数学知识与数据结构

算法设计与分析

武汉大学
国家网络安全学院
李雨晴

课时安排

- 算法基础 5学时
- 数学基础与数据结构 3学时
- 递归与分治 7 学时
- 动态规划 8学时
- 贪心算法 6学时
- 图的遍历 6学时
- 回溯与分支界限 6学时
- NP完全 3 学时

2 数学基础与数据结构

- 数学基础
- 证明方法
- 递归方程求解
- Master定理(*)
- 基本数据结构 (*)

数学基础

- 2.1 集合，关系和函数（自学）
- 2.5 阶乘和二项式系数（自学）
- 2.7 和式（自学）

数学基础：集合

集合：数学中最基本的概念，没有严格的定义理解成某些个体组成的整体，常用大写字母 A,B,C 等表示

元素：集合中的个体，通常用小写字母a,b,c等表示

例如：

(1) 全体中国人可组成一个集合，每一个中国人均是这个集合的元素

(2) 所有正整数组成一个集合，每一个正整数均是这个集合的元素

数学基础：集合

$x \in A$(x属于A): x是A的元素

$x \notin A$(x不属于A): x不是A的元素

无穷集: 元素个数无限的集合

有穷集(有限集):元素个数有限的集合.

$|A|$: A中元素个数

k元集: k个元素的集合, $k \geq 0$

数学基础：集合

列举法: 列出集合中的全体元素，元素之间用逗号分开，然后用花括号括起来

如 $A=\{ a, b, c, d \}$, $N=\{0,1,2,\ldots\}$

描述法： 用谓词***P(x)***表示x具有性质P，用 $\{x \mid P(x) \}$ 表示具有性质***P***的所有元素组成的集合

如$N=\{ x \mid x$是自然数 $\}$

说明:

(1) 集合中的元素各不相同. 如, $\{1,2,3\}=\{1,1,2,3\}$

(2) 集合中的元素没有次序. 如, $\{1,2,3\}=\{3,1,2\}=\{1,3,1,2,2\}$

(3) 有时两种方法都适用, 可根据需要选用.

数学基础：集合

集合的包含和相等是集合间的两个基本关系

包含(子集)	$A \subseteq B \Leftrightarrow \forall x\,(x \in A \rightarrow x \in B)$
不包含	$A \nsubseteq B \Leftrightarrow \exists x\,(x \in A \wedge x \notin B)$
相等	$A = B \Leftrightarrow A \subseteq B \wedge B \subseteq A$
不相等	$A \neq B \Leftrightarrow A \nsubseteq B \vee B \nsubseteq A$
真包含(真子集)	$A \subset B \Leftrightarrow A \subseteq B \wedge A \neq B$

数学基础：集合

空集$\varnothing$: 不含任何元素的集合

例如, $\{x \mid x^2(0 \wedge x \in \boldsymbol{R}\} = \varnothing$

定理1.1　空集是任何集合的子集

证 用归谬法. 假设不然, 则存在集合$\boldsymbol{A}$, 使得$\varnothing \not\subseteq \boldsymbol{A}$,

即存在x, $x \in \varnothing$且$x \notin \boldsymbol{A}$, 矛盾.

推论　空集是惟一的.

证 假设存在$\varnothing_1$和$\varnothing_2$，则$\varnothing_1 \subseteq \varnothing_2$且$\varnothing_2 \subseteq \varnothing_1$，因此$\varnothing_1 = \varnothing_2$

全集$\boldsymbol{E}$:限定所讨论的集合都是$\boldsymbol{E}$的子集.　相对

数学基础：集合

幂集： A的所有子集组成的集合，即$P(A) = \{ x \mid x \subseteq A \}$

例：

设A={a}

则0个元素的子集：$\varnothing$

1个元素的子集：{a}

因此 $P(A)=\{\varnothing,\{a\}\}$

设B={a, b}

则0个元素的子集：$\varnothing$

1个元素的子集：{a},{b}

2个元素的子集：{a,b}

因此 $P(B)=\{\varnothing,\{a\},\{b\},\{a,b\}\}$

定理1.2 如果 $|A| = n$， 则 $|P(A)| = 2^n$

证 $$|P(A)| = C_n^0 + C_n^1 + \cdots + C_n^n$$

$$= (1+1)^n = 2^n$$

数学基础：集合

集合运算：

并 $A\cup B = \{\, x \mid x\in A \vee x\in B \,\}$

交 $A\cap B = \{\, x \mid x\in A \wedge x\in B \,\}$

相对补 $A-B = \{\, x \mid x\in A \wedge x\notin B \,\}$

对称差 $A\oplus B = (A-B)\cup(B-A) = (A\cup B)-(A\cap B)$

绝对补 $\sim A = E-A = \{\, x \mid x\notin A \,\}$

例如 设$E=\{0,1, \ldots ,9\}$, $A=\{0,1,2,3\}$, $B=\{1,3,5,7,9\}$, 则

$A\cup B =\{0,1,2,3,5,7,9\}$, $A\cap B =\{1,3\}$, $A-B =\{0,2\}$,

$A\oplus B =\{0,2,5,7,9\}$, $\sim A =\{4,5,6,7,8,9\}$, $\sim B =\{0,2,4,6,8\}$

说明:1. 只使用圆括号；2. 运算顺序: 优先级分别为(1)括号, (2)~和幂集, (3)其他. 同级别的按从左到右运算

数学基础：关系

有序对： 由两个元素，如x和y，按照一定的顺序组成的二元组称为有序对，记作 (x,y)

例如：点的直角坐标 (3,–4)

有序对的性质：

有序性 $(x,y)\neq(y,x)$ （当$x\neq y$时）例如：$(0,1)\neq(1,0)$

$(x,y)=(u,v)$的充要条件是$x=u \wedge y=v$

例1 $(2,x+5)=(3y-4,y)$，求 x, y.

解 $3y-4=2,\ x+5=y \Rightarrow y=2,\ x=-3$

数学基础：关系

笛卡尔积：设A, B为集合，A与B 的笛卡儿积记作$A\times B$,

$$A\times B = \{ (x,y) \mid x\in A \wedge y\in B \}.$$

一般来讲，$A\times B \neq B\times A$, 如平面直角坐标系就是笛卡尔积 $\mathbb{R}\times\mathbb{R}$

$$A_1\times A_2 \times\ldots \times A_n= \{ (a_1, a_2 ,\ldots , a_n) \mid a_i \in A_i ,1\leq i\leq n\}.$$

笛卡尔积$A\times A$ 通常记作 $A^2=\{(x,y) \mid x\in A\wedge y\in A\}$

例： $A=\{0, 1\}$, $B=\{a, b, c\}$

$A\times B=\{(0,a),(0,b),(0,c),(1,a),(1,b),(1,c)\}$

$B\times A =\{(a,0),(b,0),(c,0),(a,1),(b,1),(c,1)\}$

$A^2=A\times A=\{(0,0),(0,1),(1,0),(1,1)\}$

$B^2=B\times B=\{(a,a),(a,b),(a,c),(b,a),(b,b),(b,c),(c,a),(c,b),(c,c)\}$

定理：对于有穷集合A和B，若$|A|=m$, $|B|=n$, 则 $|A\times B|=mn$

数学基础：关系

关系：一个基本而且普遍的概念，设A, B为非空集合，$A\times B$的子集R称为A到B的一个二元关系，即$R\subseteq A\times B$

R的定义域定义为$Dom(R)=\{a\mid 对某个b\in B, (a,b)\in R\}$，

值域定义为$Ran(R)=\{b\mid 对某个a\in A, (a,b)\in R\}$.

若$R\subseteq A\times B$, 当$(a,b)\in R$，称a与b具有关系R；

若$(a,b)\notin R$，称a与b不具有关系R；

若$A=B$，称R为A上的一个二元关系;

例： $A=\{1,2,3,4\}$,

$R=\{(1,1),(1,2),(1,4),(2,1),(3,2),(3,4)\}$

数学基础：函数

函数：设f是从A到B的一个二元关系，且对于任一$x \in A$, 都有唯一的$y \in B$，使得$(x,y) \in f$, 称f为A到B的函数, y是f在x上的值或像。

例：$A=\{a, b, c\}$，$B=\{1,2,3,4,5\}$,
$f=\{(a,1),(b,3),(c,5)\}$

2.2 证明方法

- 直接证明
- 间接证明
- 反证法
- 数学归纳法*

2.2.1 直接证明

- 证明P→Q,假设P是真，从P推出Q为真

例 2.5 要证明断言：如果 n 是偶数，则 n^2 也是偶数。该命题的直接证明如下：由于 n 是偶数，有 $n=2k$，k 是某个整数，所以有 $n^2=4k^2=2(2k^2)$，这就得出 n^2 是偶数的结论。

2.2.2 间接证明

- P→Q 等价于¬Q-> ¬P

例 2.6 考虑断言：如果 n^2 是偶数，那么 n 是偶数。如果我们用直接证明技术来证明这个定理，可以像在例 2.5 中的证明那样做。换一种更加简单的方法，证明逻辑等价的断言：如果 n 是奇数，那么 n^2 也是奇数。我们用以下直接证明的方法来证明该命题为真：如果 n 是奇数，那么 $n = 2k + 1$，k 是某个整数，那么，$n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$，所以 n^2 是奇数。

2.2.3 反证法证明

- P→Q，先假设P为真，Q为假，导出矛盾，“Q为假”必定为错。

例 2.7 证明断言：有无限多的素数。用反证法来证明这个命题如下：假设相反，仅存在 k 个素数 $p_1, p_2, \cdots, p_k$，这里 $p_1 = 2$，$p_2 = 3$，$p_3 = 5$，等等，所有其他大于 1 的整数都是合数。令 $n = p_1 p_2 \cdots p_k + 1$，令 p 为 n 的一个素数因子(注意由前面的假设，由于 n 大于 p_k，所以 n 不是素数)。既然 n 不是素数，那么 $p_1, p_2, \cdots, p_k$ 中，必定有一个能够整除 n，就是说，p 是 $p_1, p_2, \cdots, p_k$ 中的一个，因为 p 整除 $p_1 p_2 \cdots p_k$，因此，p 整除 $n - p_1 p_2 \cdots p_k$，但是 $n - p_1 p_2 \cdots p_k = 1$，由素数的定义可知，因为 p 大于 1，所以 p 不能整除 1。这是一个矛盾，于是得到，素数的个数是无限的。

2.2.5 数学归纳法

- 证明某一性质$P(n)$对于 $n = n_0, n_0 + 1, n_0 + 2 \ldots..$为真
  - 基础步：证明该性质对于n_0成立
  - 归纳步：假设该性质对于 $n_0, n_0 + 1, \ldots . n - 1$成立，证明对于$n$成立。

2.2.5 数学归纳法

例2.10 证明Bernoulli不等式：对每一个实数$x \geqslant -1$和每一个自然数n，有$(1+x)^n \geqslant 1+nx$。

基础步： 如果 $n=1$，那么 $1+x \geqslant 1+x$。

归纳步： 假定不等式对于所有的 k 成立，$1 \leqslant k < n, n > 1$，那么

$$
\begin{aligned}
(1+x)^n &= (1+x)(1+x)^{n-1} \\
&\geqslant (1+x)(1+(n-1)x) && \{\text{用归纳假设，且 } x \geqslant -1\} \\
&= (1+x)(1+nx-x) \\
&= 1+nx-x+x+nx^2-x^2 \\
&= 1+nx+(nx^2-x^2) \\
&\geqslant 1+nx && \{\text{因为对于 } n \geqslant 1, (nx^2-x^2) \geqslant 0\}
\end{aligned}
$$

因此，对于所有的 $n \geqslant 1$，有$(1+x)^n \geqslant 1+nx$。

2.8 递归方程求解

- *常系数线性同质递归方程 (linear homogeneous with constant coefficients)
- 几类特殊的非同质递归方程求解

2. 8. 1常系数线性同质递归方程

$$T(n) = a_1 T(n-1) + a_2 T(n-2) + \cdots + a_k T(n-k)$$

称为k 阶常系数线性同质递归方程。

其特征方程(characteristic equation)为：

$$x^k = a_1 x^{k-1} + a_2 x^{k-2} + \cdots + a_k \Rightarrow x^k - a_1 x^{k-1} - a_2 x^{k-2} - \cdots - a_k = 0$$

我们仅仅关注一阶和二阶同质方程，因为对于使用此类递归方程来分析算法，往往是一阶或二阶的。对于一阶情形，求解过程非常直观(直接递推)：

$$T(n) = aT(n-1) = a^2 T(n-2) = \cdots = a^n T(0)$$

2.8.1常系数线性同质递归方程

对于二阶情形，特征方程变为：$x^2 - a_1 x - a_2 = 0$

设该二次方程(quadratic equation)的两个根为：x_1, x_2

那么 $T(n)$ 可以表示为:

$$T(n) = \begin{cases} c_1 \cdot x_1^{\ n} + c_2 \cdot x_2^{\ n} & , if \quad x_1 \neq x_2 \\ c_1 \cdot r^n + c_2 \cdot n \cdot r^n & , if \quad x_1 = x_2 = r \end{cases}$$

然后利用初始值 $T(n_0)$ 及 $T(n_0+1)$,使用待定系数法解出系数即可

例:

已知 $T(n)=3T(n-1)+4T(n-2)$ 并且 $T(0)=1, T(1)=4$

解：特征方程为 $x^2-3x-4=0 \Rightarrow x_1=-1, x_2=4$

$$\therefore T(n)=c_1(-1)^n+c_2 4^n \Rightarrow \begin{cases} T(0)=1=c_1+c_2 \\ T(1)=4=-c_1+4c_2 \end{cases}$$

$$\Rightarrow c_1=0, c_2=1 \Rightarrow T(n)=4^n$$

例:

- 费氏数列是由0，1开始，之后的每一项等于前两项之和：

  0，1，1，2，3，5，8，13，21，34，55，89，144……

递归形式的算法：

```
procedure Fib(n)
    if n=1 or n=2 then return 1
    else return Fib(n-1)+Fib(n-2)
```

例:

已知 $T(n)=T(n-1)+T(n-2)$ 并且 $T(1)=T(2)=1$

解：特征方程为 $x^2-x-1=0 \Rightarrow x_1=\frac{1+\sqrt{5}}{2}, x_2=\frac{1-\sqrt{5}}{2}$

$$\therefore T(n)=c_1\left(\frac{1+\sqrt{5}}{2}\right)^n+c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$$

$$\Rightarrow c_1=\frac{1}{\sqrt{5}}, c_2=-\frac{1}{\sqrt{5}}$$

$$\Rightarrow T(n)=\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n-\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

2.8.2非同质递归方程的求解

$$T(n)=T(n-1)+g(n), n\geq 1 \Rightarrow T(n)=T(0)+\sum_{i=1}^{n} g(i)$$

$$T(n)=g(n)T(n-1), n\geq 1 \Rightarrow T(n)=g(n)g(n-1)\cdots g(1)T(0)$$

$$T(n)=g(n)T(n-1)+h(n), n\geq 1 \Rightarrow ?$$

2.8.2非同质递归方程的求解

$T(n)=g(n)T(n-1)+h(n), n\geq 1$ 且$T(0)$已知。

解：令 $k(n)=\begin{cases}\dfrac{T(n)}{g(n)g(n-1)\cdots g(1)}, n\geq 1\\ T(0), n=0\end{cases}$

$k(n)=\dfrac{T(n)}{g(n)g(n-1)\cdots g(1)}, n\geq 1 \Rightarrow T(n)=g(n)g(n-1)\cdots g(1)k(n)$

$\Rightarrow T(n-1)=g(n-1)g(n-2)\cdots g(1)k(n-1)$

$\Rightarrow \underbrace{g(n)g(n-1)\cdots g(1)k(n)}_{T(n)}=g(n)\underbrace{g(n-1)\cdots g(1)k(n-1)}_{T(n-1)}+h(n)$

$\Rightarrow k(n)=k(n-1)+\dfrac{h(n)}{g(n)\cdots g(1)}$

2.8.2非同质递归方程的求解

$$\Rightarrow \quad k(n)=k(n-2)+\frac{h(n-1)}{g(n-1)\cdots g(1)}+\frac{h(n)}{g(n)\cdots g(1)}$$

$$\Rightarrow \quad k(n)=k(0)+\sum_{i=1}^{n}\frac{h(i)}{g(i)g(i-1)\cdots g(1)}=T(0)+\sum_{i=1}^{n}\frac{h(i)}{g(i)g(i-1)\cdots g(1)}$$

$$\Rightarrow \quad T(n)=g(n)g(n-1)\cdots g(1)\left(T(0)+\sum_{i=1}^{n}\frac{h(i)}{g(i)g(i-1)\cdots g(1)}\right)$$

2.8.2非同质递归方程的求解

$$\boxed{T(n)=g(n)g(n-1)\cdots g(1)\left(T(0)+\sum_{i=1}^{n}\frac{h(i)}{g(i)g(i-1)\cdots g(1)}\right)}$$

例： $T(n)=nT(n-1)+n!, T(0)=0$

解：这里 $g(n)=n, h(n)=n!$

$$\therefore T(n)=n(n-1)\cdots 1(0+\sum_{i=1}^{n}\frac{i!}{i!})=n!n$$

2.8.3分治递推关系的解

- 分治算法中

$$f(n)=\begin{cases} d & 若\ n \leqslant n_0 \\ a_1 f(n/c_1)+a_2 f(n/c_2)+\cdots+a_p f(n/c_p)+g(n) & 若\ n > n_0 \end{cases}$$

- 展开递推式
- 代入法：猜想一个解，尝试用数学归纳法来证明

Master Theorem

设$a\geq 1$，$b>1$为常数。$s(n)$为一给定的函数，$T(n)$递归定义如下：

$$T(n)=a\cdot T(n/b)+s(n)$$

并且$T(n)$有适当的初始值。那么，当n充分大时，有：

(1) 若存在$\varepsilon>0$，使得$s(n)=\Omega(n^{\log_b^a+\varepsilon})$ 成立，并且存在$c<1$，使得 $a\cdot s(n/b)\leq c\cdot s(n)$，那么有$T(n)=\Theta(s(n))$

(2) 若$s(n)=\Theta(n^{\log_b^a})$，那么$T(n)=\Theta(n^{\log_b^a}\cdot\log n)$

(3) 若存在$\varepsilon>0$，使得$s(n)=O(n^{\log_b^a-\varepsilon})$ 成立，那么有$T(n)=\Theta(n^{\log_b^a})$

几个例子

$$T(n) = 3T(n/4) + n\log n$$

$$\because a = 3, b = 4, \log b^{a} \approx 0.793, s(n) = n\log n$$

$$\therefore \exists \varepsilon = 0.21, s(n) = n\log n = \Omega(n^{0.79+0.21})$$

$$\therefore T(n) = \Theta(n\log n)$$

几个例子

$$T(n) = T(2n/3) + 1$$

$$\because a = 1, b = \tfrac{3}{2}, s(n) = 1 \quad \therefore \log_b^a = 0$$

$$s(n) = \Theta(n^{\log_b^a}) \qquad \therefore T(n) = \Theta(\log n)$$

$$T(n) = 9T(n/3) + n$$

$$\because a = 9, b = 3, s(n) = n \quad \therefore \log_b^a = 2 \quad \therefore \exists \varepsilon = 1$$

$$s(n) = O(n^{\log_b^a - \varepsilon}) \qquad \therefore T(n) = \Theta(n^2)$$

Master Theorem

- https://www.youtube.com/watch?v=2H0GKdrIowU

Theorem 5.1 *Let a be an integer greater than or equal to 1 and b be a real number greater than 1. Let c be a positive real number and d a nonnegative real number. Given a recurrence of the form*

$$T(n) = \begin{cases} aT(n/b) + n^c & \text{if } n > 1 \\ d & \text{if } n = 1 \end{cases}$$

then for n a power of b,

1. *if $\log_b a < c$, $T(n) = \Theta(n^c)$,*
2. *if $\log_b a = c$, $T(n) = \Theta(n^c \log n)$,*
3. *if $\log_b a > c$, $T(n) = \Theta(n^{\log_b a})$.*

练习

有如下递归式：$T(n) = 2T\left(\frac{n}{4}\right) + 1$，其时间复杂度为：

A. $\Theta(n^2)$

B. $\Theta(n^{1/2\log n})$

C. $\Theta(n)$

D. $\Theta(n^{\log_4 2})$

练习

有如下递归式: $T(n) = 4T\left(\frac{n}{2}\right) + n$ ，其时间复杂度为:

A. $\Theta(n^{\log_4 2})$

B. $\Theta(n^{1/2\log n})$

C. $\Theta(n)$

D. $\Theta(n^2)$

练习

有如下递归式: $T(n) = 4T\left(\frac{n}{2}\right) + n^2$, 其时间复杂度为:

A. $\Theta(n^{\log_4 2})$

B. $\boldsymbol{\Theta(n^2 \log n)}$

C. $\Theta(n)$

D. $\Theta(n^2)$

作业

- P64-65
  - 2.12, 2.19 (a), 2.20 (c),
  - 2.21, 2.23：用主定理求解递推式

数据结构

- 算法的实现离不开数据结构。选择一个合适的数据结构对设计一个有效的算法有十分重要的影响。结构化程序设计创始人Niklaus Wirth(瑞士苏黎士高工)提出一个著名的论断：“程序=算法+数据结构”。1984年，Wirth因开发了Euler、Pascal等一系列崭新的计算语言而荣获图灵奖, 有“结构化程序设计之父”之美誉。
- 本章我们将回顾几种重要的数据结构，包括二叉树、堆、不相交集。

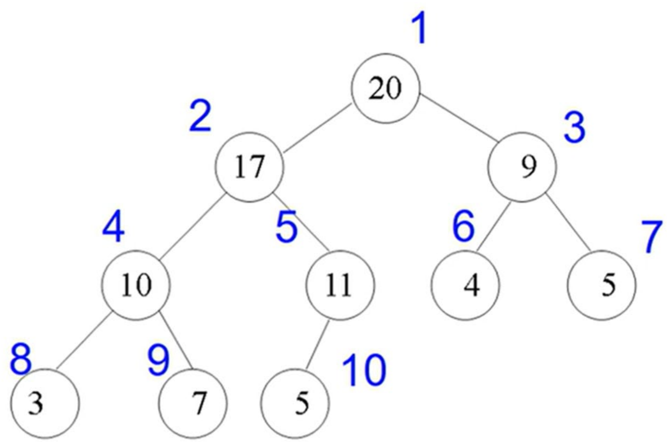
4.2 堆(Heap)

- 在许多算法中，需要大量用到如下两种操作：插入元素和寻找最大(小)值元素。为了提高这两种运算的效率，必须使用恰当的数据结构。
  - 普通队列：易插入元素，但求最大(小)值元素需要搜索整个队列。
  - 排序数组：易找到最大(小)值，但插入元素需要移动大量元素。
  - 堆则是一种有效实现上述两种运算的简单数据结构。

4.2 堆(Heap)

- 定义：堆是一个几乎完全二叉树，每个节点都满足这样的特性：任一父节点的键值(key)不小于子节点的键值(最大堆)
- 沿着每条从根到叶子的路径，元素键值以非升序排列
- 可以类似地定义最小堆，这里以最大堆为准进行分析

4. 2 堆(Heap)

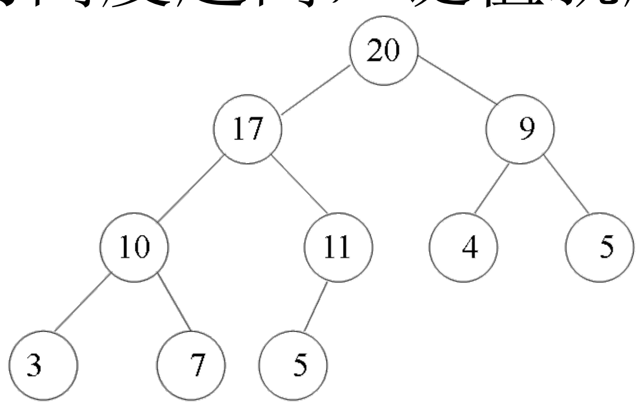


20	17	9	10	11	4	5	3	7	5
1	2	3	4	5	6	7	8	9	10

- 有n个节点的堆T,可以用一个数组$H[1\ldots n]$用下面的方式来表示
  - T的根节点存储在$H[1]$中
  - 假设T的节点x存储在$H[j]$中，那么，它的左右子节点分别存放在$H[2j]$及$H[2j+1]$中(如果有的话)
  - $H[j]$的父节点如果不是根节点，则存储在$H[\lfloor j/2 \rfloor]$中

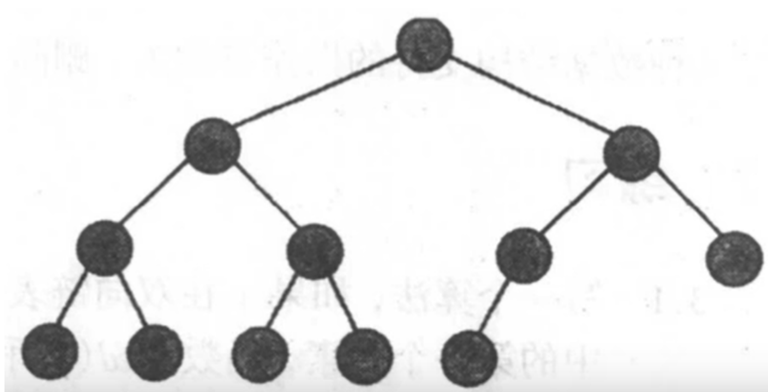
4. 2 堆(Heap)

- 观察结论：
  - 根节点键值最大，叶子节点键值较小。从根到叶子，键值以非升序排列。
  - 节点的左右子节点键值并无顺序要求。
  - 堆的数组表示呈“基本有序”状态。相应地，并非节点的高度越高，键值就越大。



4.2 堆(Heap)

- 假设$H[1...n]$是一个最大堆，那么
  - 第二大元素可能存在哪些位置？
  - 第三大元素可能存在哪些位置？
  - 最小元素可能存在哪些位置？
  - 求最小元素的时间复杂度是？



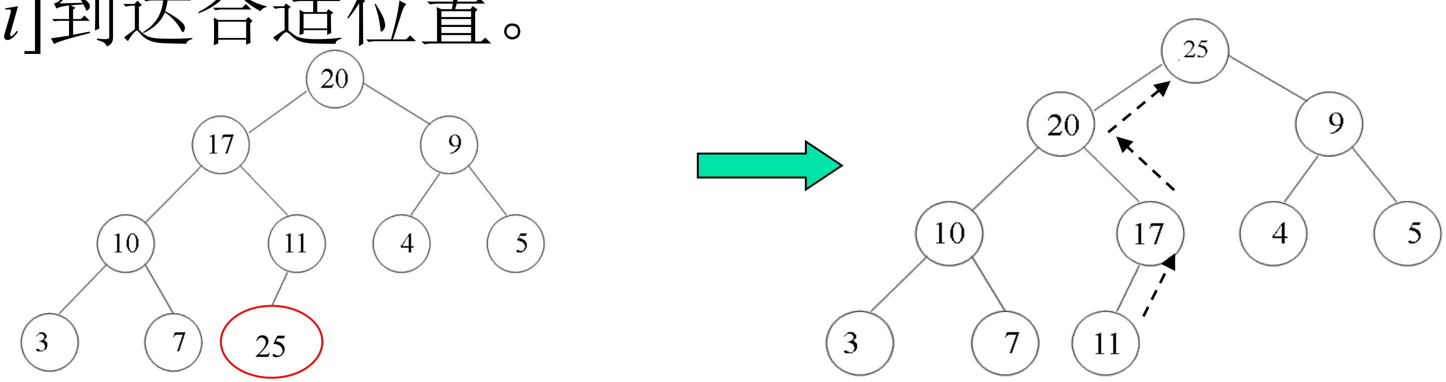
堆的基本操作

- make-heap(A): 从数组A创建堆
- insert(H,x): 插入元素x到堆H中
- delete(H,i): 删除堆H的第i项
- delete-max(H): 从非空堆H中删除最大键值并返回数据项

如何实现？

4.2.1 辅助运算sift-up

- 若某个节点$H[i]$键值大于其父节点的键值，违背了堆的特性，需要进行调整。
- 调整方法：上移。
- 沿着$H[i]$到根节点的唯一一条路径，将$H[i]$移动到合适的位置上：比较H[i]及其父节点$H[\lfloor i/2 \rfloor]$的键值，若key($H[i]$)>key($H[\lfloor i/2 \rfloor]$)，则二者进行交换，直到$H[i]$到达合适位置。



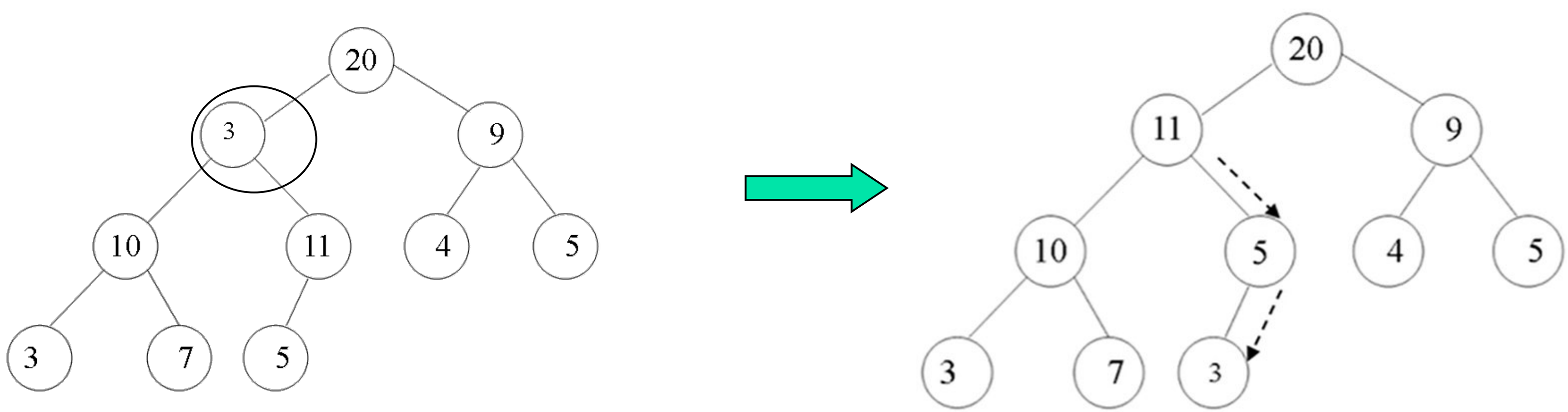
上移(sift-up)操作

过程 Sift-up(H,i)
输入：数组H[1…n]，索引1≤ i≤n
输出：上移H[i] (如果需要)，使它的键值不大于父节点的键值
1. while i>1
2. if key(H[i])>key(H[$\lfloor i/2 \rfloor$]) then 互换 H[i] 和H[$\lfloor i/2 \rfloor$]
3. else return //调整过程至此已经满足要求，可退出
4. i←$\lfloor i/2 \rfloor$
//调整进行到根节点，或到某一节点终止
5. return

- 时间复杂度：$O(\log n)$
- 空间复杂度：$O(1)$

4.2.1 辅助运算sift-down

- 假如某个内部节点$H[i]$ ($i\le\lfloor n/2\rfloor$)，其键值小于儿子节点的键值,即key($H[i]$)<key($H[2i]$)或key($H[i]$)<key($H[2i+1]$) (如果右儿子存在)，违背了堆特性，需要进行调整。
- 调整方法：下渗。
- 沿着从$H[i]$到子节点(可能不唯一，取其**键值较大者**)的路径，比较$H[i]$与子节点的键值，若key($H[i]$) < max($H[2i]$, $H[2i]$+1)则交换之。这一过程直到叶子节点或满足堆特性为止。



下移(sift-down)操作

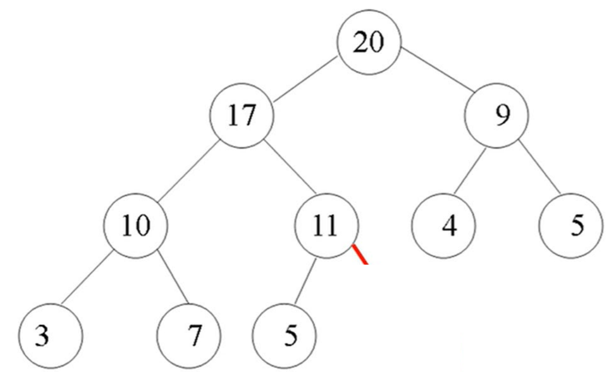
```
过程 Sift-down(H,i)
输入：数组H[1…n]，索引1≤ i≤n
输出：下滲H[i] (若它违背了堆特性)，使H满足堆特性
1.  while 2i≤n
2.      i←2i
3.      if i+1≤n and key(H[i+1] )> key(H[i]) then i=i+1 //有右儿子,取
                                              //左右孩子中较大者
4.      if  key(H[⌊i/2⌋])<key(H[i]) then 互换 H[i] 和 H[⌊i/2⌋]
5.      else return //调整过程至此已经满足堆特性，可退出
6.      end if
7.   //调整进行到叶节点，或到某一节点终止
8.  return
```

- 时间复杂度：$O(\log n)$
- 空间复杂度：$O(1)$

操作insert(*H*, *x*)

- 思路：先将*x*添加到*H*的末尾，利用Sift-up，调整*x*在*H*中的位置，直到满足堆特性

```
输入：堆H[1…n]和元素x
输出：新堆H[1…n+1]，x是其中元素之一
1. n←n+1 {堆大小增1}
2. H[n] ← x;
3. Sift-up(H,n) {调整堆}
```

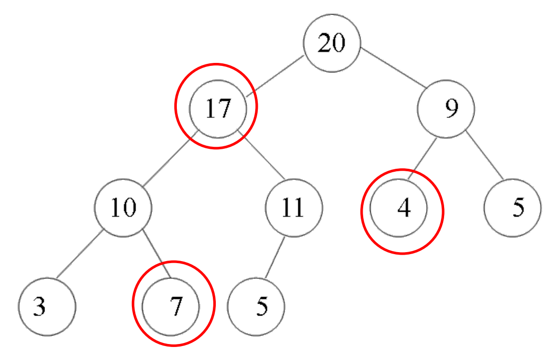


树的高度为$\lfloor \log n \rfloor$，所以将一个元素插入大小为*n*的堆所需要的时间是O(logn)

操作 delete(*H*, *i*)

- 思路：先用*H*[*n*]取代*H*[*i*]，然后对*H*[*i*]作Sift-up或Sift-down，直到满足堆特性。

输入：非空堆H[1…n]，索引i，1≤i≤n.
输出：删除H[i]之后的新堆H[1…n-1].
1. x←H[i]; y←H[n];
2. n←n-1; {堆大小减1}
3. if i=n+1 then exit {要删除的刚好是最后
   一个元素，叶节点}
4. H[i]←y; {用原来的H[n]取代H[i]}
5. if key(y) ≥key(x) then Sift-up(H,i)
6. else Sift-down(H,i);
7. end if



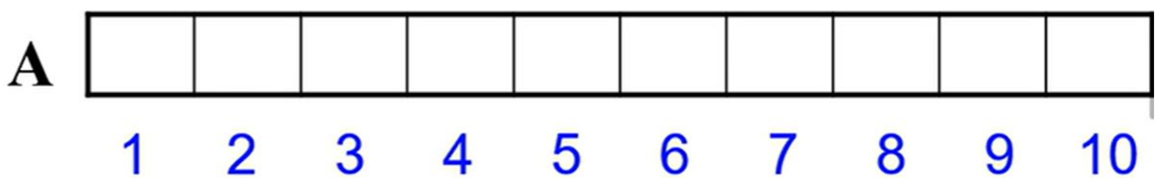
所需要的时间是O(logn).

操作delete-max(H)

输入：堆H[1…n]
输出：返回最大键值元素，并将其从堆中删除
1. x←H[1]
2. delete(H,1)
3. return x

操作make-heap(A)

- 方法1：从一个空堆开始，逐步插入A中的每个元素，直到A中所有元素都被转移到堆中

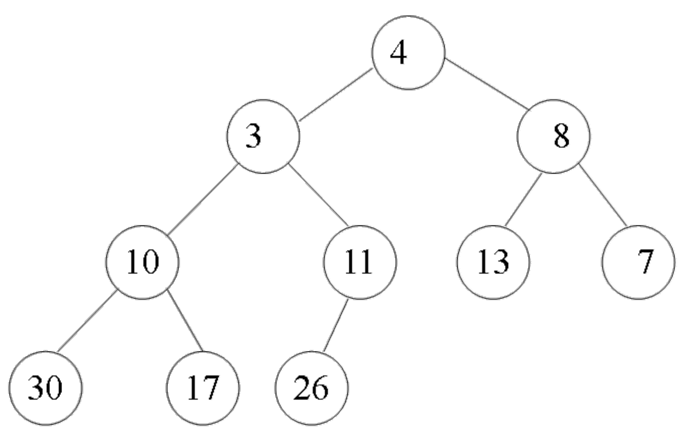


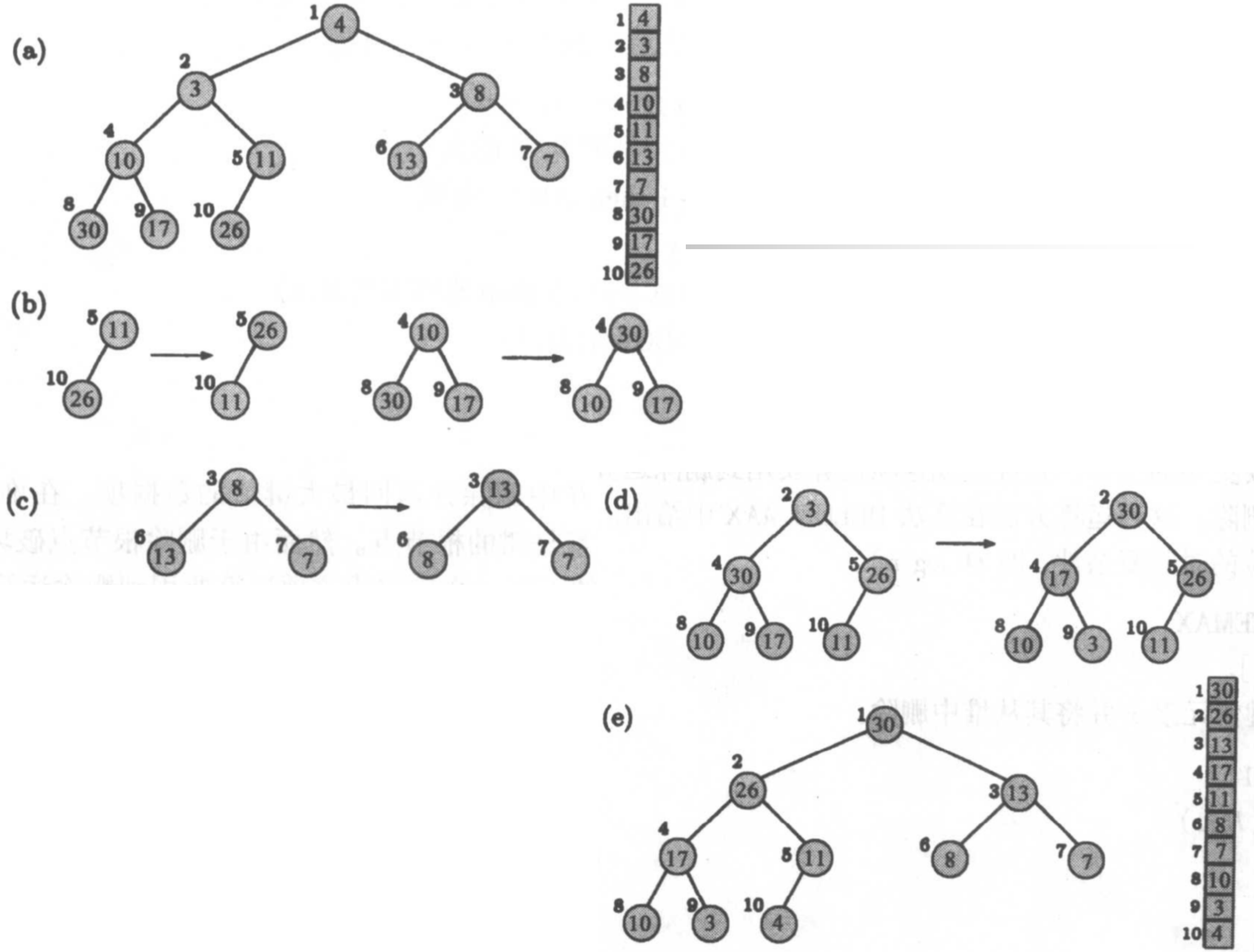
- 时间复杂度为$O(n\log n)$.为什么？(阅读教材)

方法2：

MAKEHEAP (创建堆)
输入：数组A[1…n]
输出：将A[1…n]转换成堆
1. for i← $\lfloor n/2 \rfloor$ downto 1
2. Sift-down(A,i) {使以A[i]为根节点的子树调整成为堆，故调用down过程}
3. end for

例：给定数组$A[1\ldots10] = \{4, 3, 8, 10, 11, 13, 7, 30, 17, 26\}$




(a)
(b)
(c)
(d)
(e)

复杂度分析

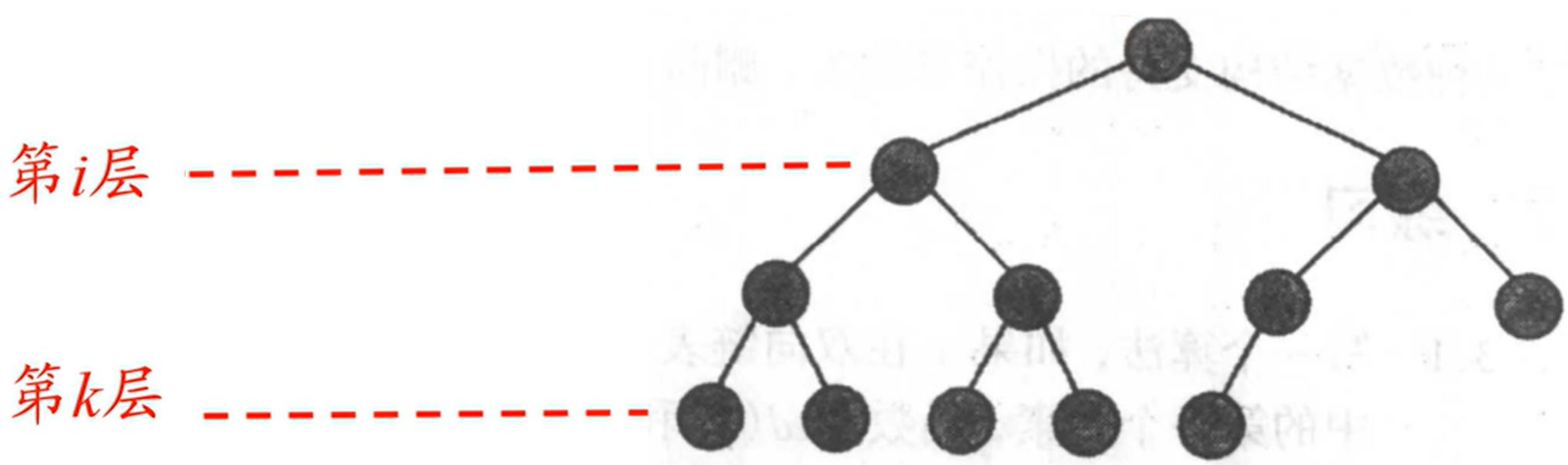
- 树的高度$k=$?

$$k = \lfloor \log n \rfloor,$$

- 对于$0 \le i < k$，第i层的节点个数？

第i层正好2^i个节点

- 对于第i层的每个节点，down过程最多执行***k-i***次



复杂度分析

- 树高$k=\lfloor \log n \rfloor$，第i层正好2^i个节点, $0 \le i < k$, (不含最深的叶子节点层), 每个节点的down过程最多执行$k-i$次,故down过程执行次数上限为

$$\sum_{i=0}^{k-1}(k-i)2^i = \sum_{j=k}^{1} j2^{k-j} (\text{令} k-i=j)$$

$$= 2^k \sum_{j=1}^{k} j2^{-j} = 2^k \Theta(1)$$

$$\le n \cdot \Theta(1) < 2n$$

- 时间复杂度为O(n).

堆排序

算法 4.5　HEAPSORT

输入： n 个元素的数组 $A[1 \cdots n]$。

输出： 以非降序排列的数组 A。

1. **MAKEHEAP**(A)
2. **for** $j \leftarrow n$ **downto** 2
3. 　　互换 $A[1]$和 $A[j]$
4. 　　SIFT-DOWN($A[1 \cdots j-1]$,1)
5. **end for**

算法复杂度

- 时间复杂度：$O(n \log n)$
- 空间复杂度：$\Theta(1)$

排序的最优算法是不是nlogn？

- 目前所知，如果是比较排序的话，是
- 非比较排序，可以更低

计数排序

- 算法（适用于整数排序且整数数值较小）
  - 统计每个数的个数，存储在数组C
    - C的标号代表数值，C的值代表个数
  - 将C的每个元素值依次往后累加
    - $C[i]=C[i]+1$
    - 针对每个数x，得出x应放的位置n（小于等于x的元素有n-1个）
  - 将x放在第n个位置

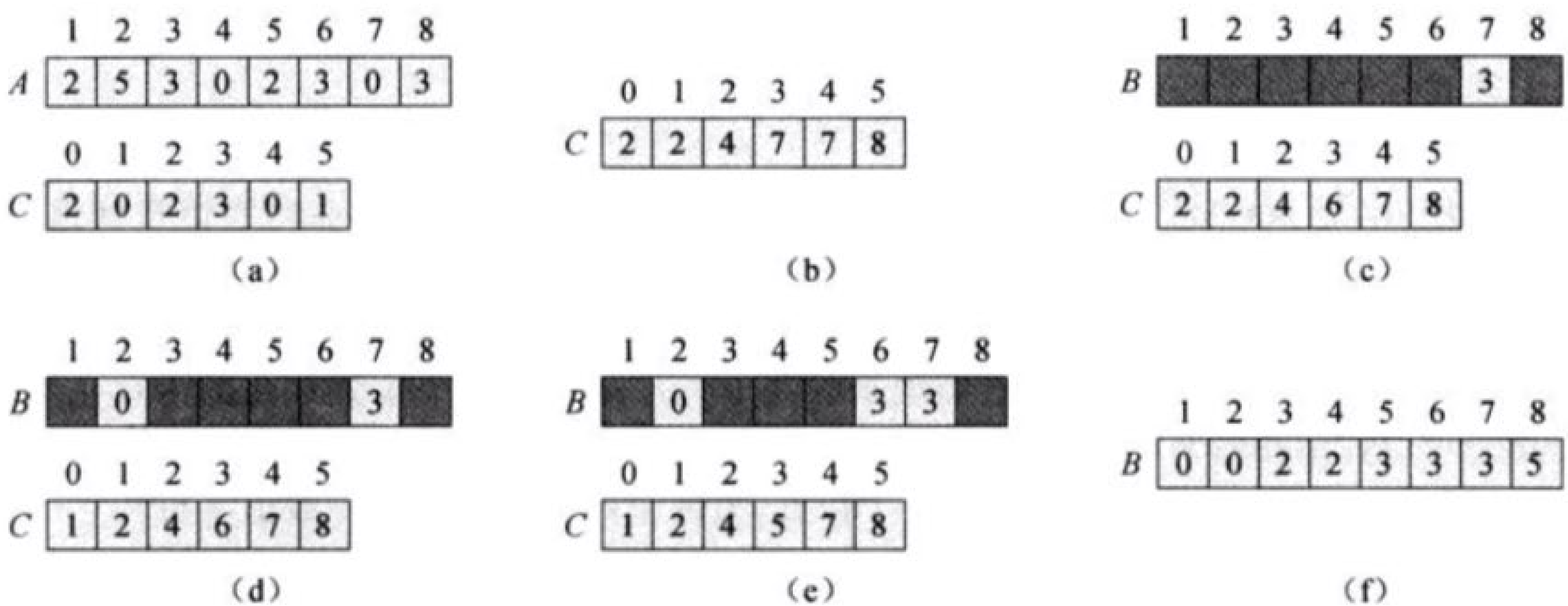
计数排序

COUNTING SORT

```
COUNTING-SORT(A, B, k)
for i ← 1 to k
    do C[i] ← 0
for j ← 1 to length[A]
    do C[A[j]] ← C[A[j]]+1
//C[i] now contains the number of elements equal to i.
for i ← 2 to k
    do C[i] ← C[i]+ C[i-1]
//C[i] now contains the number of elements less than or equal to i.
for j ← length[A] downto 1
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]]-1
```

$A[1\ldots n]$为要排序的数组，$B[1\ldots n]$存放排序好的数组，k为最大的数，$C[0\ldots k]$提供临时存储空间

计数排序



总的时间代价就是 $\Theta(k+n)$。在实际工作中，当 $k=O(n)$ 时，我们一般会采用计数排序，这时的运行时间为 $\Theta(n)$。

基数排序

- 算法（适用于具有相同或相近位数的数据的排序）
  - 对所有数据按最后一位进行排序
  - 在上述排序的基础上，按前一位进行排序
  - 重复第二步一直到最高位

Operation of radix sort

3 2 9		7 2 0
4 5 7		3 5 5
6 5 7		4 3 6
8 3 9	Stable sort	4 5 7
4 3 6		6 5 7
7 2 0		3 2 9
3 5 5		8 3 9

Operation of radix sort

3 2 9		7 2 0	7 2 0
4 5 7		3 5 5	3 2 9
6 5 7		4 3 6	4 3 6
8 3 9	Stable sort →	4 5 7	8 3 9
4 3 6		6 5 7	3 5 5
7 2 0		3 2 9	4 5 7
3 5 5		8 3 9	6 5 7

Stable sort →

Operation of radix sort

3 2 9		7 2 0		7 2 0		3 2 9
4 5 7		3 5 5		3 2 9		3 5 5
6 5 7		4 3 6		4 3 6		4 3 6
8 3 9	Stable sort →	4 5 7	Stable sort →	8 3 9	Stable sort →	4 5 7
4 3 6		6 5 7		3 5 5		6 5 7
7 2 0		3 2 9		4 5 7		7 2 0
3 5 5		8 3 9		6 5 7		8 3 9

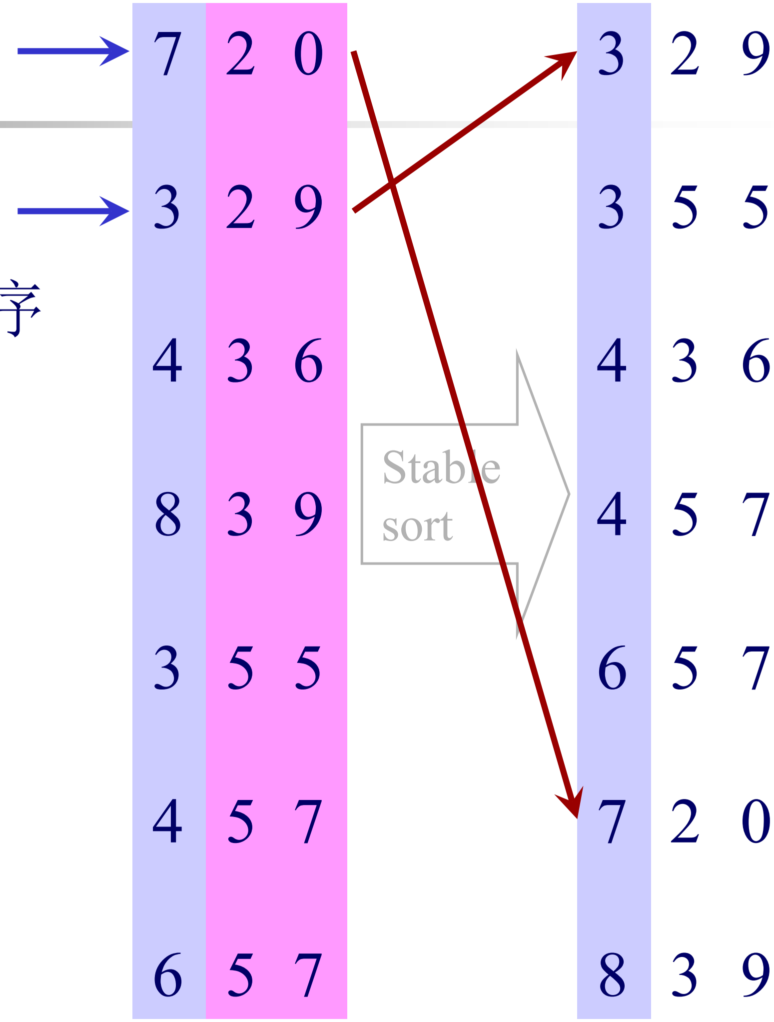
Operation of radix sort-correctness

看一下基数排序的正确性

- 假设已经按照 *t – 1* 位进行排序

- 现对 *t*位进行排序

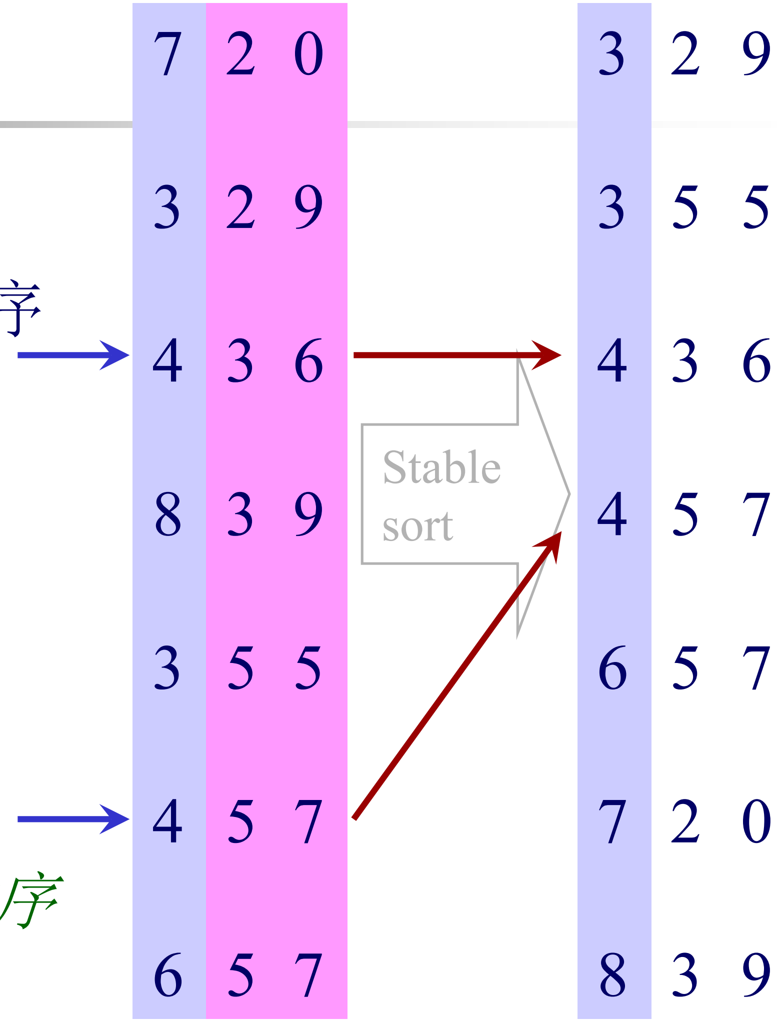
不同的t位数据按照正确的排序



Operation of radix sort-correctness

看一下基数排序的正确性

- 假设已经按照 *t – 1* 位进行排序



- 现对 *t*位进行排序

不同的t位数据按照正确的排序

相同的t位数据也按照正确的排序

基数排序

算法 5.3 RADIXSORT

输入： 一张有 n 个数的表 $L=\{a_1, a_2, \cdots, a_n\}$ 和 k 位数字。

输出： 按非降序排列的 L。

```
for j ← 1 to k
    准备 10 个空表 L0, L1, …, L9。
    while L 非空
        a ← L 中的下一元素；删除 a。
        i ← a 中的第 j 位数字；将 a 加入表 Li 中
    end while
    L ← L0
    for i ← 1 to 9
        L ← L, Li    {将表 Li 加入 L 中}
    end for
end for
return L
```

- $L0, L1, \ldots, L9$表用于存放每一位上相应的数据，即当比较第i位时，数据a的第i位5，则存放在$L5$中
- L按顺序存放$L0$一直到$L9$

注：设置$L0$到$L9$十个表的作用就在于避免排序

可以从高到低位排序吗？

- 不能直接按上面的思路排，需要一种递归的方法

基数排序

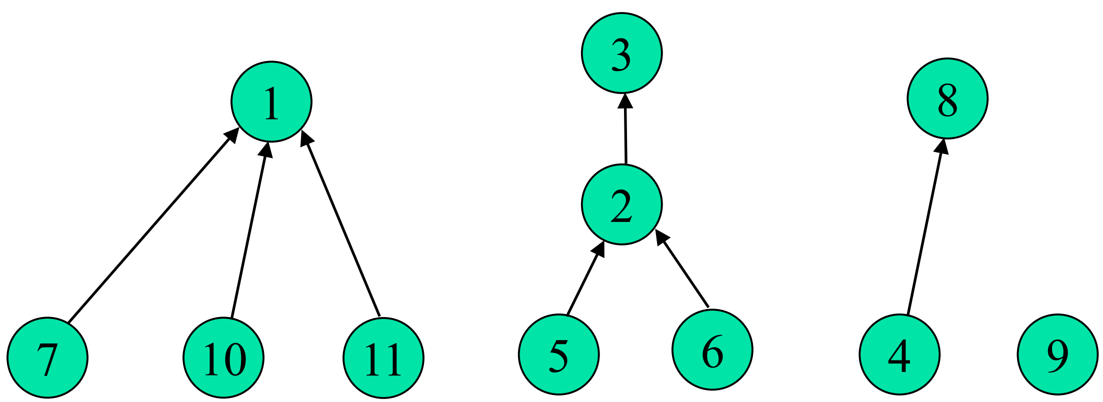
- 算法时间复杂度(按迭代次数计算): $\Theta(kn)=\Theta(n)$
- 算法空间复杂度（十个表，每个表都是n）：$\Theta(10n)=\Theta(n)$

不相交集(Disjoint Sets)

- 假设有n个元素，被分成若干个集合。例如S={1,2,...11}分成4个子集1:{1,7,10,11}，3:{2,3,5,6}，8:{4,8}，9:{9}并分别命名。
- 事实上，每个子集可以用树表示，除根节点外，每个节点都有指针指向父节点。上例可以用树表示为:

不相交集(Disjoint Sets)

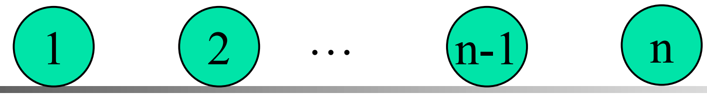
- 4个子集1:{1,7,10,11}，3:{2,3,5,6}，8:{4,8}，9:{9}并分别命名。



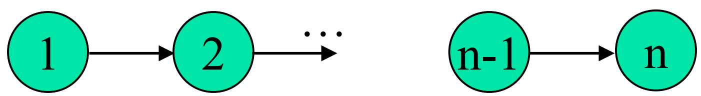
不相交集(Disjoint Sets)

- 假如要执行如下计算任务：
  - FIND(x): 寻找包含元素x的集合的名字
  - UNION(x,y): 将包含元素x和y的两个集合合并，重命名
- 记root(x)为包含元素x的树的根，则FIND(x)返回root(x).
- 执行合并UNION(x, y)时，首先依据x找到root(x),记为u,依据y找到root(y)，记为v；然后，将u指向v。
- 优点：简单明了
- 缺点：多次合并后，树高度可能很大，查找困难。

例：初始状态：$\{1\},\{2\},\ldots,\{n\}$



执行合并序列：UNION(1,2),UNION(2,3),…UNION(n-1,n)，得到



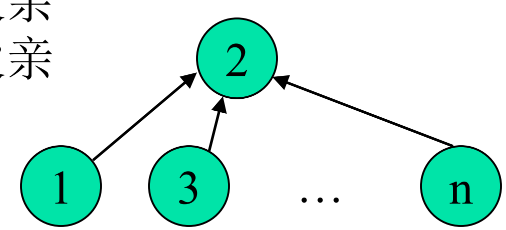
执行查找序列：FIND(1), FIND(2),…, FIND(N).需要比较的次数是：

$$n+(n-1)+\cdots+2+1=\frac{n(n+1)}{2}$$

目标：降低树的高度。措施：Rank Heuristic。

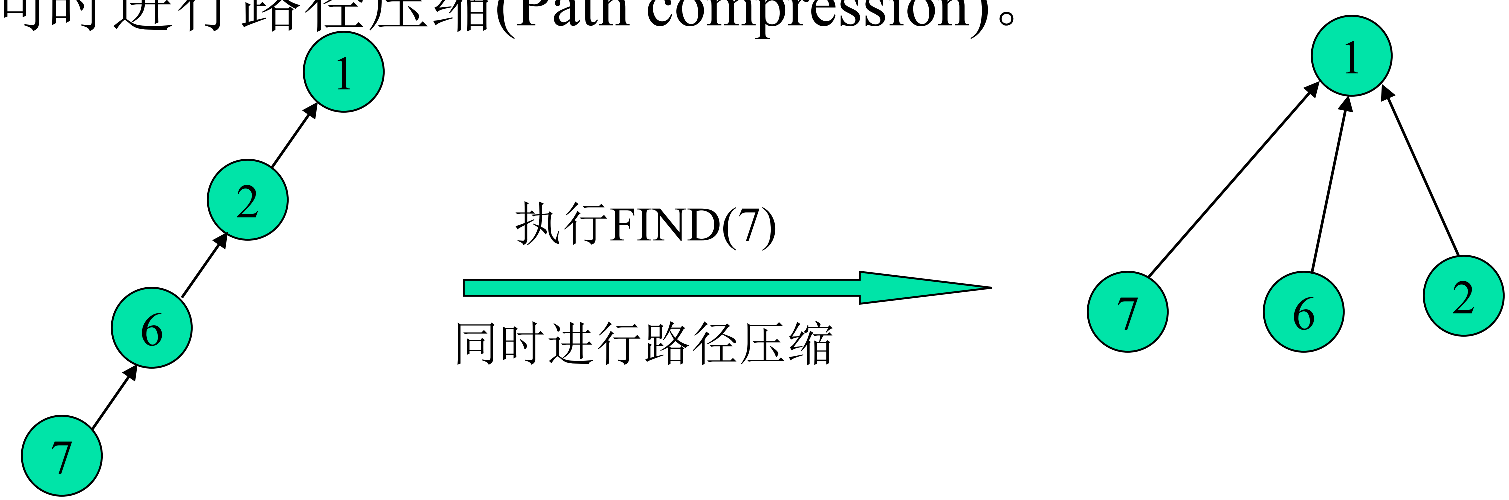
1.给每个树的<u>根节点</u>定义一个秩(rank)，表示该树的高度。
2.在执行UNION(x, y)，首先找到u=root(x)，v=root(y)。
3.然后比较rank(u)和rank(v)

若rank(u) = rank(v)，则使u指向v，v成为u的父亲，同时rank(v)+1
若rank(u) < rank(v)，则使u指向v，v成为u的父亲
若rank(u) > rank(v)，则使v指向u，u成为v的父亲



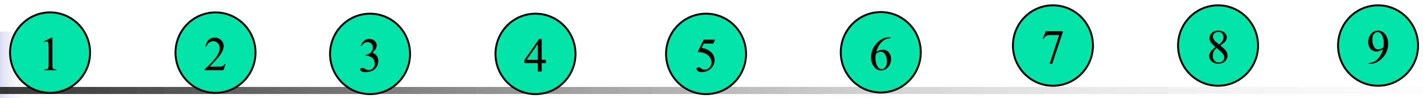
Algorithm: UNION
输入：两个元素x, y.
输出：将包含x, y的两棵树合并
1. u←FIND(x); v←FIND(y)
2. if rank(u)≤rank(v) then // Rank Heuristic
3. p(u) ←v
4. if rank(u) = rank(v) then rank(v) = rank(v) + 1
5. else
6. p(v) ←u
7. end if

目标：进一步提高FIND的操作的性能。措施：在执行FIND操作时，同时进行路径压缩(Path compression)。

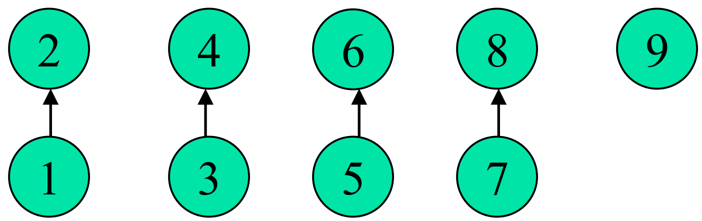


```
Algorithm: FIND
输入：节点x
输出：root(x)和路径压缩后的树
1.  y←x
2.  while p(y)≠null {寻找包含x的树的根}
3.      y←p(y)
4.  end while
5.  root←y; y←x {重新赋值为原来的节点x}
6.  while p(y) ≠null {执行路径压缩}
7.      w←p(y) {父节点暂存为w}
8.      p(y) ←root {该路径上的节点直接指向根节点}
9.      y←w {继续下一步压缩}
10.end while
11.return root
```

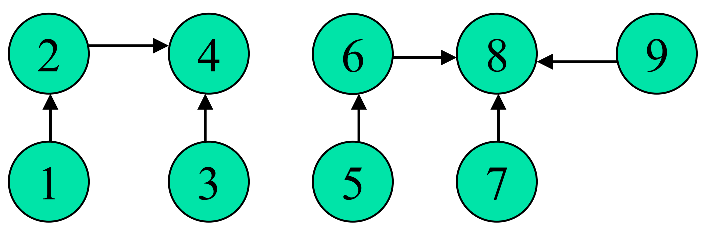
例：初始状态：{1},{2},…,{9}



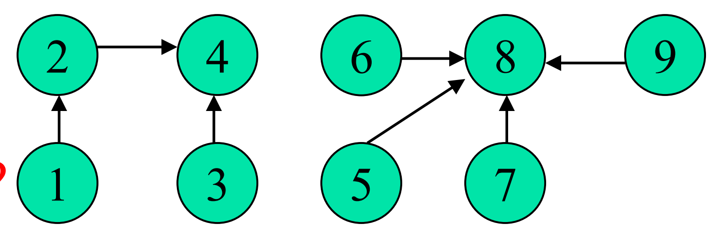
执行合并序列：UNION(1,2),UNION(3,4),UNION(5,6) ,UNION(7,8),得到



继续执行合并序列：UNION(2,4),UNION(8,9),UNION(6,8)，得到



继续执行：FIND(5)得到

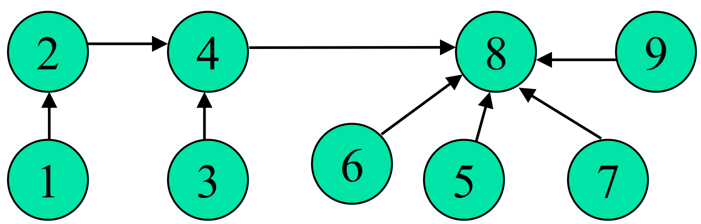


继续执行UNION(4,8)和FIND(1)呢?

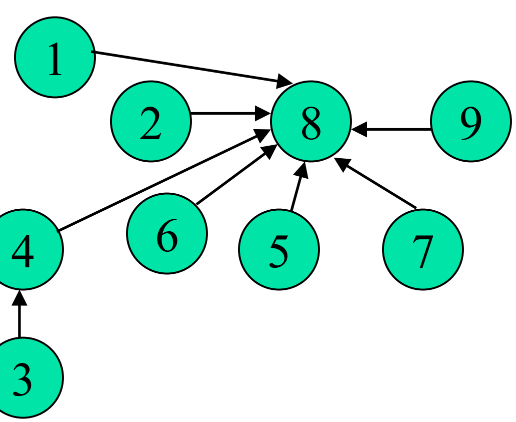
注意：路径压缩时，秩不会改变。

即执行FIND操作后，根节点的秩有可能大于树的高度。

继续执行：UNION(4,8)得到的结果是：



继续执行：FIND(1)得到的结果是：



作业

- P86-87
  - 4.3, 4.7, 4.13,
  - 4.29