第4章 贪心法
（Greedy Approach）

4.1贪心法的设计思想

例4.1 活动选择问题

输入：$S=\{1, 2, \dots, n\}$为n 项活动的集合，s_i, f_i 分别为活动 i 的开始和结束时间，活动 i 与 j 相容 $\Leftrightarrow$ $s_i \ge f_j$ 或 $s_j \ge f_i$.

求：最大的两两相容的活动集 A

实例

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

策略1：排序使得 $s_1 \le s_2 \le \dots \le s_n$，从前向后挑选

策略2：排序使得 $f_1 - s_1 \le f_2 - s_2 \le \dots \le f_n - s_n$，从前向后挑选

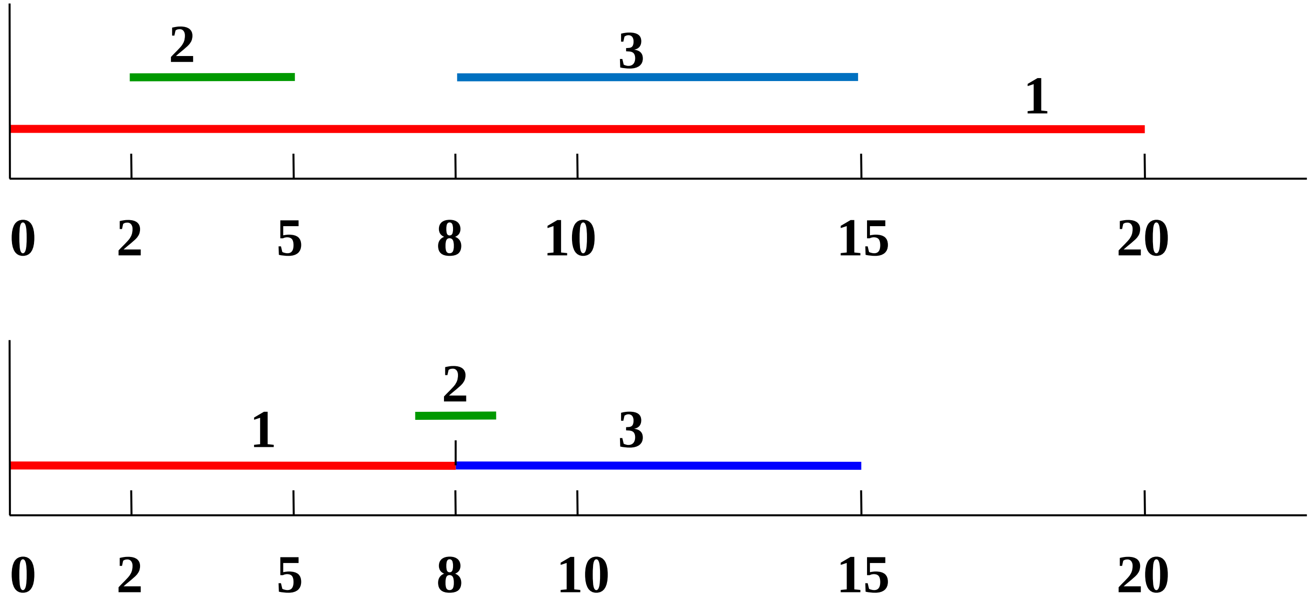
策略3：排序使得 $f_1 \le f_2 \le \dots \le f_n$，从前向后挑选

以上策略中的挑选都要注意满足相容性条件

两个反例

策略1：S={1, 2, 3}， s_1=0, f_1=20, s_2=2, f_2=5, s_3=8, f_3=15

策略2：S={1, 2, 3}， s_1=0, f_1=8, s_2=7, f_2=9, s_3=8, f_3=15



贪心算法

算法4.1 Greedy Select

输入：活动集 S，s_i，f_i，i =1, 2, …, n，且 $f_1 \le \ldots \le f_n$

输出：$A \subseteq S$，选中的活动子集

1. $n \leftarrow length[S]$ // 活动个数
2. $A \leftarrow \{1\}$
3. $j \leftarrow 1$ //已选入的最后一个活动的标号
4. for $i \leftarrow 2$ to n do
5.   if $s_i \ge f_j$ //判断相容性
6.   then $A \leftarrow A \cup \{ i \}$
7. $j \leftarrow i$
8. return A

最后完成时间 $t = \max \{ f_k : k \in A \}$

算法运行实例

输入：$S = \{1, 2, \ldots, 10\}$

i	1	2	3	4	5	6	7	8	9	10
s_i	1	3	0	5	3	5	6	8	8	2
f_i	4	5	6	7	8	9	10	11	12	13

解：$A = \{1, 4, 8\}$，$t = 11$

时间复杂度：排序+活动选择=$O(n\log n)+O(n)=O(n\log n)$

问题：如何证明该算法对所有的实例都能得到正确的解？

算法的正确性证明

定理4.1 算法Select 执行到第 k 步, 选择 k 项活动 $i_1 = 1, i_2, \ldots,$ i_k, 那么存在最优解 A 包含 $i_1=1, i_2, \ldots, i_k$.

根据定理：算法至多到第 n 步得到最优解

证：$S=\{1, 2, \ldots, n\}$是活动集，且 $f_1 \le \ldots \le f_n$

归纳基础：$k=1$, 证明存在最优解包含活动 1.

任取最优解A，A中的活动按截止时间递增排列．如果A的第一个活动为j，$j \ne 1$, 令

$$A' = (A - \{j\}) \cup \{1\},$$

由于 $f_1 \le f_j$, A'也是最优解，且含有1.

算法正确性证明（续）

归纳步骤：假设命题对 k 为真, 证明对 k+1 也为真.

算法执行到第 k 步, 选择了活动 i_1=1, i_2, …, i_k, 根据归纳假设存在最优解 A 包含 i_1= 1, i_2, … , i_k ,

A中剩下的活动选自集合 $S' = \{ i \mid i \in S, s_i \geq f_k \}$, 且

$$A = \{ i_1, i_2, \ldots , i_k \} \cup B$$

其中B是S'的最优解.（<u>若不然, S'的最优解为B^*, B^*的活动比 B多，那么$B^* \cup \{1, i_2, \ldots , i_k\}$是 S 的最优解，且比 A的活动多,与 A 的最优性矛盾.）</u>

根据归纳基础，存在 S'的最优解B'含有S'中的第一个活动，即i_{k+1}, 且$|B'|=|B|$, 于是

$$\{i_1, i_2, \ldots, i_k\} \cup B' = \{i_1, i_2, \ldots, i_k, i_{k+1}\} \cup (B' - \{i_{k+1}\})$$

也是原问题的最优解.

贪心算法的特点

设计要素：

(1) 贪心法适用于组合优化问题.

(2) 求解过程是多步判断过程，最终的判断序列对应于问题的最优解.

(3) 判断依据某种“短视的”贪心选择性质，性质的好坏决定了算法的成败.

(4) 贪心法必须进行正确性证明.

贪心法的优势：

算法简单，时间和空间复杂性低.

4.2 贪心法的正确性证明

数学归纳法

1. 叙述一个描述算法正确性的命题$P(n)$，n为算法步数或者问题规模
2. 归纳基础：$P(1)$ 或 $P(n_0)$为真，n_0为某个自然数
3. 归纳步骤：$P(k) \Rightarrow P(k+1)$ 第一数学归纳法

   $\forall k(k<n)P(k) \Rightarrow P(n)$ 第二数学归纳法

交换论证

1. 分析算法的解的结构特征
2. 从一个最优解逐步进行结构变换(替换成分、交换次序等)得到一个新的解（结构上与贪心算法的解更接近）
3. 证明：上述变换最终得到算法的解，且变换在有限步结束，每步变换都保持解的最优性不降低.

最优装载 Loading

例4.2 最优装载

n 个集装箱1, 2, …, n 装上轮船，集装箱 i 的重量 w_i, 轮船装载重量限制为C， 无体积限制. 问如何装使得船上的集装箱最多？不妨设 $w_i \le C$.

$$\max \sum_{i=1}^{n} x_i$$

$$\sum_{i=1}^{n} w_i x_i \le C$$

$$x_i = 0,1 \quad i = 1,2,\ldots,n$$

贪心法：将集装箱按照从轻到重排序，轻者先装.

正确性证明

命题$P(n)$，n为算法步数或者问题规模

题目： 对装载问题将箱子从轻到重排序后，$N=\{1, 2, \ldots, n\}$,第i个箱子重量 w_i，限重C.

思路1： 按照"轻者先装"贪心算法，执行到第 k 步, 选择了前 k个箱子1, 2, …, k, 那么存在最优解 I 包含这k个箱子。

证：归纳基础: k=1，假设一个最优解I'中最轻的箱子为j, 若j不为1，用箱子1替换I' 中的j得到某解I, 由于$w_1 \le w_j$, I必然也是最优解;

归纳步骤：假设第k步时命题成立，即最优解 I中包含前k个最轻的箱子，不妨设$\sum_{i=1}^{k+1} w_i \le C$. 对于新的装载问题$N'=\{k+1, k+2, \ldots, n\}$, 限重$C'=C-\sum_{i=1}^{k} w_i$，不妨设$w_{k+1} \le C'$, 由归纳基础知，必有某最优解$I'$包含$k$+1. 那么对于原问题$N= N' \cup \{1, \ldots, k\}$， $C=C'+\sum_{i=1}^{k} w_i$, 必然有$I= I' \cup \{1, \ldots, k\}$是最优解，即$\{1, \ldots, k+1\}$ 包含在最优解I中。

若不然，存在包含 $\{1, \ldots, k\}$的 N 的最优解 I^* (若 I^* 中没有$\{1, \ldots, k\}$，用 $\{1, \ldots, k\}$ 替换 I^* 中前k个最轻箱子可得另一最优解)，且 $|I^*| > |I|$; 那么 $I^* - \{1, \ldots, k\}$是关于 N' 和C'的解且 $|I^* - \{1, \ldots, k\}| > |I - \{1, \ldots, k\}| = |I'|$， 与 I' 的最优性矛盾.

正确性证明

命题$P(n)$，n为算法步数或者问题规模

思路2，定理4.2 对装载问题任何规模为k 的输入，"轻者优先"贪心法都可以得到最优解.（$N=\{1, 2, \ldots, n\}$,第i个箱子重量 w_i，且为升序排列，限重C.）

证： $k=1$, 只有1个箱子，算法显然正确.

假设规模为k时命题成立，即对于规模为 k 个箱子的装载问题，贪心法都可以得到最优解.

考虑规模为$k+1$的问题，$N = \{1, 2, \ldots, k+1\}$, 限重$C$. 令$N' = \{2,3,\ldots,k+1\} = N-\{1\}$, $C' = C-w_1$，则为规模为k的装载问题. 由归纳假设知对其应用贪心法可得到最优解I'. 再考虑原问题 $N = N'+\{1\}$, 令$I = I'\cup\{1\}$，则 I (算法解)是关于 N 的最优解.

若不然，存在包含 1 的 N 的最优解 I^*（若 <u>I^* 中没有1</u>，<u>用 1 替换 I^* 中最轻箱子可得另一最优解</u>），且$|I^*| > |I|$； 那么 $I^* - \{1\}$是关于 N' 和C'的解且 $|I^*-\{1\}| > |I-\{1\}| = |I'|$ 与 I' 的最优性矛盾.

最小延迟调度

例4.3 最小延迟调度

给定客户集合A，$\forall i \in A$，t_i为服务时间，d_i为希望的完成时间，t_i, d_i 为正整数. 一个调度是函数 $f: A \to \mathrm{N}$，$f(i)$为客户 i 的开始时间. 求最大延迟达到最小的调度，即求f使得

$$\min_{f}\{\max_{i \in A}\{f(i)+t_i-d_i\}\}$$

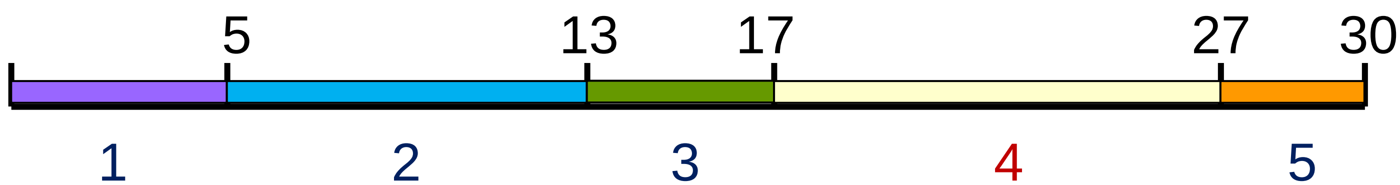
$$\forall i, j \in A, i \neq j, f(i)+t_i \leq f(j) \text{ or } f(j)+t_j \leq f(i)$$

实例

$A=\{1, 2, 3, 4, 5\}$, $T = <5, 8, 4, 10, 3>$, $D=<10, 12, 15, 11, 20>$

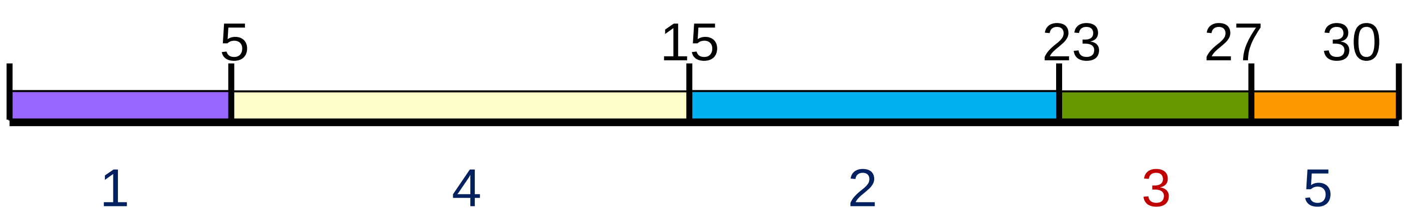
调度1： $f_1(1)=0$, $f_1(2)=5$, $f_1(3)=13$, $f_1(4)=17$, $f_1(5)=27$

各任务延迟：0, 1, 2, 16, 10； 最大延迟：16



调度2： $f_2(1)=0$, $f_2(2)=15$, $f_2(3)=23$, $f_2(4)=5$, $f_2(5)=27$

各任务延迟：0, 11, 12, 4, 10；最大延迟：12



贪心策略选择

贪心策略1：按照 t_i 从小到大安排任务

贪心策略2：按照 $d_i - t_i$ 从小到大安排任务

贪心策略3：按照 d_i 从小到大安排任务

策略1 对某些实例得不到最优解.

反例：t_1=1, d_1=100, t_2=10, d_2=10

按策略1，执行顺序为 *1*、*2*，任务2产生延迟1.

若执行顺序为 *2* 、*1*，则无延迟.

策略2 对某些实例得不到最优解.

反例：t_1=1, d_1=2, t_2=10, d_2=10

按策略1，执行顺序为 *2* 、*1* ，任务1产生最大延迟11-2=9.

若执行顺序为 *1*、*2* ，则最大延迟为任务2：11-10=1.

算法设计

算法4.3 Schedule

输入： ***A*, *T*, *D***

输出： ***f***

1. 排序A使得 $d_1 \le d_2 \le \ldots \le d_n$
2. $f(1) \leftarrow 0$
3. $i \leftarrow 2$
3. while $i \le n$ do
4. $\quad f(i) \leftarrow f(i-1)+t_{i-1}$ //任务i-1结束时刻是任务 i 开始时刻
5. $\quad i \leftarrow i+1$

设计思想：按完成时间从早到晚安排任务，没有空闲

交换论证：正确性证明

该贪心算法的解的性质：

(1) 没有空闲时间，没有逆序.

(2) 逆序 (i, j): $f(i) < f(j)$ 且 $d_i > d_j$

引理4.1 所有没有逆序、没有空闲时间的调度具有相同的最大延迟.

证： 设 f 没有逆序，在 f 中具有相同完成时间d的客户 $i_1, i_2, \ldots, i_k$ 必被连续安排. 在这k个客户中最大延迟是最后一个客户，被延迟的时间是

$$t_0 + \sum_{j=1}^{k} t_{i_j} - d$$

t_0为开始安排k个客户的时刻

与 $i_1, i_2, \ldots, i_k$ 的排列次序无关.

交换论证

若某最优解存在空闲，显然去掉空闲仍然最优

证明思想：从一个没有空闲时间的最优解出发，在不改变最优性的条件下，转变成没有逆序的解. 根据引理 4.1，这个解和算法的解具有相同的最大延迟.

证明要点

(1) 相邻逆序的存在性：如果一个最优调度存在逆序，那么存在 $i < n$ 使得 $(i, i+1)$ 构成一个逆序.

(2) 交换相邻的逆序 i 和 j，得到的解的调度仍旧最优.

(3) 每次交换后逆序数减1，至多经过 $n(n-1)/2$ 次交换得到一个没有逆序的最 优调度.

定理4.3 在一个没有空闲时间的最优解中，最大延迟是 r，如果仅对具有相邻逆序的客户进行交换，得到的解的最大延迟不会超过 r。

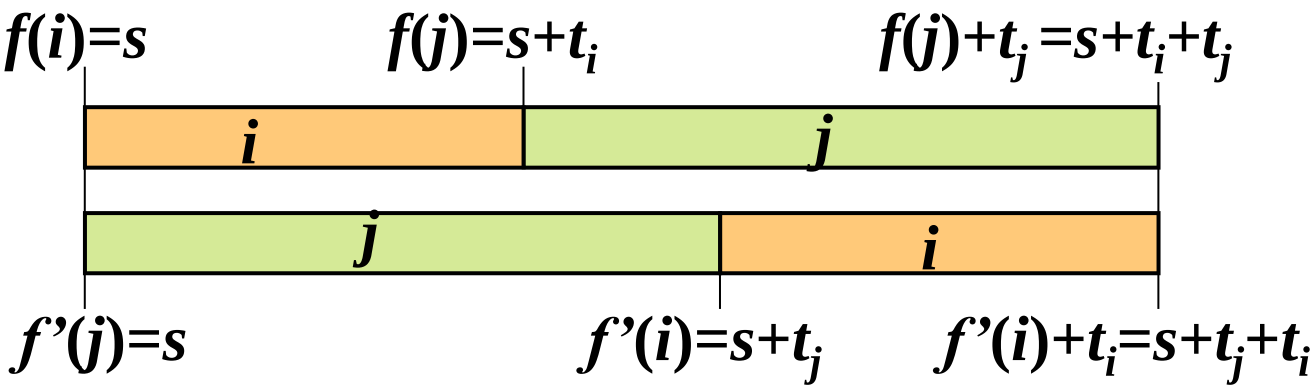
交换相邻逆序 (i, j) 不影响最优性

f 交换相邻逆序任务 i 和 j， $d_j<d_i$，得到调度 f'

(1) 交换 i, j 对其他客户的延迟时间没影响

(2) 交换后不增加 j 的延迟

(3) i 在 f' 的延迟delay(f', i)小于 j 在 f 的延迟 delay(f, j)，因此小于f 的最大延迟 r



$$\text{delay}(f',i) = s + t_j + t_i - d_i < \text{delay}(f, j) \le r$$

$$\text{delay}(f, j) = s + t_i + t_j - d_j$$

$$\boxed{d_j<d_i \Rightarrow \text{delay}(f',i) < \text{delay}(f, j)}$$

最小延迟调度证明小结

证明步骤总结：

第一步：没有空闲、没有逆序的解等价，即它们最大延迟相同；

第二步：如果一个最优解有空闲，则去掉该空闲仍为最优解；

第三步：只需证：没有空闲、有逆序的最优解，可以转化为没有空闲、没有逆序的最优解；

第四步：由于逆序可以通过有限步相邻逆序交换顺序，可以转化为没有逆序的等价解.

因此第三步等价于证： 没有空闲、有逆序的最优解，经过一次相邻逆序交换，仍未最优解。

4.3 得不到最优解的处理方法

讨论对于哪些输入贪心法能得到最优解：输入条件
讨论贪心法的解最坏情况下与最优解的误差（见第8章）

例4.4 找零钱问题

设有 n 种零钱， 重量分别为 $w_1, w_2, \ldots, w_n$，价值分别为 $v_1=1, v_2, \ldots, v_n$，$v_1<v_2<\ldots<v_n$。需要付的总钱数是 Y.不妨设币值和钱数都为正整数．问：如何付钱使所付钱的总重最轻?

令选用第 i 种硬币的数目是 x_i，$i=1,2,\ldots,n$

$$\min\{\sum_{i=1}^{n} w_i x_i\}$$

$$\sum_{i=1}^{n} v_i x_i = Y, \quad x_i \in \mathrm{N}, \quad i = 1,2,\ldots,n$$

动态规划算法

属于整数规划问题，动态规划算法可以得到最优解

设 $F_k(y)$ 表示用前 k 种零钱，总钱数为 y 的最小重量

递推方程

$$F_{k+1}(y)=\min_{0\le x_{k+1}\le\left\lfloor \frac{y}{v_{k+1}}\right\rfloor}\{F_k(y-v_{k+1}x_{k+1})+w_{k+1}x_{k+1}\}$$

$$F_1(y)=w_1\left\lfloor \frac{y}{v_1}\right\rfloor=w_1 y$$

$$k=2,\ldots,n，\quad y=0,1,\ldots,Y$$

Greedy算法

假设 $$\frac{w_1}{v_1} \geq \frac{w_2}{v_2} \geq \ldots \geq \frac{w_n}{v_n}$$

使用前 k 种零钱，总钱数为 y
贪心法的总重为 $G_k(y)$，则有如下递推方程

$$G_{k+1}(y) = w_{k+1}\left\lfloor \frac{y}{v_{k+1}} \right\rfloor + G_k(y \bmod v_{k+1}) \quad k > 1$$

$$G_1(y) = w_1\left\lfloor \frac{y}{v_1} \right\rfloor = w_1 y$$

Greedy算法

假设 $$\frac{w_1}{v_1} \ge \frac{w_2}{v_2} \ge \dots \ge \frac{w_n}{v_n}$$

使用前 k 种零钱，总钱数为 y

贪心法的总重为 $G_k(y)$，则有如下递推方程

$$G_{k+1}(y) = w_{k+1}\left\lfloor \frac{y}{v_{k+1}} \right\rfloor + G_k(y \bmod v_{k+1}) \quad k > 1$$

$$G_1(y) = w_1\left\lfloor \frac{y}{v_1} \right\rfloor = w_1 y$$

实例：v_1=1, v_2=5, v_3=14, v_4=18, w_i=1, i=1,2,3,4. y=28 .

贪心法解：x_4=1, x_2=2, x_1=x_3=0，总重3.

最优解：x_3=2, x_1=x_2=x_4=0，总重2.

动态规划与贪心算法

动态规划：$O(ny^2)$

$$F_{k+1}(y)=\min_{0\le x_{k+1}\le\left\lfloor \frac{y}{v_{k+1}}\right\rfloor}\{F_k(y-v_{k+1}x_{k+1})+w_{k+1}x_{k+1}\}$$

$$F_1(y)=w_1\left\lfloor \frac{y}{v_1}\right\rfloor=w_1 y$$

$$k=2,\ldots,n，\quad y=0,1,\ldots,Y$$

贪心法： $O(n)$

$$G_{k+1}(y)=w_{k+1}\left\lfloor \frac{y}{v_{k+1}}\right\rfloor+G_k(y \bmod v_{k+1})\quad k>1$$

$$G_1(y)=w_1\left\lfloor \frac{y}{v_1}\right\rfloor=w_1 y$$

n=1, 2 贪心法得到最优解

$n = 1$ 只有一种零钱，$F_1(y) = G_1(y)$

$n = 2$, 使用价值大的钱越多(x_2越大)，得到的解越好

$$\boxed{F_2(y) = \min_{0 \le x_2 \le \lfloor y/v_2 \rfloor} \{F_1(y - v_2 x_2) + w_2 x_2\}}$$

$$\begin{aligned}
&[F_1(y - v_2 x_2) + w_2 x_2] \\
&\quad - [F_1(y - v_2(x_2 + \delta)) + w_2(x_2 + \delta)] \\
&= [w_1(y - v_2 x_2) + w_2 x_2] \\
&\quad - [w_1(y - v_2 x_2 - v_2\delta) + w_2 x_2 + w_2 \delta] \\
&= w_1 v_2 \delta - w_2 \delta = \delta(w_1 v_2 - w_2) \ge 0
\end{aligned}$$

所以 $F_2(y) = G_2(y)$

$n>2$得到最优解的判定条件

定理4.5 对每个正整数 k，假设对所有非负整数 y 有$G_k(y) = F_k(y)$，那么 $G_{k+1}(y) \le G_k(y) \Leftrightarrow F_{k+1}(y) = G_{k+1}(y)$.

定理4.6 对每个正整数k，假设对所有非负整数 y 有$G_k(y) = F_k(y)$, 且存在 p 和 δ 满足

$$v_{k+1} = pv_k - \delta,\ 其中 0 \le \delta < v_k，\ v_k < v_{k+1}，\ p 为正整数，$$

则下面的命题等价：

(1) $G_{k+1}(y) = F_{k+1}(y)$ 对一切正整数 y；

(2) $G_{k+1}(pv_k) = F_{k+1}(pv_k)$；

(3) $w_{k+1} + G_k(\delta) \le pw_k$.

条件(3)需 $O(k)$ 时间验证 $G_{k+1}(y)=F_{k+1}(y)$，整个验证时间为 $O(n^2)$

实例

$$v_{k+1} = pv_k - \delta, \quad 0 \le \delta < v_k, \quad p \in Z^+$$
$$w_{k+1} + G_k(\delta) \le pw_k$$

例4.5 $v_1 = 1, v_2 = 5, v_3 = 14, v_4 = 18, w_i = 1, i = 1, 2, 3, 4.$

对一切 y 有 $G_1(y)=F_1(y), G_2(y)=F_2(y)$.

验证 $G_3(y) = F_3(y)$

$v_3 = pv_2 - \delta \Rightarrow p=3, \delta=1.$

$w_3 + G_2(\delta) = 1+1 = 2$

$pw_2 = 3\times 1 = 3$

$w_3 + G_2(\delta) \le p\, w_2$

$v_4 = pv_3 - \delta \Rightarrow p=2, \delta=10$

$w_4 + G_3(\delta) = 1+2 = 3$

$pw_3 = 2\times 1 = 2$

$w_4 + G_3(\delta) > pw_3$

结论：$G_3(y) = F_3(y)$, 对于 $y = pv_3 = 28$，$G_4(y) > F_4(y)$

小结

- 贪心策略不一定得到最优解，在这种情况下可以有两种处理方法:
  (1) 参数化分析：分析参数取什么值 可得到最优解
  (2) 估计贪心法得到的解在最坏情况下与最优解的误差（第10章近似算法）
- 一个参数化分析的例子:找零钱问题

4.4 贪心法的典型应用

4.4.1 最优前缀码

二元前缀码

用**0-1**字符串作为代码表示字符，要求任何字符的代码都不能作为其它字符代码的前缀

非前缀码的例子

***a*: 001, *b*: 00, *c*: 010, *d*: 01**

解码的歧义，例如字符串 **0100001**

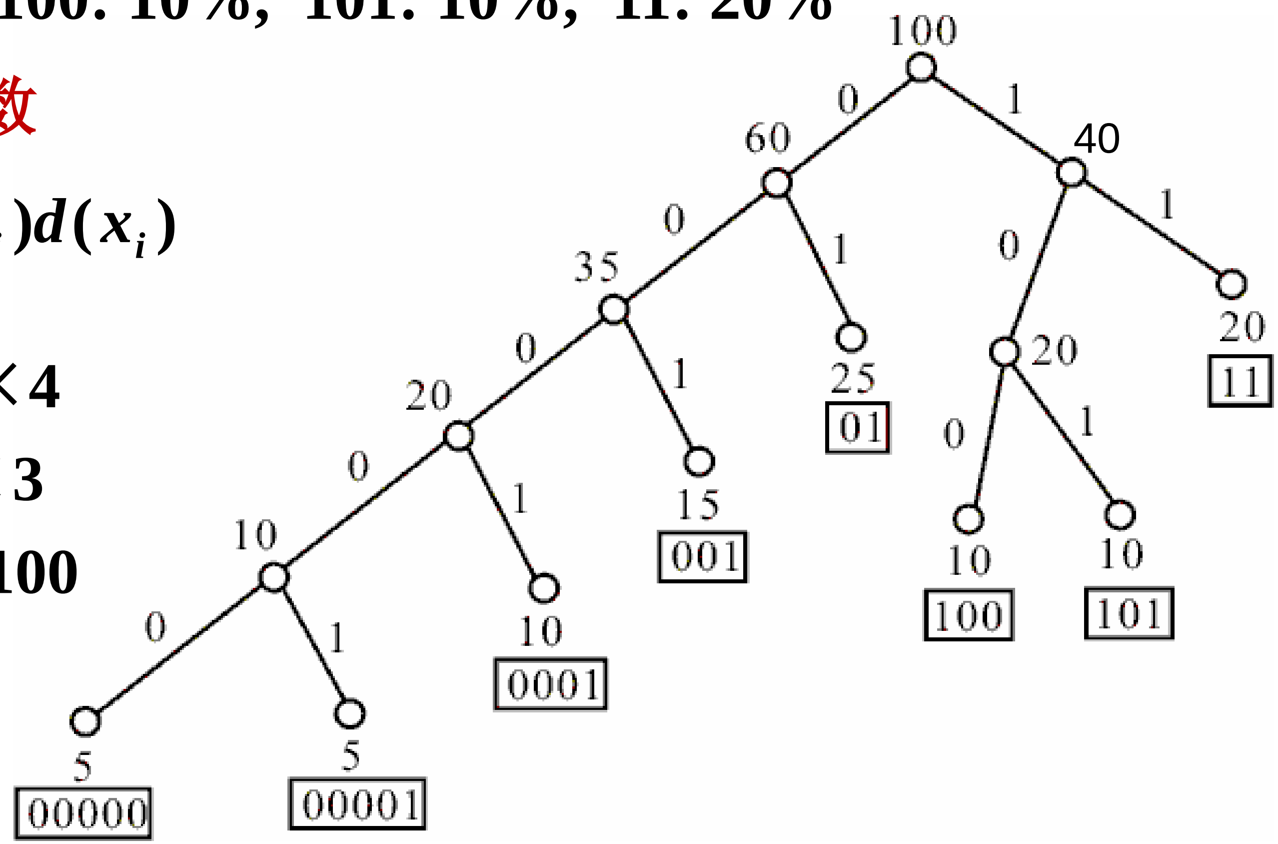
解码**1: 01, 00, 001** ***d, b, a***

解码**2: 010, 00, 01** ***c, b, d***

前缀码的二叉树及权值

前缀码： { 00000, 00001, 0001, 001, 01, 100, 101, 11}

频率： 00000: 5%， 00001: 5%， 0001: 10%， 001: 15%，
01: 25%，100: 10%，101: 10%，11: 20%



平均的二进制位数

$$B = \sum_{i=1}^{n} f(x_i) d(x_i)$$

B=[(5+5)×5+10×4

+(15+10+10)×3

+(25+20)×2]/100

=2.85

最优前缀码

权值B最小

最优前缀码问题

例4.6 最优前缀码 给定字符集 $C=\{x_1, x_2, \ldots, x_n\}$和每个字符的频率$f(x_i)$, i=1,2,…,n, 求关于C 的一个最优前缀码.

算法4.4 Huffman(C)

输入：$C=\{x_1, x_2, \ldots, x_n\}$，$f(x_i)$, i=1, 2, … , n.

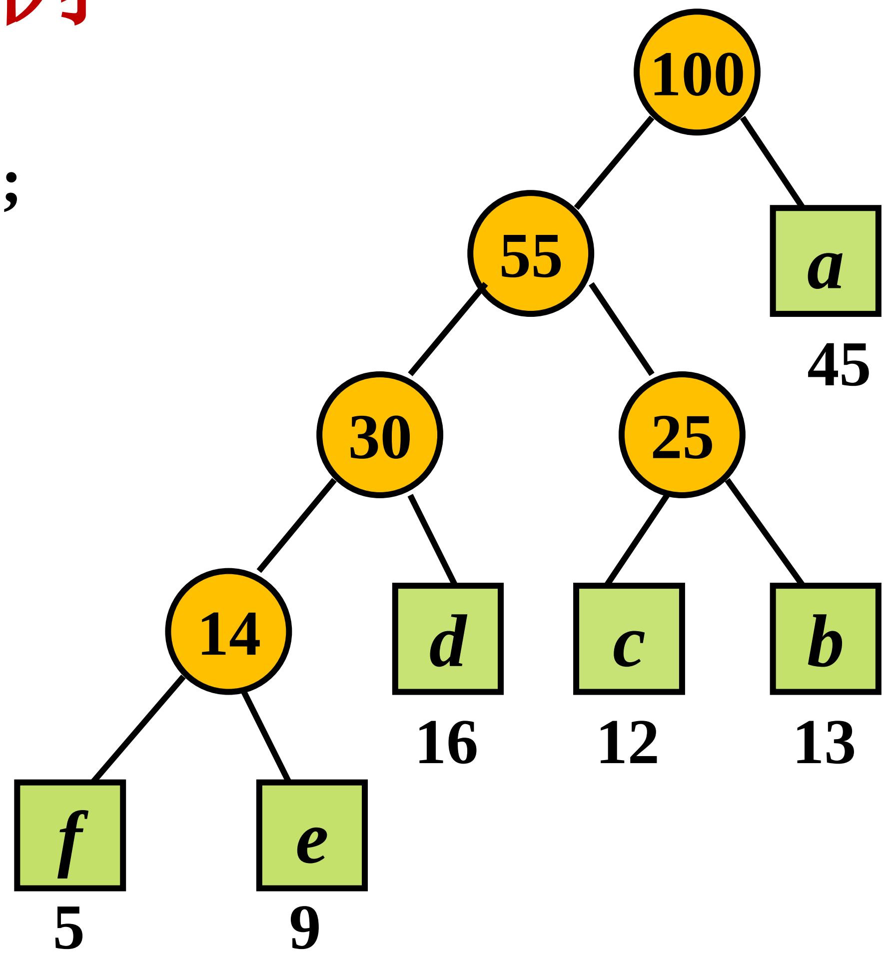
输出：Q //队列

1. $n \leftarrow |C|$
2. $Q \leftarrow C$ //按频率递增构成队列Q
3. for $i \leftarrow 1$ to $n-1$ do
4. $z \leftarrow$Allocate-Node() // 生成结点 z
5. $z.left \leftarrow Q$中最小元 // 取出Q最小元作z的左儿子
6. $z.right \leftarrow Q$中最小元 // 取出Q最小元作z的右儿子
7. $f(z) \leftarrow f(x)+f(y)$
8. Insert(Q,z) // 将 z 插入Q
9. return Q

实例

例如 ***a***: 45, ***b***: 13; ***c***: 12;

d: 16; ***e***: 9; ***f***: 5



编码:

***f*--0000, *e*--0001,**

***d*--001, *c*--010,**

***b*—011, *a*--1**

平均位数:

$$4 \times (0.05 + 0.09) + 3 \times (0.16+0.12+0.13) + 1\times 0.45 = 2.24$$

算法正确性证明:引理4.2

引理4.2 设C是字符集，$\forall c \in C$, $f(c)$为频率，$x, y \in C$, $f(x)$, $f(y)$ 频率最小，那么存在最优二元前缀码使得 x, y 的码字等长，且仅在最后一位不同.

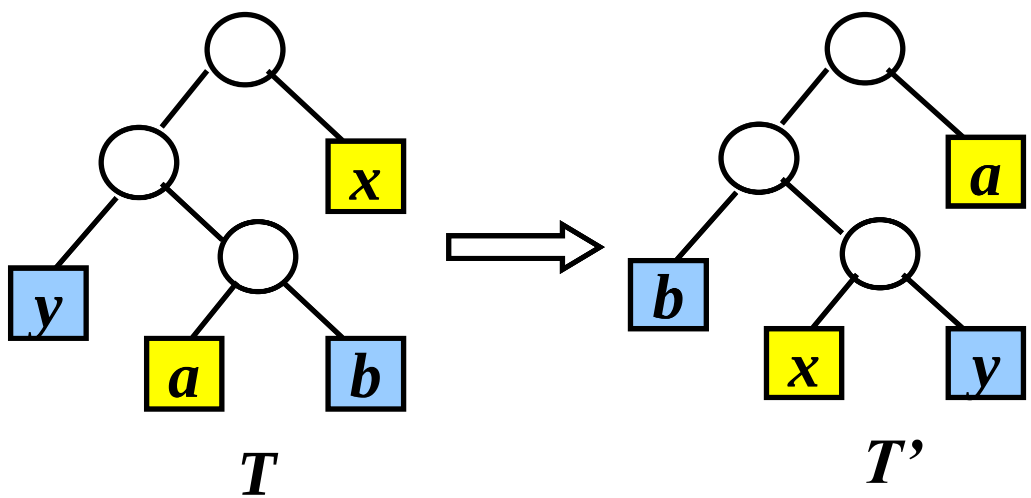
$T \rightarrow T'$

$f[x] \le f[a]$

$f[y] \le f[b]$

a与 x 交换

b与 y 交换



则 T 与 T' 的权之差为

$$B(T) - B(T') = \sum_{i \in C} f[i] d_T(i) - \sum_{i \in C} f[i] d_{T'}(i) \ge 0$$

其中 $d_T(i)$为 i 在T 中的层数（i 到根的距离）

引理4.3

引理4.3 设 T 是二元前缀码所对应的二叉树，$\forall\, x, y \in T$, x, y 是树叶兄弟，z 是 x, y 的父亲，令 $T' = T - \{x, y\}$, 且令 z 的频率 $f(z) = f(x)+f(y)$，T' 是对应于二元前缀码 $C' = (C-\{x, y\})\cup\{z\}$ 的二叉树，那么 $B(T)=B(T')+f(x)+f(y)$.

证 $\forall c\in C-\{x, y\}$,有 $d_T(c) = d_{T'}(c) \Rightarrow f(c)d_T(c) = f(c)d_{T'}(c)$

$$d_T(x) = d_T(y) = d_{T'}(z) +1.$$

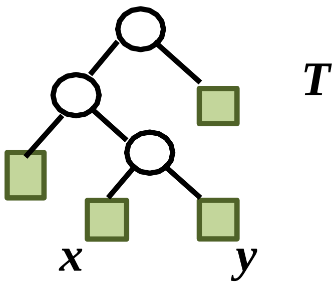
$$B(T) = \sum_{i\in T} f(i)d_T(i)$$

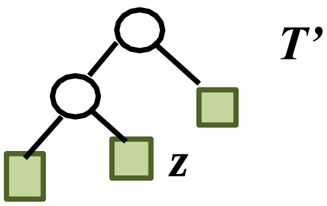
将 $d_T(x) = d_T(y) = d_{T'}(z) +1$，$f(z) = f(x)+f(y)$ 带入

$$= \sum_{i\in T, i\neq x,y} f(i)d_T(i) + f(x)d_T(x) + f(y)d_T(y)$$

$$= \sum_{i\in T', i\neq z} f(i)d_{T'}(i) + f(z)d_{T'}(z) + (f(x) + f(y))$$

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





证明：归纳法

定理4.7 Haffman 算法对任意规模为 n（$n \geq 2$）的字符集C都得到关于C 的最优前缀码的二叉树.

归纳基础 $n=2$，字符集 $C=\{x_1,x_2\}$，Huffman算法得到的代码是 0 和 1，是最优前缀码.

归纳步骤 假设Huffman算法对于规模为k 的字符集都得到最优前缀码. 考虑规模为 $k+1$的字符集 $C=\{x_1, x_2, ..., x_{k+1}\}$, 其中 $x_1, x_2 \in C$是频率最小的两个字符. 令

$$C' = (C-\{x_1, x_2\}) \cup \{ z \}, \quad f(z) = f(x_1)+f(x_2)$$

根据归纳假设，Huffman算法得到一棵关于字符集 C'、频率 $f(z)$ 和 $f(x_i)$（$i = 3, 4, ... , k+1$）的最优前缀码的二叉树T'.

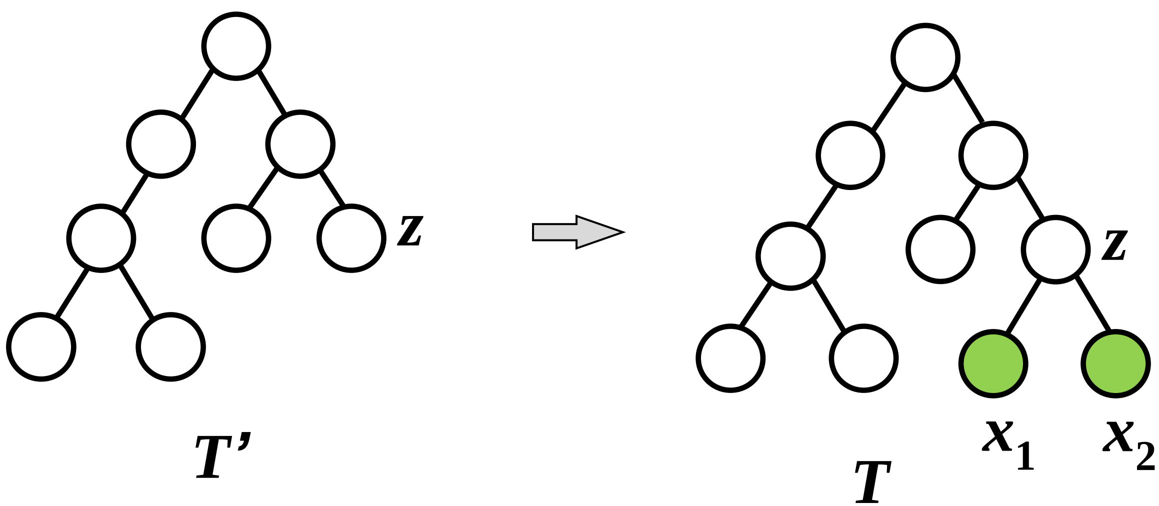
证明：归纳法(续)

把 x_1和 x_2作为 z 的儿子附加到 T'上，得到树T，那么T是关于字符集 $C = (C'-\{z\})\cup\{x_1, x_2\}$ 的最优前缀码的二叉树.

如若不然，存在更优的树 T^*. 根据引理4.2，其最深层树叶是 x_1, x_2，且 $B(T^*)<B(T)$. 去掉 T^*中的 x_1和 x_2，根据引理4.3，所得二叉树 $T^{*\prime}$满足

$$B(T^{*\prime}) = B(T^*) - (f(x_1)+f(x_2)) < B(T) - (f(x_1)+f(x_2)) = B(T')$$

与 T'是一棵关于C'的最优前缀码的二叉树矛盾.



Huffman树应用:文件归并

例4.7 问题：给定一组不同长度的排好序文件构成的集合

$$S = \{ f_1, \dots , f_n\}$$

其中 f_i 表示第 i 个文件含有的项数. 使用二分归并将这些文件归并成一个有序的文件.

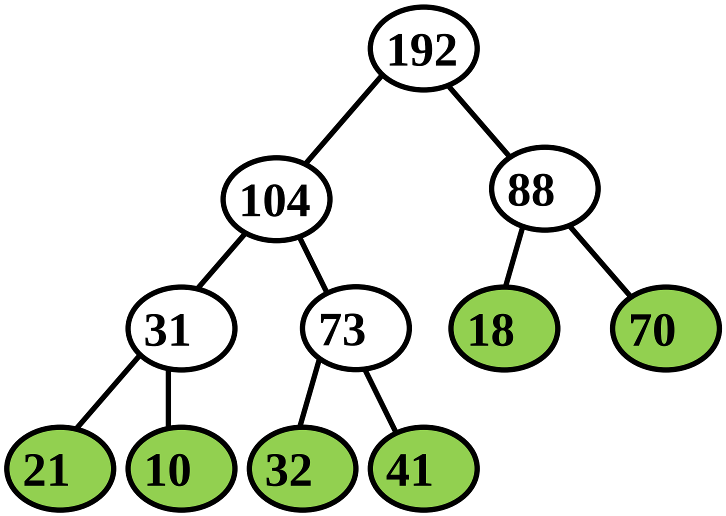
归并过程对应于二叉树：文件为树叶. i与 j归并的文件是它们的父结点.

归并代价(最多的比较次数)：结点 i与 j 归并代价为 $f_i + f_j - 1$.
总的代价：每个文件 (树叶)的深度乘以文件大小之和再减掉归并次数 $n-1$

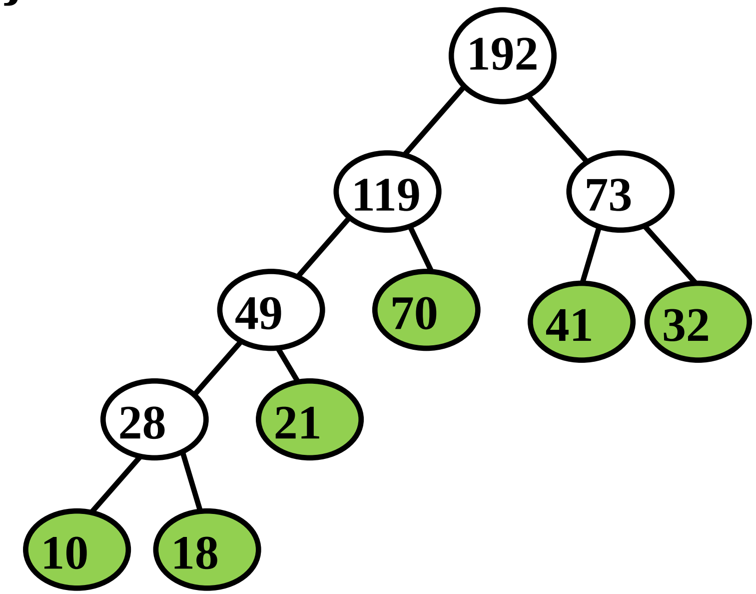
$$\sum_{i\in S} d(i) f_i - (n-1)$$

实例

实例：$S = \{ 21,10,32,41,18,70 \}$



顺序归并



Huffman树归并

代价

顺序归并： $(21+10+32+41)\times 3+(18+70)\times 2-5=483$

Huffman树归并： $(10+18)\times 4+21\times 3+(70+41+32)\times 2-5=456$

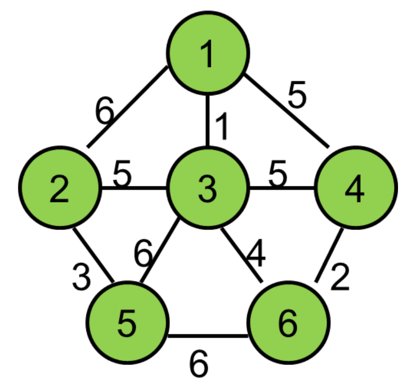
4.4.2 最小生成树

无向连通带权图 $G=(V,E,W)$，$w(e)\in W$是边e的权. G的一棵生成树是包含了G的所有顶点的树, 树中各边的权之和称为树的权，具有最小权的生成树称为 G 的**最小生成树**.

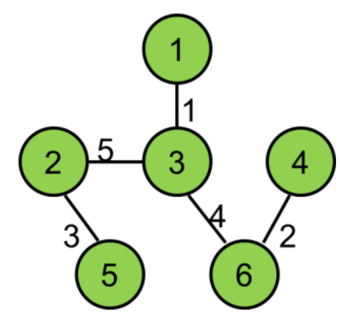
命题4.1 设G是 n 阶连通图，那么

(1) T是G 的生成树当且仅当 T 有$n-1$条边.

(2) 如果T是G的生成树，$e\notin T$，那么$T\cup\{e\}$含有一个圈 (回路).



问题：给定连通带权图G，求G的一棵最小生成树.



算法：**Prim算法** 和 **Kruskal算法**

Prim算法

算法4.5 Prim

输入：连通图 $G = <V, E, W>$

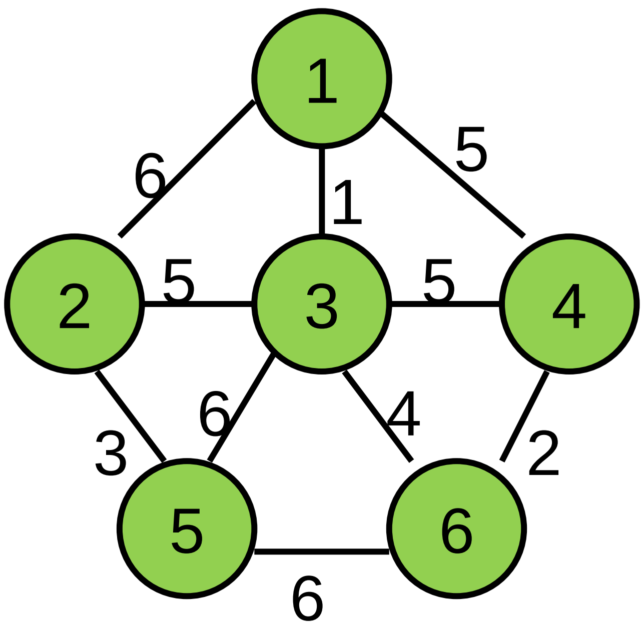
输出：G 的最小生成树 T

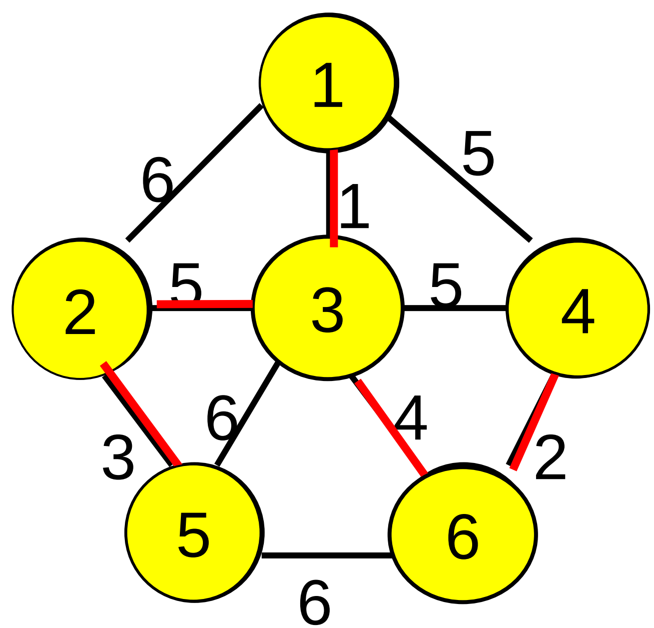
1．$S \leftarrow \{1\}$；$T \leftarrow \varnothing$

2．while $V - S \neq \varnothing$ do

3． 从 $V - S$ 中选择 j 使得 j 到 S 中顶点的边 e 的权最小

4． $S \leftarrow S \cup \{j\}$, $T \leftarrow T \cup \{e\}$





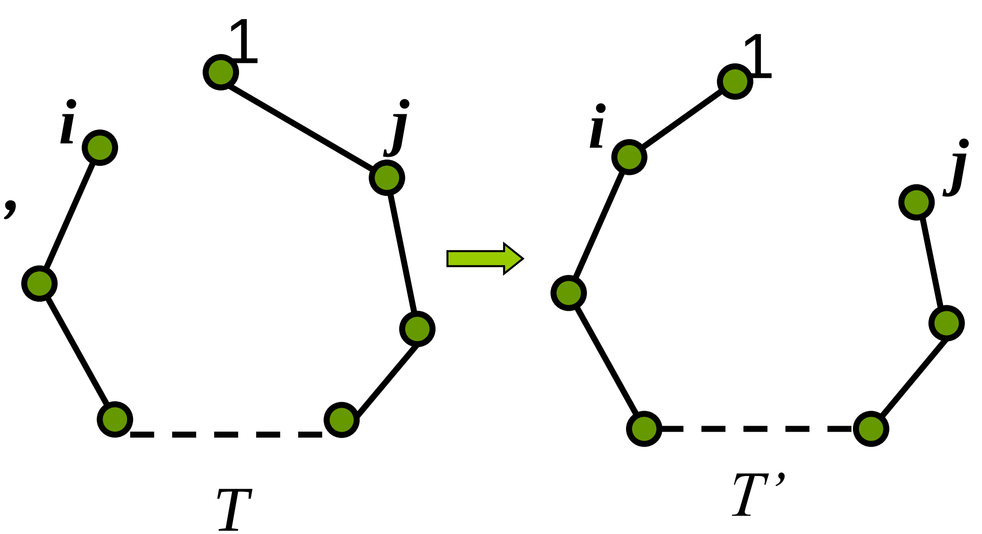
正确性证明

对步数归纳

定理4.8 对于任意 $k < n$, 存在一棵最小生成树包含算法前 k 步选择的边.

归纳基础：$k=1$, 存在一棵最小生成树 T 包含边 $e = (1, i)$, 其中$(1, i)$是所有关联 1 的边中权最小的.

设T 为一棵最小生成树，假设T 不包含$(1, i)$, 则$T\cup\{(1, i)\}$含有一条回路，回路中关联1的另一条边为 $(1, j)$，令 $T' = (T-\{(1, j)\})\cup\{(1, i)\}$，则$T'$也是生成树，且$W(T')\le W(T)$.



正确性证明(续)

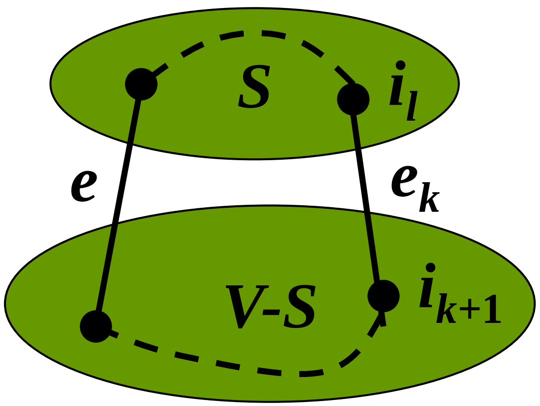
归纳步骤：

假设算法进行了$k-1$步，生成树的边为 $e_1,e_2,\ldots,e_{k-1}$，这些边的 k 个端点构成集合S. 由归纳假设存在G 的一棵最小生成树T 包含这些边.

算法第k 步选择了顶点 i_{k+1}, 则 i_{k+1}到S中顶点的边权最小, 设这条边为 $e_k=(i_{k+1},i_l)$. 假设T不含有e_k，则将 e_k 加到T 中形成一条回路. 这条回路有另外一条连接S与$V-S$中顶点的边e, 令

$$T^*=(T-\{e\})\cup\{e_k\},$$

则T^*是G 的一棵生成树，包含$e_1,e_2,\ldots,e_k$, $W(T^*)\le W(T)$.



算法时间：$T(n)=O(n^2)$

Kruskal算法

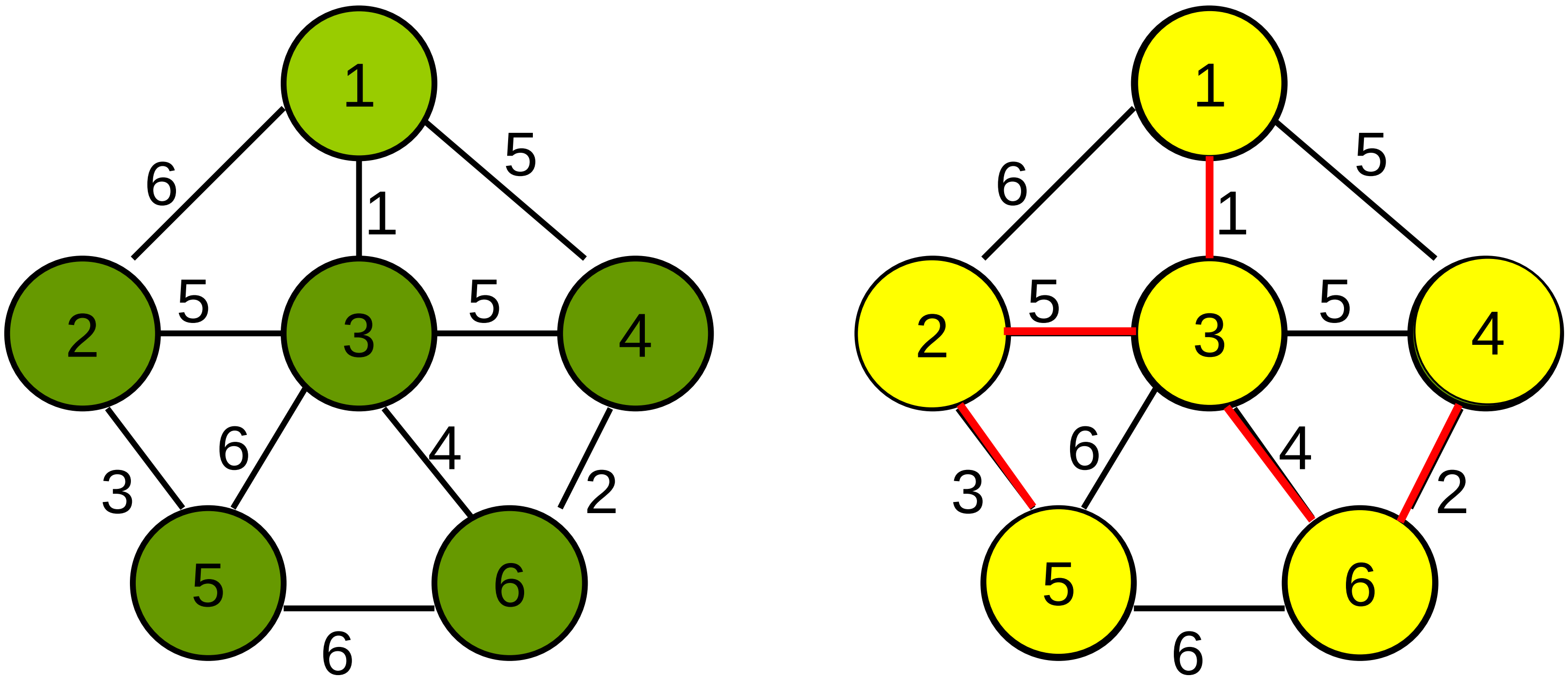
算法4.6 Kruskal

输入：连通图G=<V,E,W>　　　// 顶点数n，边数m

输出：G的最小生成树

1. 按权从小到大排序G中的边，使得$E=\{e_1, e_2, \ldots, e_m\}$
2. $T \leftarrow \varnothing$
3. **repeat**
4. 　　$e \leftarrow E$中的最短边
5. 　　**if** e的两端点不在同一个连通分支
6. 　　**then** $T \leftarrow T \cup \{e\}$
7. 　　$E \leftarrow E - \{e\}$
8. **until** T包含了$n-1$条边

实例

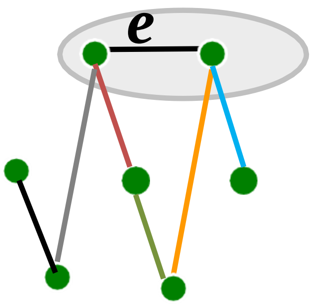


Kruskal算法正确性证明

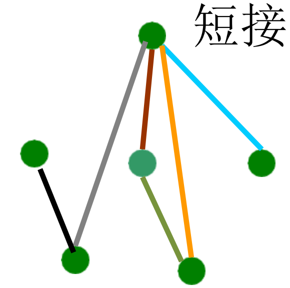
定理4.9 对任意 $n>1$, 算法对 n 阶图得到一棵最小生成树.

证明 $n=2$, 只有一条边，命题显然为真.

假设对于n个顶点的图算法正确，考虑$n+1$个顶点的图G, G中最小权边 $e=(i,j)$，从G 中短接 i 和 j，得到图G'. 根据归纳假设，由算法存在G'的最小生成树T'.令$T=T'\cup\{e\}$, 则T 是关于G 的最小生成树.



否则存在G 的含边e 的最小生成树T^*, $W(T^*)<W(T)$. (如果 $e\notin T^*$, 在T^*中加边e，形成回路. 去掉回路中任意别的边所得生成树的权仍旧最小). 在T^*中短接 e 得到G' 的生成树$T^*-\{e\}$, 且



$W(T^*-\{e\}) = W(T^*)-w(e) < W(T)-w(e) = W(T')$, 与$T'$ 的最优性矛盾.

算法的实现与时间复杂度

引入连通分支标记数组FIND[].

如G的顶点集合V = {1,2,3,4,5,6,7,8}，初始连通分支标记数组FIND=[1,2,3,4,5,6,7,8].

- 若e_1=(2,3)是权最小的边，则算法首先选择该边，2、3两个点连通后标记改为相同，如令Find[3]=2.
- 若当前连通分支为{1,5,7,8} 、{2,3,6}、{4}对应标记数组为FIND=[1,2,2,4,1,2,1,1].需要选择e_2=(5,6)，由于Find[5]$\neq$ FIND [6]，5、6不在同一分支，可以添加该边. 此时，需要修改5或6所在分支集合的标记，将较小分支改为较大分支标记，则此处令FIND [2]= FIND [3]= FIND [6]=1.
- 选择较小分支集合中的顶点标记更新，可以保证对于任一顶点 i , 修改一次Find[i], 则i所在的分支合并后规模至少翻一倍. 那么 i 最多需要 k ($2^k \leq n$)次修改即可完成最小生成树，因此$k \leq \log n$. 所有顶点标记需要修改的次数为$O(n\log n)$.

算法的实现与时间复杂度

数据结构：

建立FIND数组，FIND[i] 是顶点i 的连通分支标记.

(1) 初始FIND[i]=i.

(2) 两个连通分支合并，则将较小分支顶点的FIND值更新为较大分支的标记

时间复杂度：

(1) 每个顶点至多更新log$n$次，建立和更新FIND数组的总时间为$O(n\log n)$

m为边数，n为顶点数.
所以$n-1 \le m \le n(n-1)/2 \le n^2$

(2) 算法时间为

$$\underbrace{O(m\log m)}_{\text{边排序}} + \underbrace{O(n\log n)}_{\text{FIND数组}} + \underbrace{O(m)}_{\text{其他}} = O(m\log m) = O(m\log n)$$

最小生成树算法小结

- 两种算法， **Prim算法**和**Kruskal算法**
  - **Prim**算法时间复杂度是$O(n^2)$
  - **Kruskal**算法时间复杂度是$O(m\log n)$
- 如何选择算法？
  - 如果图中含有较多的边，例如$m=\Theta(n^2)$, 此时**Kruskal**算法时间复杂度是$O(n^2\log n)$，选择**Prim**算法更好
  - 如果是稀疏图，边比较少，如$m=\Theta(n)$, 此时**Kruskal**算法时间复杂度是$O(n\log n)$，效率比**Prim**更高

最小生成树的应用

- 最小生成树常被用在网络路由、聚类分析等算法中

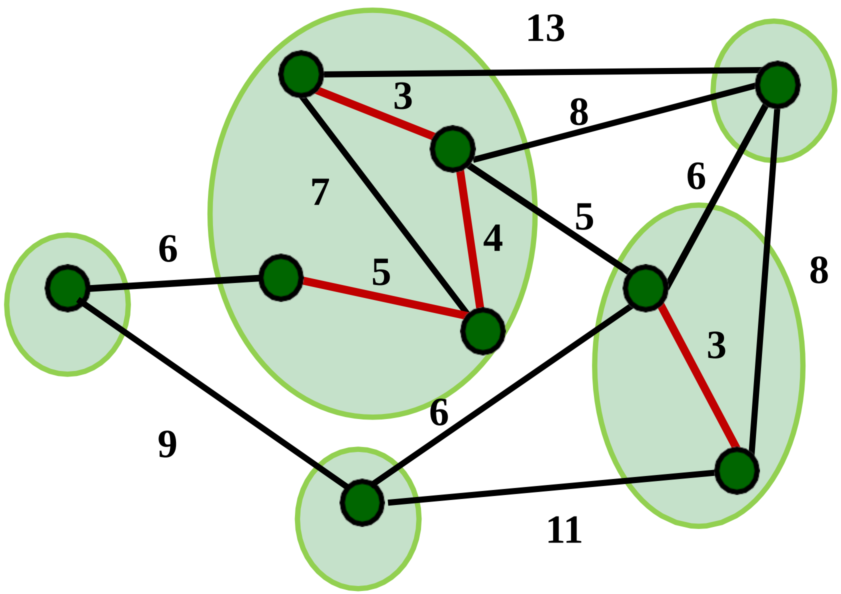
例：设集合$S=\{1,2,\ldots,n\}$, 对$\forall i, j\in S$, $i\neq j$, $d(i, j)=d(j, i)$表示i与j的相似度，假设需要将S划分成k个子集$C_1, C_2,\ldots,C_k$, 聚类$L=\{C_1, C_2,\ldots,C_k\}$的间隔定义为：$D(L)=\min(d\{i, j\}|\ i\in C_t, j\in C_s, 1\leq t < s \leq k)$. 给定$S$和$S$集合中元素的相似度，目标是寻找使得$D(L)$达到最大的$k$聚类$L$.

采用类似于**Kruskal**算法：令S是顶点集，$d(i, j)$表示顶点i到j的边长，得到无向带权图G.

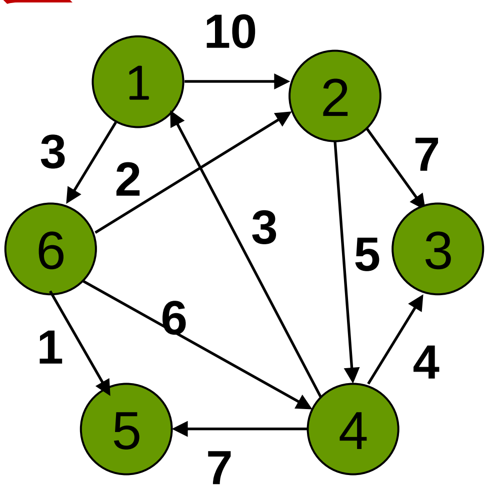
1. 对边长从小到大排序.

2. 依次考察当前最短边e, 如果 e 与已经选中的边不构成回路, 则把 e 加入集合, 否则跳过 e. 计数图的连通分支个数.

3. 直到保留了k个连通分支为止.



4.4.3 单源最短路径



问题：给定带权有向网络 $G=(V, E, W)$, 每条边 $e = \langle i, j \rangle$的权 $w(e)$为非负实数，表示从 i 到 j 的距离. 源点 $s \in V$，求从 s 出发到达其它结点的最短路径.

Dijkstra算法：

$x \in S \Leftrightarrow x \in V$ 且从 s 到 x 的最短路径长度已知,

初始：$S = \{s\}$，$S = V$ 时算法结束

从 s 到 u 相对于S 的最短路径：从 s 到 u且仅经过S 中顶点的最短路径，记为$dist[u]$.

$dist[u]$：从 s 到 u 的相对于S 的最短路径的长度

$short[u]$：从 s 到 u 的最短路径的长

$dist[u] \geq short[u]$

Dijkstra算法

算法4.7 Dijkstra

输入：带权有向图G=<V,E,W>，源点$s \in V$

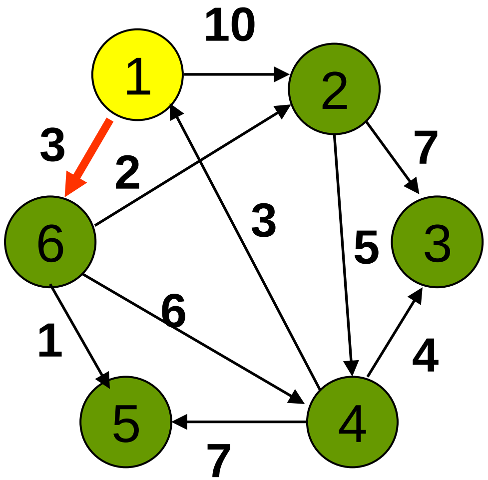
输出：从 s 到每个结点 i 的最短路径

1. $S \leftarrow \{s\}$
2. $dist[s] \leftarrow 0$
3. for $i \in V-\{s\}$ do
4. $\quad dist[i] \leftarrow w(s,i)$ // 如果 s 到 i 没有边, $w(s,i)=\infty$
5. while $V-S \neq \varnothing$ do
6. $\quad$ 从$V-S$ 中取出具有相对S的最短路径的顶点 j
7. $\quad S \leftarrow S \cup \{j\}$;
8. $\quad$ for $i \in V-S$ do
9. $\quad\quad$ if $dist[j] + w(j,i) < dist[i]$
10. $\quad\quad$ then $dist[i] \leftarrow dist[j]+w(j,i)$ // 更新$dist[i]$

实例

输入：*G*=<*V*,*E*,*W*>， 源点 1

V={1, 2, 3, 4, 5, 6}

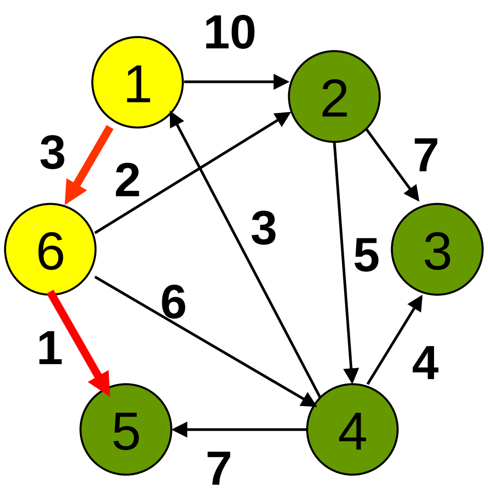


S={1},

dist[1]=0

dist[2]=10, *dist*[6]=3

dist[3]=*dist*[4]=*dist*[5]=∞



S={1,6},

dist[1]=0, *dist*[6]=3

dist[2]=5, *dist*[4]=9, *dist*[5]=4

dist[3]=∞

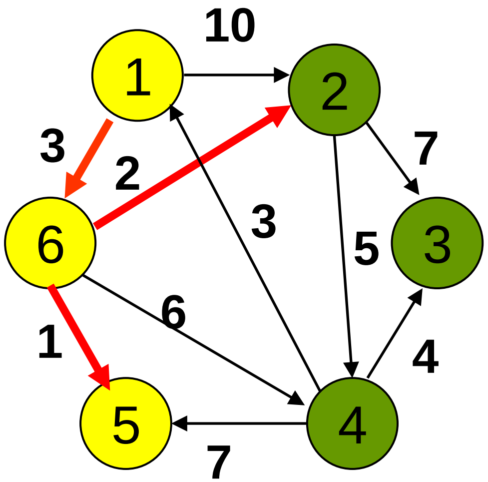
实例（续）

S={1,6,5},

dist[1]=0, *dist*[6]=3, *dist*[5]=4

dist[2]=5, *dist*[4]=9,

dist[3]=∞



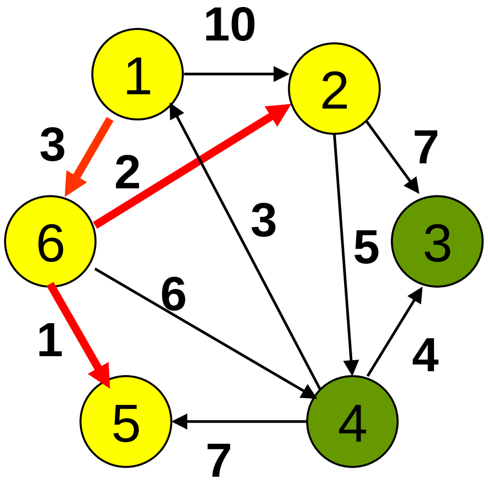
S={1,6,5,2},

dist[1]=0, *dist*[6]=3, *dist*[5]=4

dist[2]=5

dist[3]=12

dist[4]=9



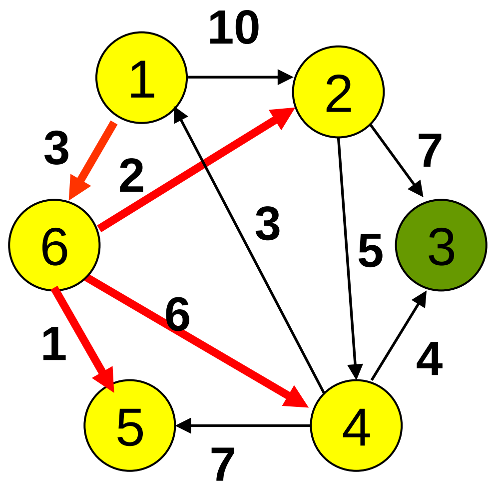
实 例（续）

S={1,6,5,2,4},

dist[1]=0, *dist*[6]=3, *dist*[5]=4

dist[2]=5, *dist*[4]=9,

dist[3]=12

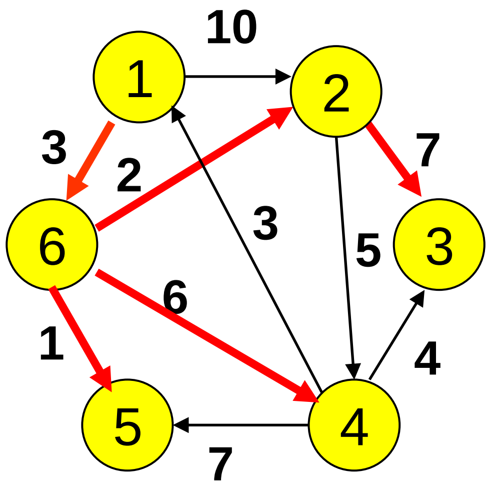


S={1,6,5,2,4,3},

dist[1]=0, *dist*[6]=3, *dist*[5]=4

dist[2]=5, *dist*[4]=9

dist[3]=12



解：

short[1]=0, short[2]=5,

short[3]=12, short[4]=9,

short[5]=4, short[6]=3.

算法正确性证明

定理4.10 当算法进行到第 k 步时，对于S 中每个结点 i，

$$dist[i] = short[i]$$

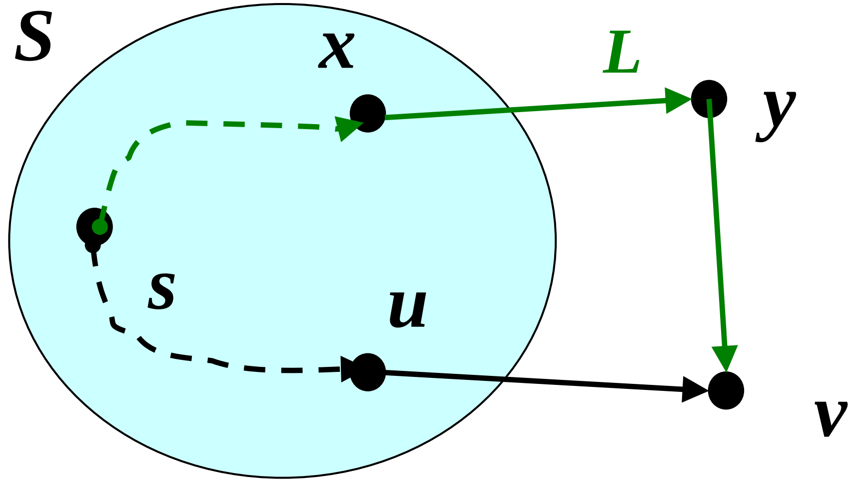
归纳基础 $k=1$, $S=\{s\}$, $dist[s]=short[s]=0$, 命题为真.

归纳步骤 假设命题对于k 为真. 考虑 $k+1$步, 选择顶点v (边 $<u,v>$). 假若存在另一条 s-v 路径 L (绿色)，最后一次出S的顶点为 x, 在这次从S 中出来后经过$V–S$ 的第一个顶点为 y.

$dist[v] \le dist[y]$ //v先被选

$\le dist[y]+d(y,v) = L$

$dist[v]=short[v]$



算法时间复杂度

算法4.7 **Dijkstra**

输入：带权有向图G=<V,E,W>，源点$s\in V$

输出：从 s 到每个结点 i 的最短路径

1. $S\leftarrow\{s\}$
2. $dist[s]\leftarrow 0$
3. for $i\in V-\{s\}$ do
4. $\quad dist[i]\leftarrow w(s,i)$ // 如果 s 到 i 没有边, $w(s,i)=\infty$
5. while $V-S\neq\varnothing$ do
6. $\quad$从$V-S$ 中取出具有相对S的最短路径的顶点j
7. $\quad S\leftarrow S\cup\{j\}$;
8. $\quad$for $i\in V-S$ do
9. $\quad\quad$if $dist[j]+w(j,i)<dist[i]$
10. $\quad\quad$then $dist[i]\leftarrow dist[j]+w(j,i)$ // 更新$dist[i]$

- 时间复杂度：O(n^2)
  - 算法进行n步，每步选一个结点放进S
  - 每步需要修改V-S中节点得$dist$值，每个值需要$O(1)$，每步修改需要$O(n)$复杂度

贪心法小结

(1) 适用于优化问题， 求解过程是多步判断. 判断的依据是局部最优策略，使目标值达到最大(或最小)，与前面的子问题计算结果无关.

(2) 局部最优策略的选择是算法正确性的关键.

(3) 正确性证明方法：数学归纳法、交换论证. 使用数学归纳法主要通过对算法步数或者问题规模进行归纳. 如果要证明贪心策略是错误的，只需举出反例.

(4) 自顶向下求解，通过选择将问题归约为小的子问题.

(5) 如果贪心法得不到最优解，可以对问题的输入进行分析或者估计算法的近似比.

(6) 如果对原始数据排序之后，贪心法往往是一轮处理，时间复杂度和空间复杂度低.