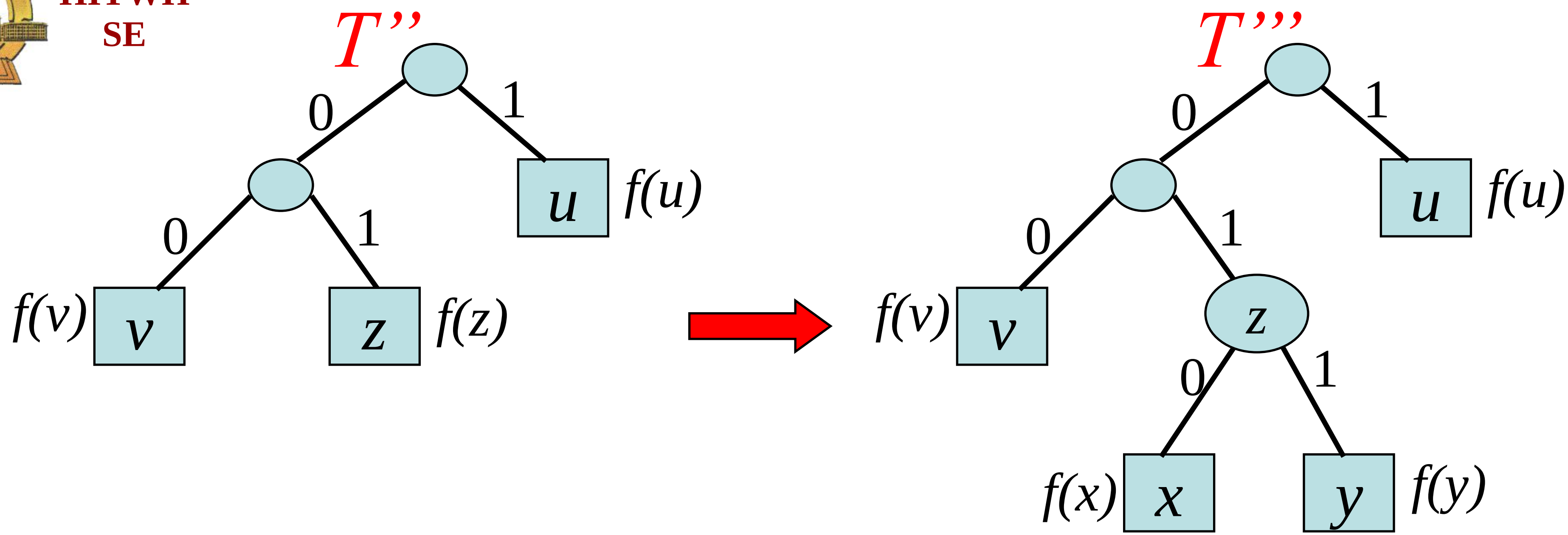
因为z是C'中字符,它必为T''中的叶子.

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'''：



HITWH
SE



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.

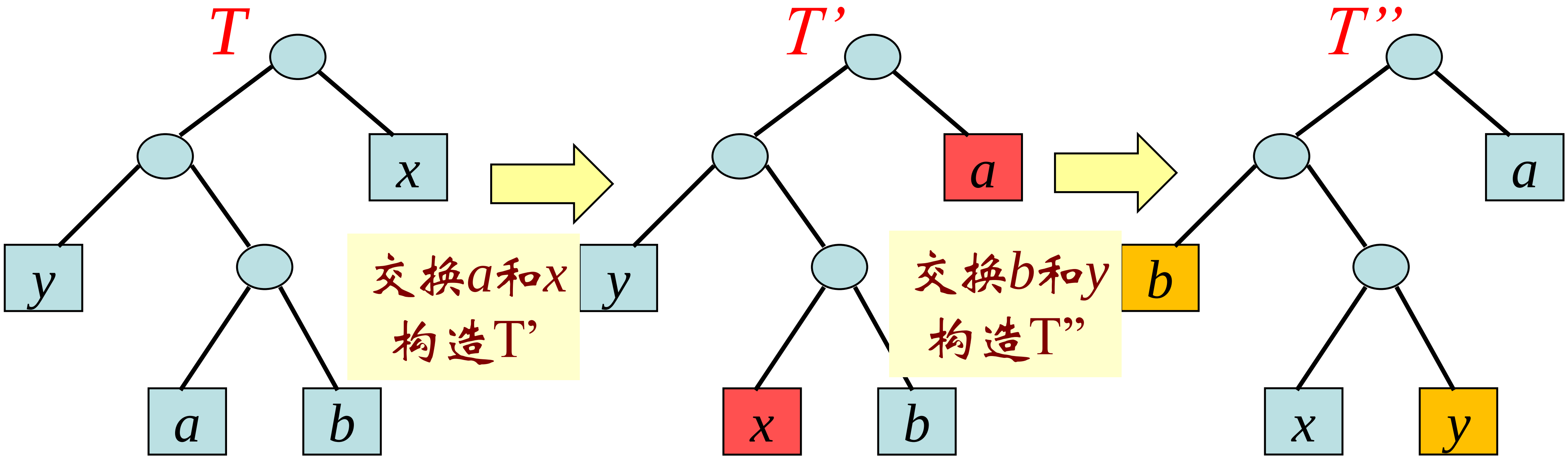
Greedy选择性

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$f(a) \le f(b)$, $f(x) \le f(y)$.

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

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.

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

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

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

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

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

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

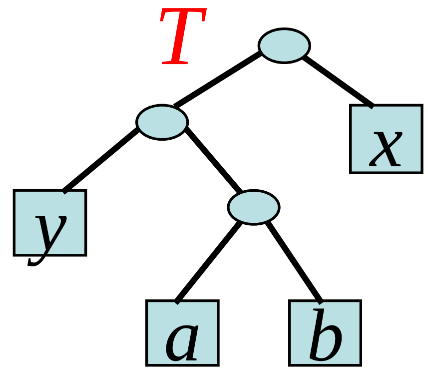
$\therefore B(T)-B(T')\geq 0,\ B(T)\geq B(T')$

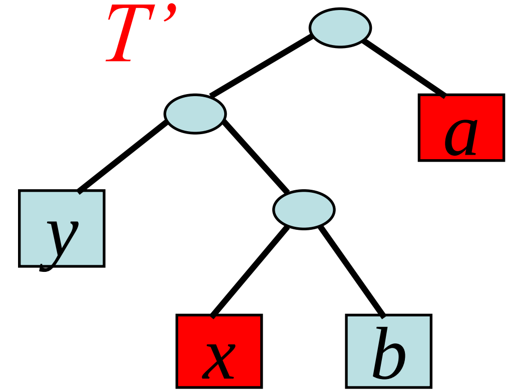
同理可证$B(T')\geq B(T'')$. 于是$B(T)\geq B(T'')$.

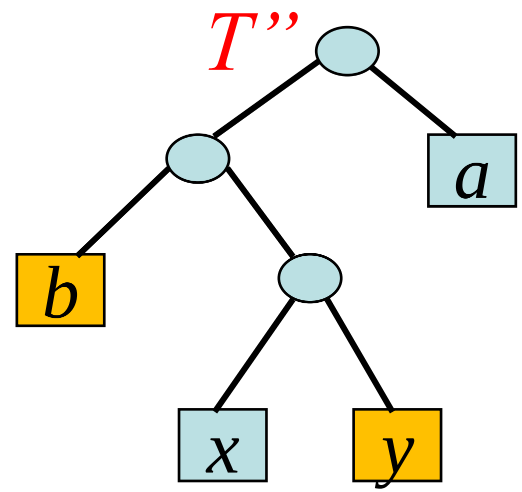
由于T是最优化的，所以$B(T)\leq B(T'')$.

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

- 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. $\quad z \leftarrow$ Allocate-Node( );
4. $\quad left[z] \leftarrow x \leftarrow$ Extract-min(*Q*) /*从*Q*删除*x**/; *O*(log*n*)
5. $\quad right[z] \leftarrow y \leftarrow$ Extract-min(*Q*) /*从*Q*删除*y* */; *O*(log*n*)
6. $\quad f(z) \leftarrow f(x)+f(y)$;
7. $\quad$ Insert(*Q*, *z*, *f*(*z*)); *O*(log*n*)
8. Return 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，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中存在具有最大深度而且相邻的两个的叶子a和b，它们的频率分别为$f(x)$和$f(y)$.

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)≤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）显然，命题在第一次合并之后成立；

（4）如果合并过程中，往证的命题不再成立，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，往证的命题成立，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)≤f(r')，r的高度不小于r'的高度。又由于往证的命题在合并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')≤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。

正确性证明

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（贪心选择性），优化解可通过贪心算法求解。

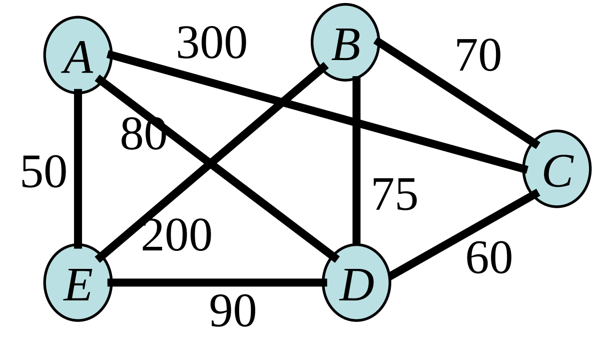
根据引理3，Huffman算法按照引理2的Greedy选择性确定的规则进行局部优化选择，所以Huffman算法产生一个优化前缀编码树。

5.4 Minimal spanning tree problem

- 问题定义
- Kruskal算法
- Prim算法

问题的定义

- 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 $W: E \rightarrow \{实数\}$ 是G的权函数, T的权值定义为$W(T)=\sum_{(u,v)\in T} W(u,v)$.



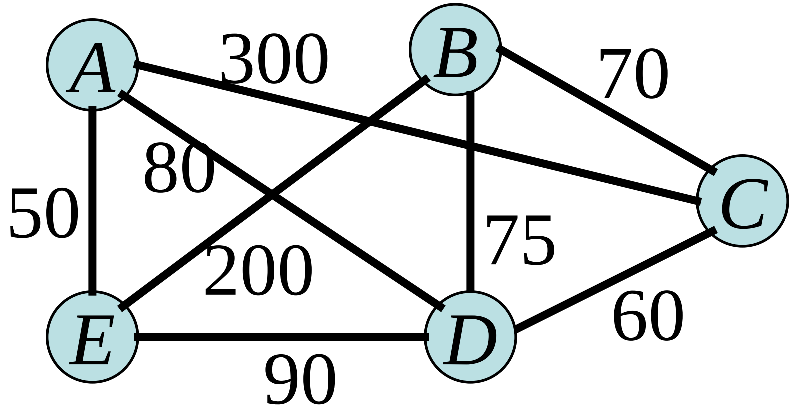
- 最小生成树
  - G的最小生成树是$W(T)$最小的G之生成树.

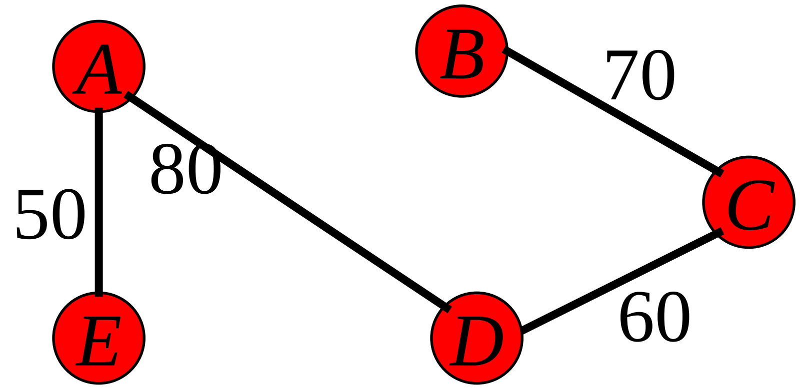
- 问题的定义

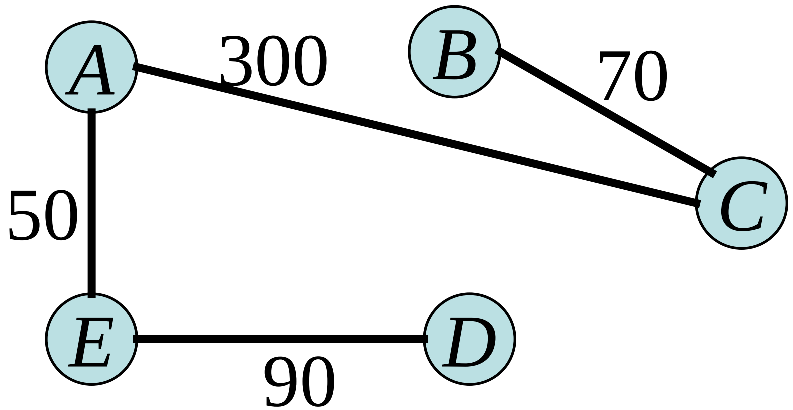
  输入: 无向连通图$G=(V, E)$, 权函数W

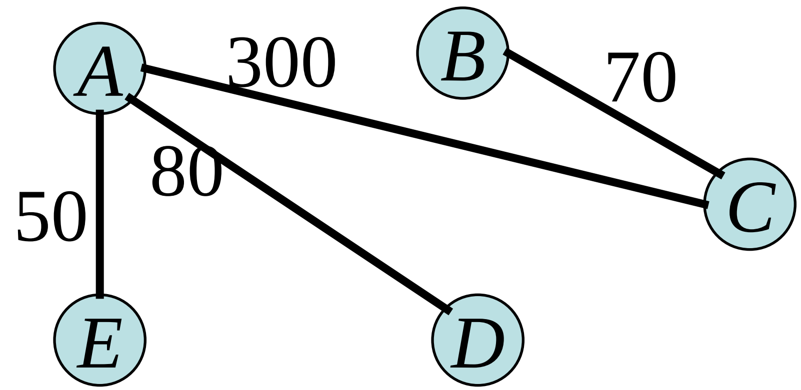
  输出: G的最小生成树

• 实例


A
B
C
D
E
300
70
80
50
200
75
60
90


A
B
C
D
E
70
80
50
60

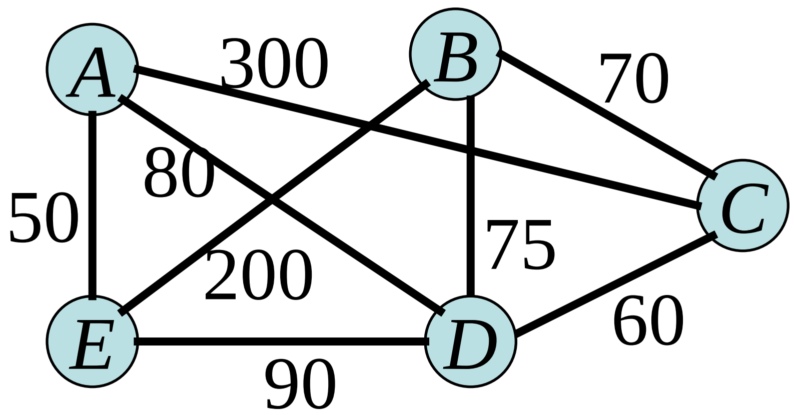

A
B
C
D
E
300
70
50
90


A
B
C
D
E
300
70
80
50

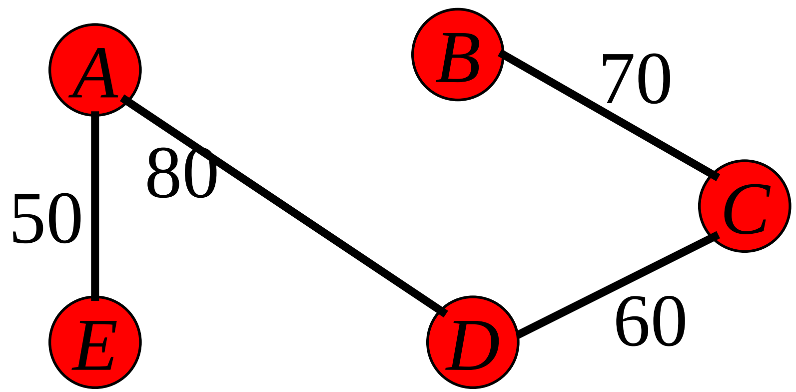
- Kruskal算法
  - 设计贪心选择方法
  - 优化解结构分析
  - Greedy选择性证明
  - 算法复杂性

设计贪心选择方法

- 基本思想



- 初始：A=空；构造森林$G_A=(V, A)$;

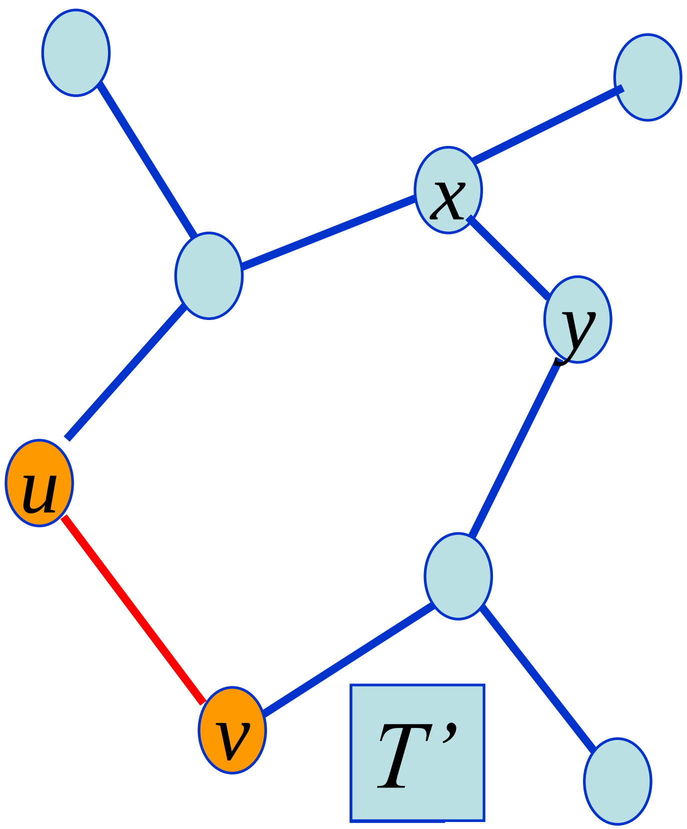


剩余子问题？

- 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'.

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

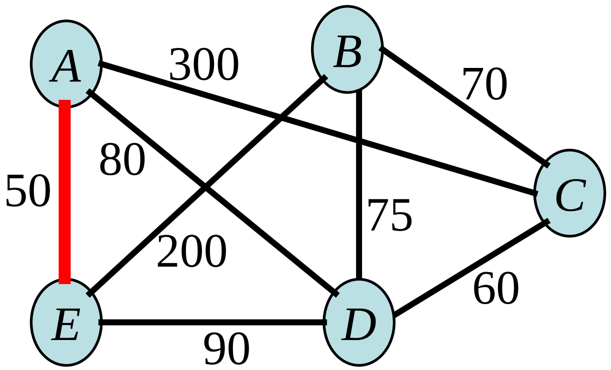
又T是最小生成树, $w(T) \le w(T')$

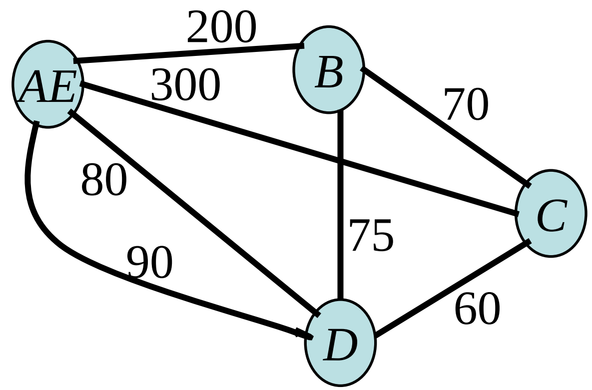
则T'也是一棵MST,且包含边(u,v).

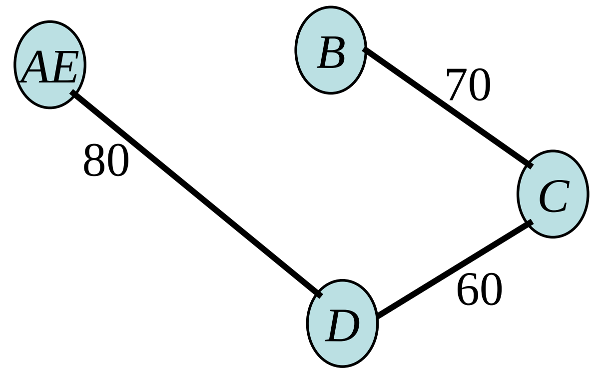
HITWH SE

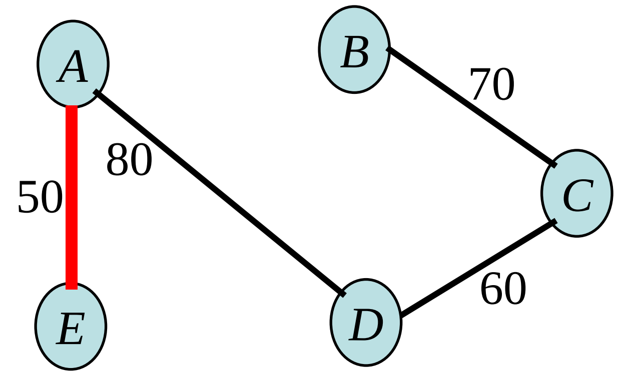
剩余子问题?

优化解的结构分析









- 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$T''=T'|_z^{(u,v)}$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。

Kruskal算法

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

$A=\Phi$;

For $\forall v \in V$ Do

    Make-Set(v); /* 建立只有v的集合 */

按照W值的递增顺序排序E中的边;

For $\forall (u, v) \in E$ (按W值的递增顺序) Do

    If Find-Set(u)$\neq$Find-Set(v) (判断是否出现回路)

    Then $A=A \cup \{(u, v)\}$; Union(u, v); (合并u,v集合)

Return A

MST-Kruskal($G(V,E)$, W)
1. A=Φ;
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)≠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）

算法正确性

定理2. MST-Kruskal(G, W)算法能够产生图G的最小生成树.

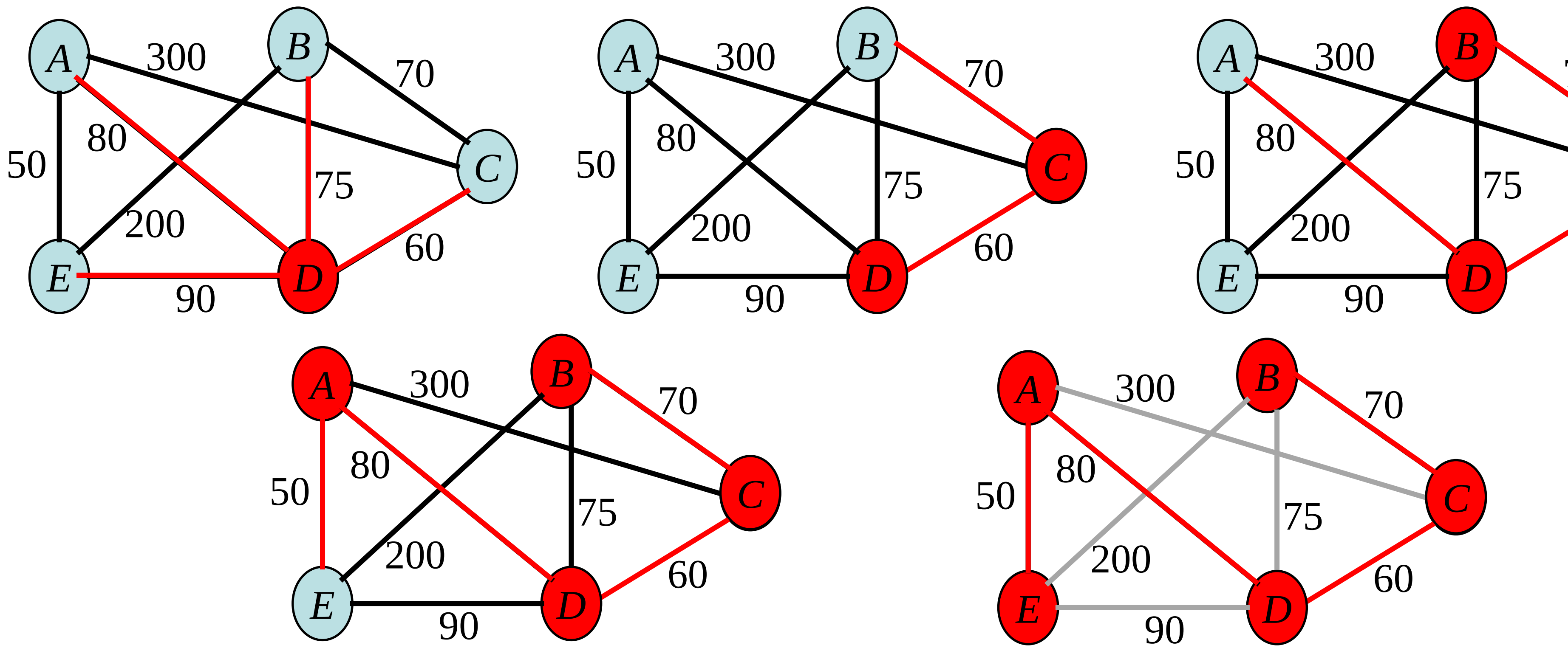
证.

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

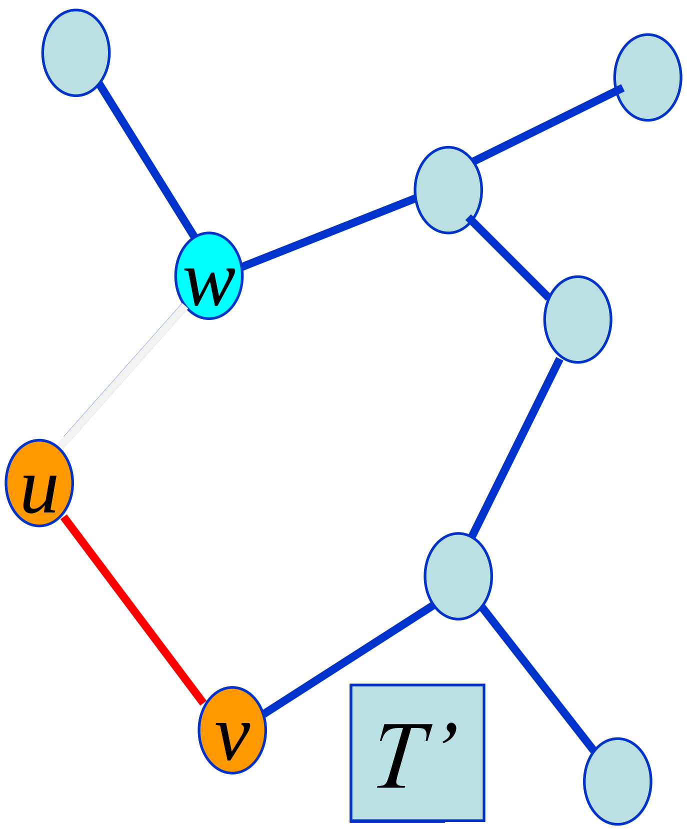
设计贪心选择方法

- 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
  - $C= C\cup\{y\}$



Greedy选择性

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'.

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

又T是最小生成树, $w(T) \le w(T')$

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

HITWH
SE

Prim算法

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 ∀ v∈V[G]  Do
2.        key[v]←+∞        //所有与v邻接的边的最小权值
3.        π[v] ←null        //与v邻接的具有最小权值的边
4.  key[r] ←0
5.  Q ←V[G]                 //Q是一个最小堆
6.  While Q≠∅ do
7.        u ←Extract_Min(Q)
8.        for ∀v∈Adj[u]  do
9.            if v∈Q 且 w(u,v)<key[v]  then
10.                   π[v]    ←u
11.                   key[v] ←w(u,v)        //更新信息
12. Return  A={(v, π[v])| v ∈V[G] -r}
```

Line 1-5: $O(|V|)$

Line 6循环$O(|V|)$次

Line 7: 每次$O(\log |V|)$

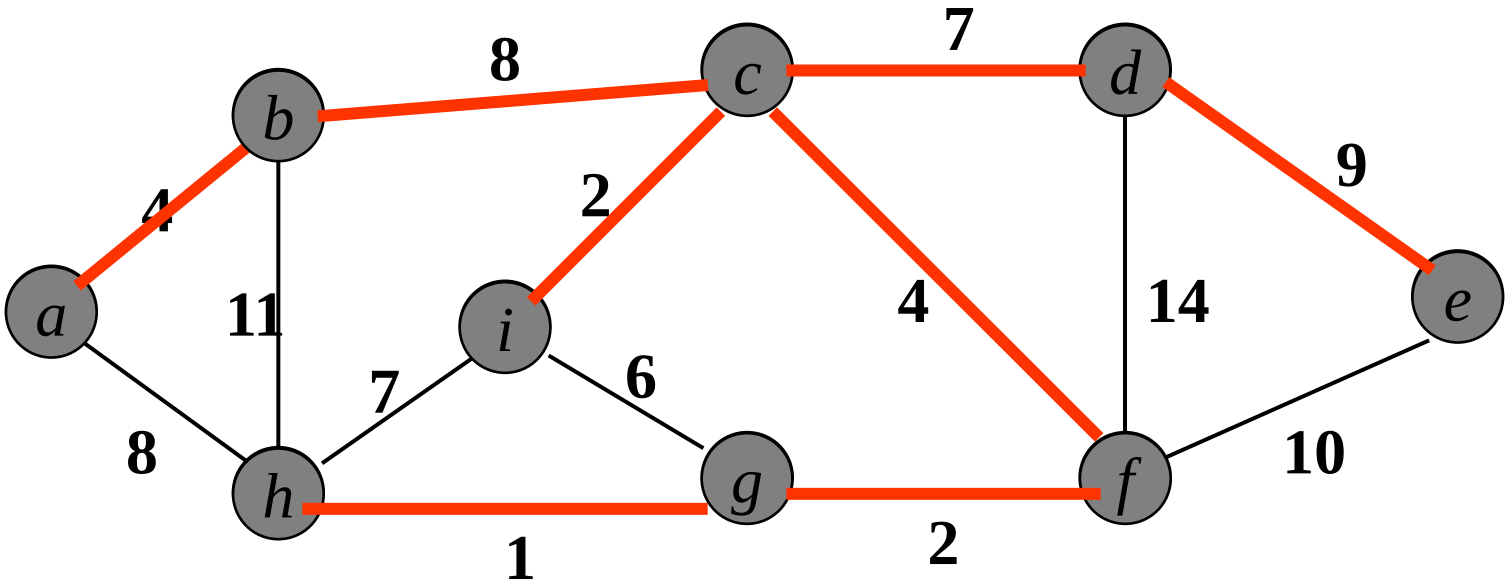
Line 8循环$O(|E|)$次

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



	π[v]	key[v]
a	null	0
b	a	4
c	b	8
d	c	7
e	d	9
f	c	4
g	f	2
h	g	1
i	c	2

总结

- 设计贪心选择方法
- 优化解的结构分析
  - Greedy选择性证明
  - 优化子结构证明
- 算法设计
- 算法的正确性分析
  - 算法执行贪心选择方法
- 算法的复杂性分析