武漢大學 WUHAN UNIVERSITY

算法设计与分析

数学预备知识与数据结构

林海
Lin.hai@whu.edu.cn

数学基础：集合

集合是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 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

又例如 所有的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∅：不含任何元素的集合

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

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A，使得$\varnothing \not\subseteq A$，即存在$x$，$x\in\varnothing$且$x\notin A$，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E:限定所讨论的集合都是E的子集.　　相对性

幂集

幂集$P(A)$：A的所有子集组成的集合， 即

$$P(A) = \{ x \mid x \subseteq A \}$$

例2 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

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

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

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

1、有序对

定义 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 $<x,y>$

实例：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：$<0,1> \neq <1,0>$

$<x,y>$与$<u,v>$相等的充分必要条件是

$$<x,y>=<u,v> \Leftrightarrow x=u \wedge y=v$$

例1 $<2,x+5>=<3y-4,y>$，求 x, y.

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

笛卡儿积

定义 设A, B为集合，A与B 的**笛卡儿积**记作$A\times B$，

$$A\times B = \{ \langle x,y\rangle \mid x\in A \wedge y\in B \}.$$

笛卡尔积$A\times A$ 我们常记作A^2

$$A^2 = \{(x, y) \mid x\in A \wedge y\in A\}$$

例2 $A=\{0, 1\}$, $B=\{a, b, c\}$

$A\times B=\{\langle 0,a\rangle,\langle 0,b\rangle,\langle 0,c\rangle,\langle 1,a\rangle,\langle 1,b\rangle,\langle 1,c\rangle\}$

$B\times A =\{\langle a,0\rangle,\langle b,0\rangle,\langle c,0\rangle,\langle a,1\rangle,\langle b,1\rangle,\langle c,1\rangle\}$

$A^2=A\times A=\{\langle 0,0\rangle,\langle 0,1\rangle,\langle 1,0\rangle,\langle 1,1\rangle\}$

$B^2=B\times B=\{\langle a,a\rangle,\langle a,b\rangle,\langle a,c\rangle,\langle b,a\rangle,\langle b,b\rangle,\langle b,c\rangle,$
$\langle c,a\rangle, \langle c,b\rangle,\langle c,c\rangle\}$

基数：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

定义

如果一个集合满足以下条件之一：

（1）集合非空，且它的元素都是有序对(有次序的二元组)

（2）集合是空集

则称该集合为一个二元关系，简称为关系，关系的名字一般用大写英文字母表示，通常记作R.

如$\langle x,y\rangle \in R$，可记作 xRy；如果$\langle x,y\rangle \notin R$，则记作$x \not R\, y$

实例：$R=\{\langle 1,2\rangle,\langle a,b\rangle\}$, $S=\{\langle 1,2\rangle,a,b\}$.

R是二元关系，当a, b不是有序对时，S不是二元关系

根据上面的记法，可以写$1R2$, aRb, $a\not R c$等.

关系的性质：自反性与反自反性

定义 设R为集合A上的关系,

(1) 若$\forall x(x\in A\rightarrow <x,x>\in R)$, 则称$R$在$A$上是自反的.

(2) 若$\forall x(x\in A\rightarrow <x,x>\notin R)$, 则称$R$在$A$上是反自反的.

自反：A上的全域关系E_A, 恒等关系I_A, 小于等于关系L_A, 整除关系D_A

反自反：实数集上的小于关系、幂集上的真包含关系.

例1 $A = \{a, b, c\}$, R_1, R_2, R_3 是 A上的关系, 其中

$R_1 = \{<a,a>,<b,b>\}$

$R_2 = \{<a,a>,<b,b>,<c,c>,<a,b>\}$

$R_3 = \{<a,c>\}$

R_2自反, R_3 反自反, R_1既不自反也不反自反.

对称性与反对称性

定义　设R为A上的关系,

(1) 若$\forall x\forall y(x,y\in A\wedge<x,y>\in R\rightarrow<y,x>\in R)$，则称$R$为$A$上对称的关系.

(2) 若$\forall x\forall y(x,y\in A\wedge<x,y>\in R\wedge<y,x>\in R\rightarrow x=y)$，则称$R$为$A$上的反对称关系.

实例　对称：A上的全域关系E_A, 恒等关系I_A和空关系$\varnothing$

反对称：恒等关系I_A，空关系是A上的反对称关系

例2 设$A=\{a,b,c\}$, R_1, R_2, R_3和R_4都是A上的关系, 其中

$R_1=\{<a,a>,<b,b>\}$， $R_2=\{<a,a>,<a,b>,<b,a>\}$

$R_3=\{<a,b>,<a,c>\}$， $R_4=\{<a,b>,<b,a>,<a,c>\}$

R_1 对称、反对称. R_2 对称，不反对称.

R_3 反对称，不对称. R_4 不对称、也不反对称

传递性

定义　设R为A上的关系, 若

$$\forall x\forall y\forall z(x,y,z\in A\wedge<x,y>\in R\wedge<y,z>\in R\rightarrow<x,z>\in R),$$

则称R是A上的传递关系.

实例：A上的全域关系 E_A, 恒等关系 I_A和空关系 $\varnothing$, 小于等于关系, 小于关系, 整除关系, 包含关系, 真包含关系

例3 设$A=\{a, b, c\}$, R_1, R_2, R_3是A上的关系, 其中

$R_1=\{<a,a>,<b,b>\}$

$R_2=\{<a,b>,<b,c>\}$

$R_3=\{<a,c>\}$

R_1 和 R_3 是A上的传递关系, R_2不是A上的传递关系.

等价关系

定义　设R为非空集合上的关系. 如果R是自反的、对称的和传递的, 则称R为A上的等价关系. 设 R 是一个等价关系, 若$<x,y>\in R$, 称 x等价于y, 记做$x\sim y$.

例如　数的相等关系是任何数集上的等价关系。

又例如　一群人的集合中姓氏相同的关系也是等价关系。

但朋友关系不是等价关系，因为它不可传递。

模n相等是不是等价关系？

等价关系性质

定义　设R为非空集合A上的等价关系, $\forall x\in A$，令

$$[x]_R = \{\, y \mid y\in A \wedge xRy \,\}$$

称 $[x]_R$ 为x关于R 的等价类, 简称为 x 的等价类, 简记为$[x]$.

实例　A={ 1, 2, … , 8 }上模 3 等价关系的等价类:

[1]=[4]=[7]={1,4,7}

[2]=[5]=[8]={2,5,8}

[3]=[6]={3,6}

等价关系划分集合，即所有等价类的并集就是A

偏序关系

定义　非空集合A上的**自反、反对称和传递**的关系，称为A上的**偏序关系**，记作≼.

设≼为偏序关系, 如果$<x, y>\in$≼, 则记作 $x≼y$, 读作 x“小于或等于” y.

例如:

集合A上的恒等关系 I_A是A上的偏序关系.

小于等于关系, 整除关系和包含关系也是相应集合上的偏序关系.

偏序关系：可比

定义 x与y可比 设R为非空集合A上的偏序关系，

任何 $x, y \in A$, x与y 可比 $\Leftrightarrow$ $x \preccurlyeq y \vee y \preccurlyeq x$.

例如:

在正整数集合的小于等于关系中，任何两个正整数x和y都是可比的。

而对于整除关系，任意两个正整数不一定可比。例如2不能整除3，3也不能整除2，所以2和3不是可比的。

鸽巢原理

定理 2.3 如果把 n 个球分别放在 m 个盒子中，那么：

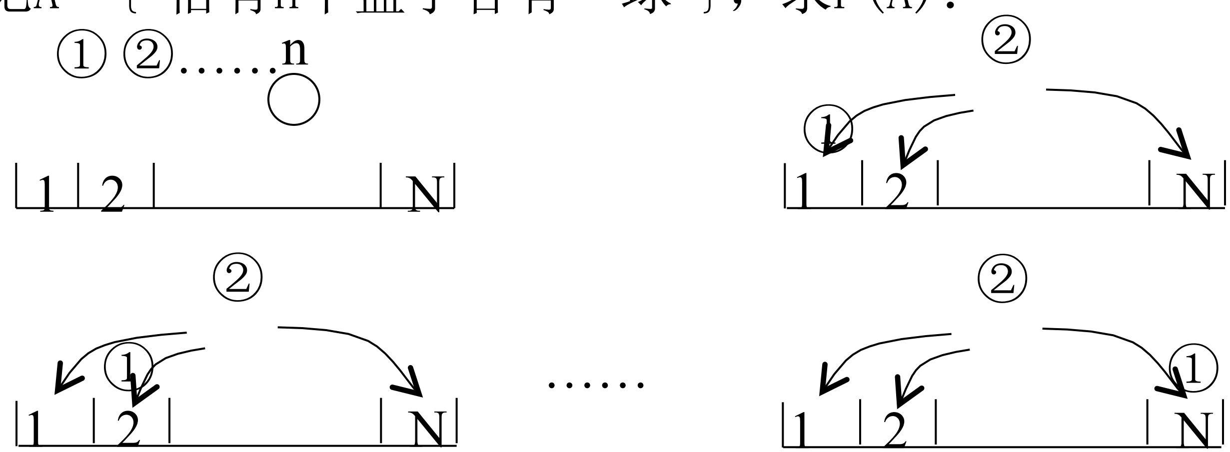
(1) 存在一个盒子，必定至少装$\lceil n/m \rceil$个球；

(2) 存在一个盒子，必定最多装$\lfloor n/m \rfloor$个球。

- 将n个不同的球，投入N个不同的盒中(n≤N)，设每一球落入各盒的概率相同，且各盒可放的球数不限，

  记A＝{ 恰有n个盒子各有一球 }，求P(A).

解：



即当n＝2时，共有N^2个样本点；一般地，n个球放入N个盒子中，总样本点数为N^n，使A发生的样本点数 $=C_N^n \cdot n! \Rightarrow P(A)=C_N^n \cdot n!/N^n$

2.1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 $n=4k^2=2(2k^2)$，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 $n=p_1p_2\cdots p_k+1$，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 $p_1p_2\cdots p_k$，因此，p 整除 $n-p_1p_2\cdots p_k$，但是 $n-p_1p_2\cdots p_k=1$，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=n0,n0+1,n0+2…..为真
  - 基础步：证明该性质对于n0成立
  - 归纳步：假设该性质对于，n0,n0+1,….n-1 成立，证明对于n成立。

数据结构

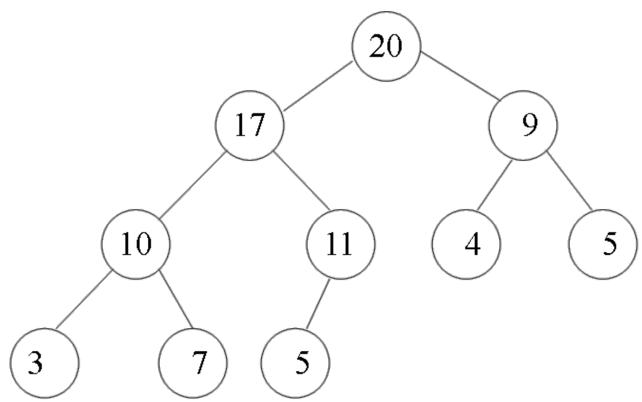
- 算法的实现离不开数据结构。选择一个合适的数据结构对设计一个有效的算法有十分重要的影响。结构化程序设计创始人Niklaus Wirth(瑞士苏黎士高工)提出一个著名的论断："程序=算法+数据结构"。1984年，Wirth因开发了Euler、Pascal等一系列崭新的计算语言而荣获图灵奖, 有"结构化程序设计之父"之美誉。
- 本章我们将回顾几种重要的数据结构，包括二叉树、堆、不相交集。

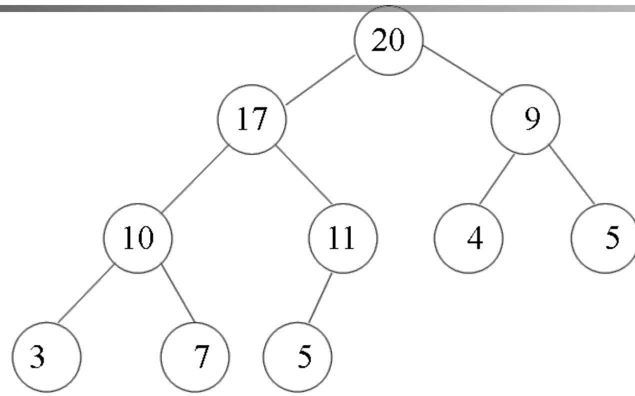
4.2堆(Heap)

- 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)

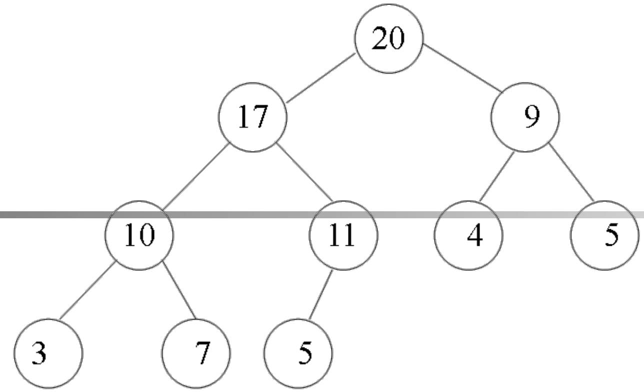
- 堆的定义：堆是一个几乎完全的二叉树，每个节点都满足这样的特性：任一父节点的键值(key)不小于子节点的键值。
- 沿着每条从根到叶子的路径，元素键值以非升序排列





- 有n个节点的堆T,可以用一个数组H[1…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。

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



观察结论：

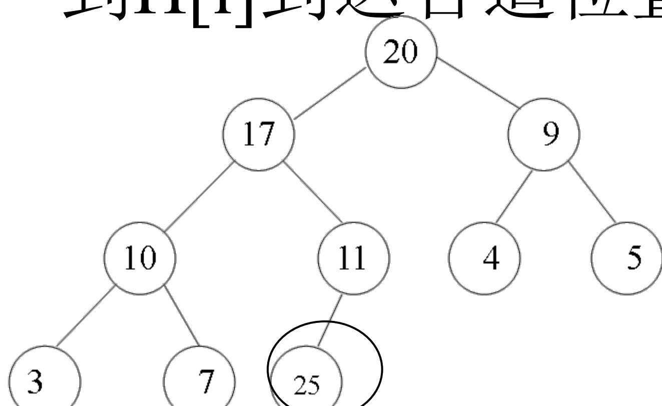
- 根节点键值最大，叶子节点键值较小。从根到叶子，键值以非升序排列。
- 节点的左右儿子节点键值并无顺序要求。
- 堆的数组表示呈“基本有序”状态。相应地，并非节点的高度越高，键值就越大。

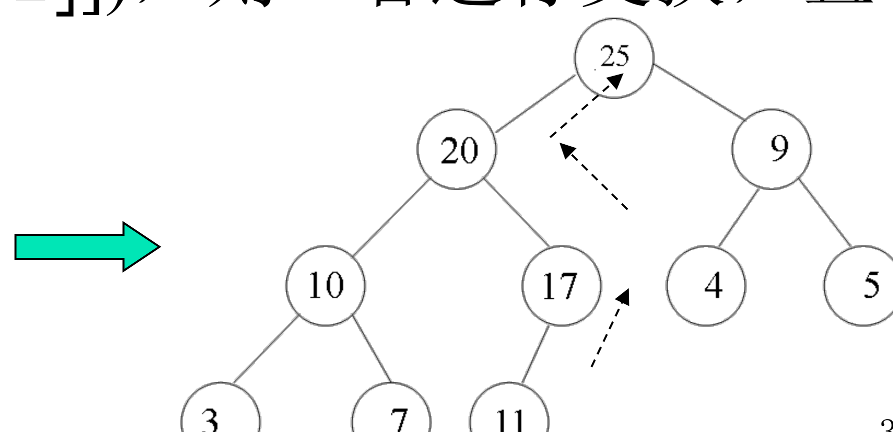
堆的基本操作

- 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

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，就违背了堆的特性，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[⌊i/2⌋]的键值，若key(H[i])>key(H[⌊i/2⌋])，则二者进行交换，直到H[i]到达合适位置。





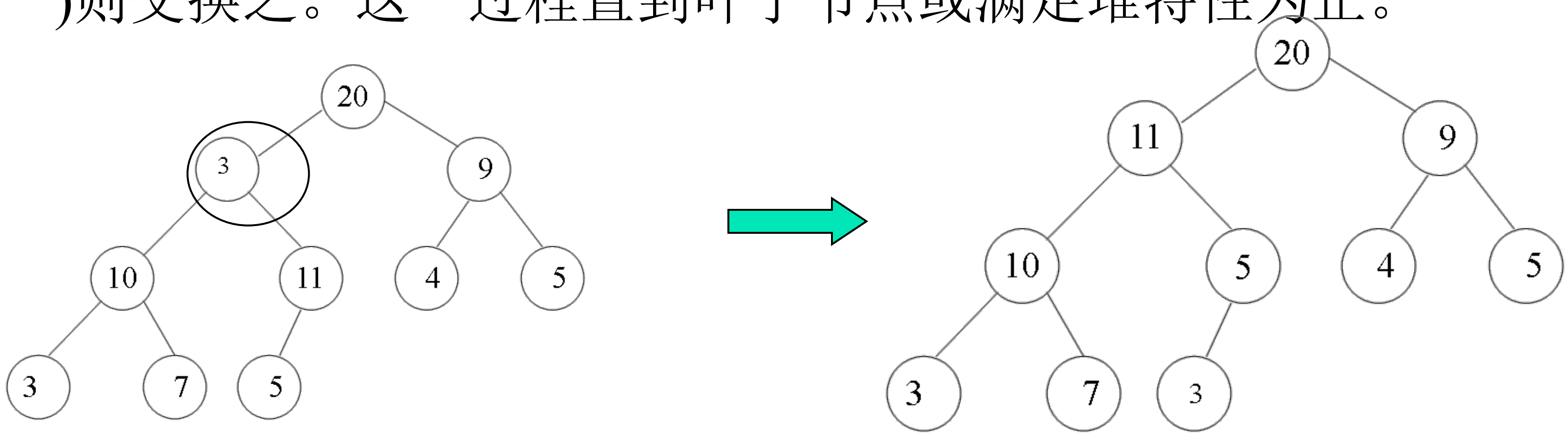
Sift-up

过程 Sift-up(H,i)
输入：数组H[1…n]，索引$1 \le i \le n$
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. done←false
2. if i=1 then exit {根节点}
3. repeat
4. 　　if key(H[i])>key(H[$\lfloor i/2 \rfloor$]) then 互换 H[i] 和H[$\lfloor i/2 \rfloor$]
5. 　　else done←true {调整过程至此已经满足要求，可退出}
6. 　　i←$\lfloor i/2 \rfloor$
7. until i=1 or done {调整进行到根节点，或到某一节点终止}

4.2.1 辅助运算Sift-down

- 假如某个内部节点H[i] (i≤⌊n/2⌋)，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，则取其键值较大者)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。



Sift-down

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. done←false
2. if 2i>n, then exit {叶子节点，无须进行}
3. repeat
4. 　　i←2i
5. 　　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

6. 　　if key(H[⌊i/2⌋])< key(H[i]) then 互换 H[i] 和 H[⌊i/2⌋]
7. 　　else done←true {调整过程至此已经满足堆特性，可退出}
8. 　　end if
9. until 2i>n or done {调整进行到叶节点，或到某一节点终止}

操作insert(H,x): 插入元素x到堆H中

- 思路：先将x添加到H的末尾，然后利用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⌊logn⌋,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) {如果最后一个元素比被删
   除的元素大，则需要上移}
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. $x \leftarrow H[1]$
2. delete(H,1)
3. return x

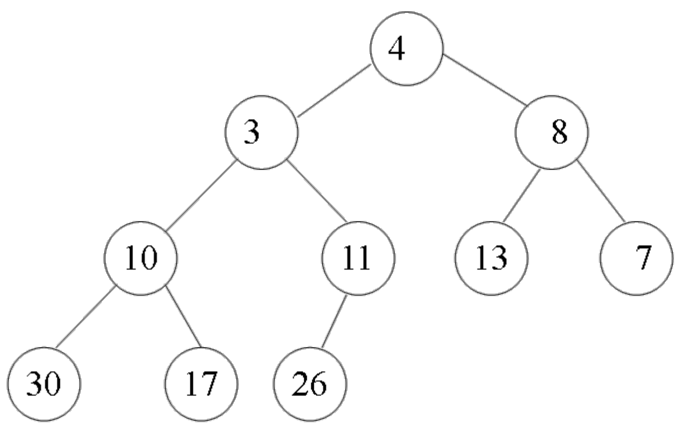
make-heap(A): 从数组A创建堆

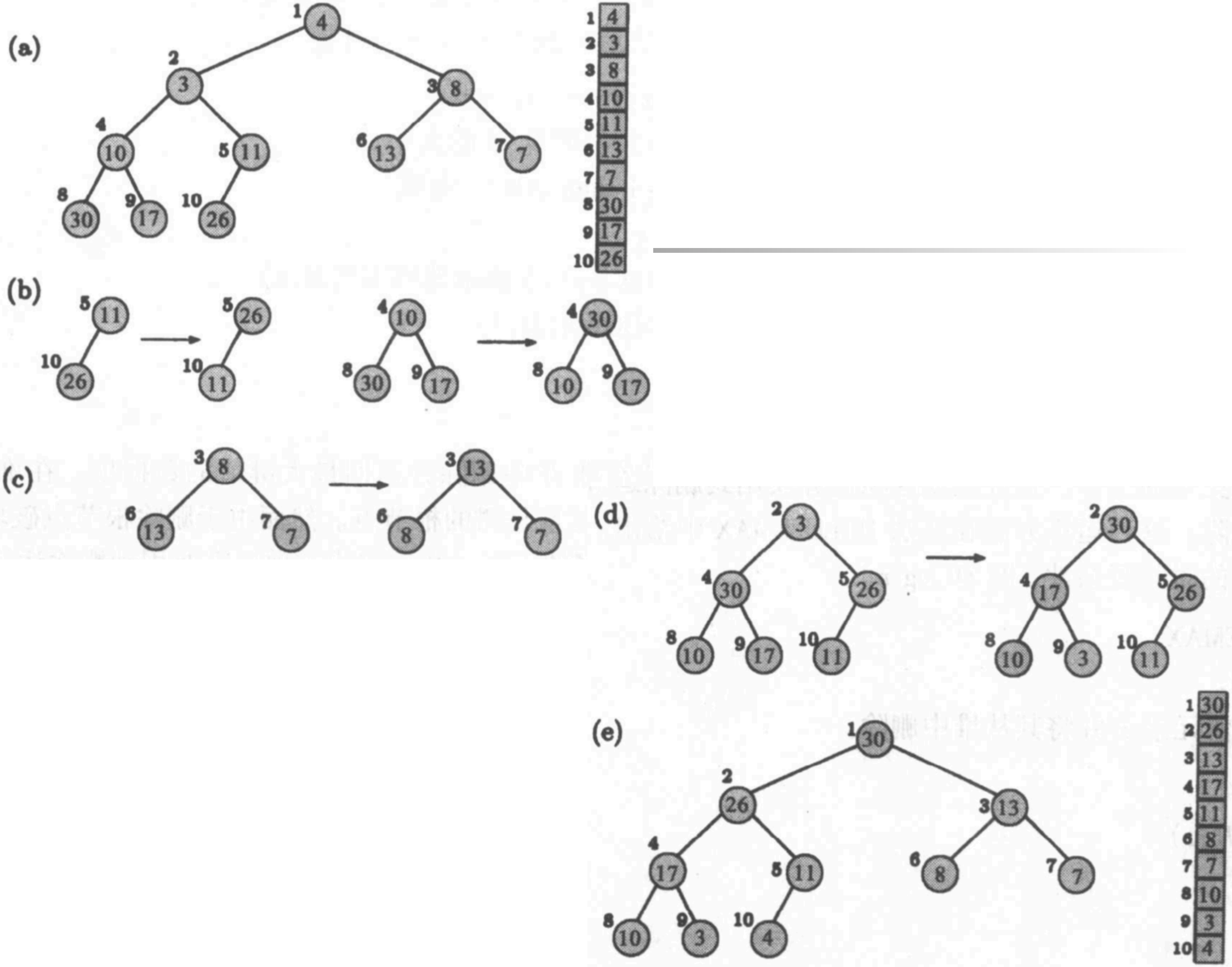
- 方法1：从一个空堆开始，逐步插入A中的每个元素，直到A中所有元素都被转移到堆中。
- 时间复杂度为O(nlogn).为什么？
  - 因为插入一个元素需要log n，总共需要插入n个元素

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{必须从下到上，叶子无需调整}
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…10] = {4, 3, 8, 10, 11, 13, 7, 30, 17, 26}





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

复杂度分析

- 树高k=⌊logn⌋，第i层正好2^i个节点, 0≤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

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

$$=2^k\sum_{j=1}^{k}j2^{-j}=2^k\Theta(1) \qquad 公式2.14$$

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

- 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

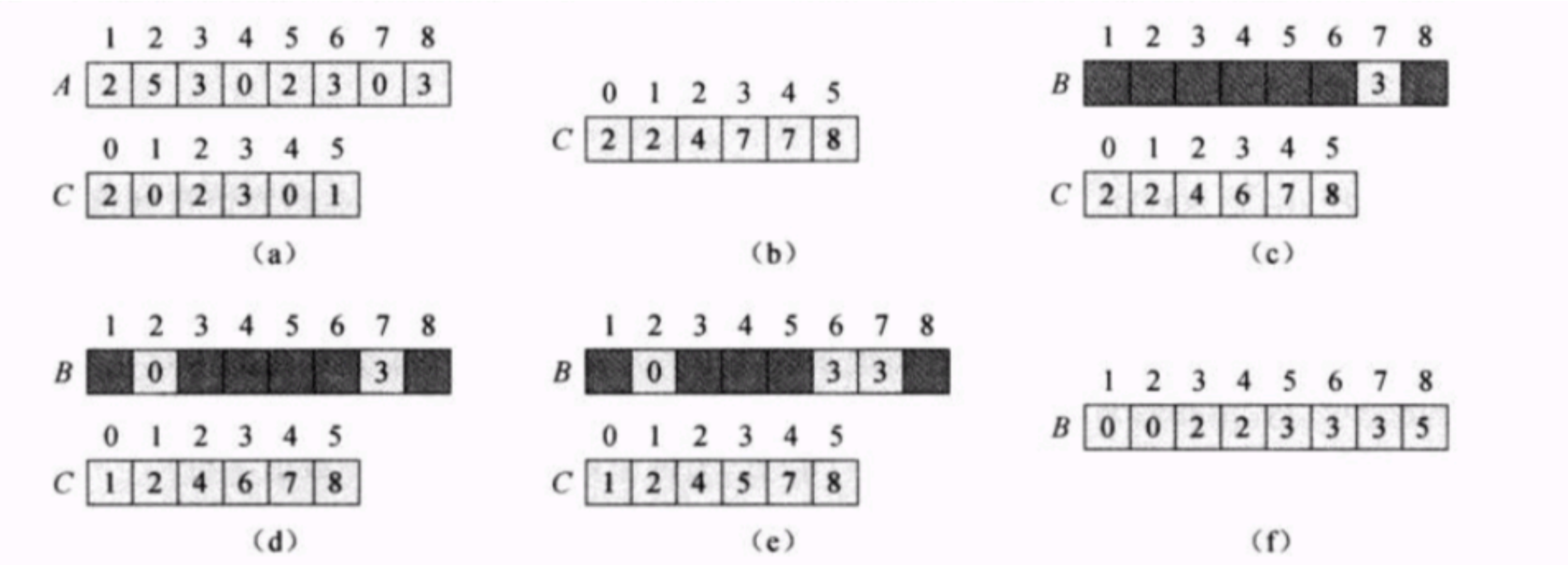
计数排序

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⋯n]为要排序的数组，B[1⋯n]存放排序好的数组，k为最大的数，C[0⋯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	Stable sort →	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

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

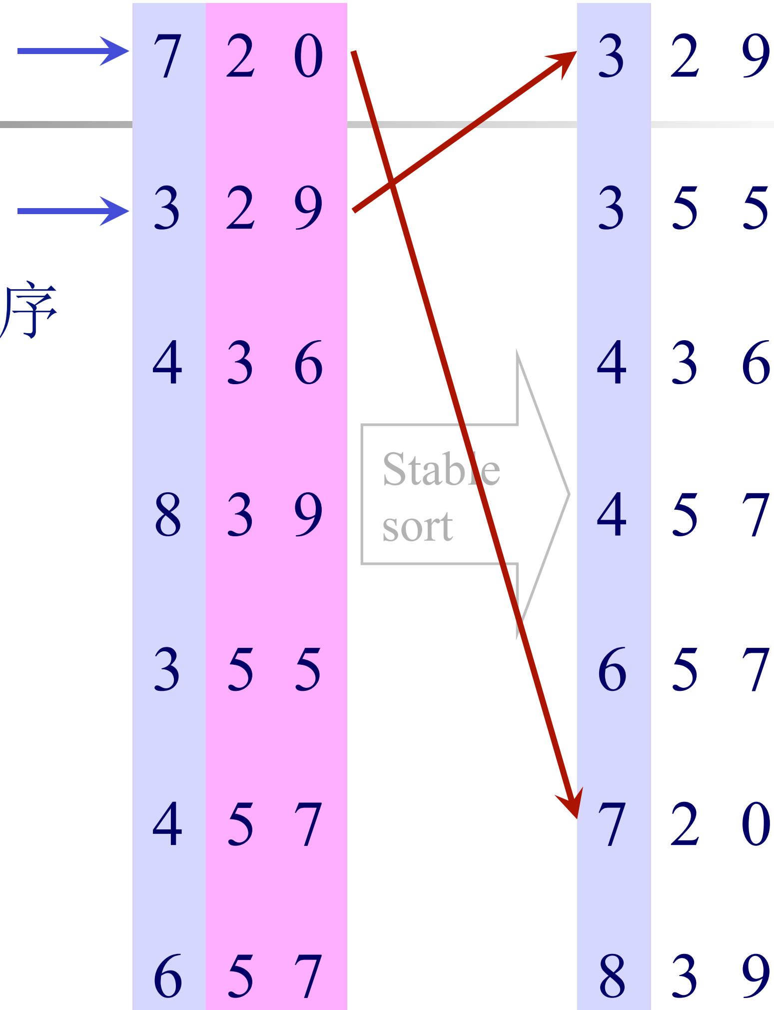
Operation of radix sort-correctness

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

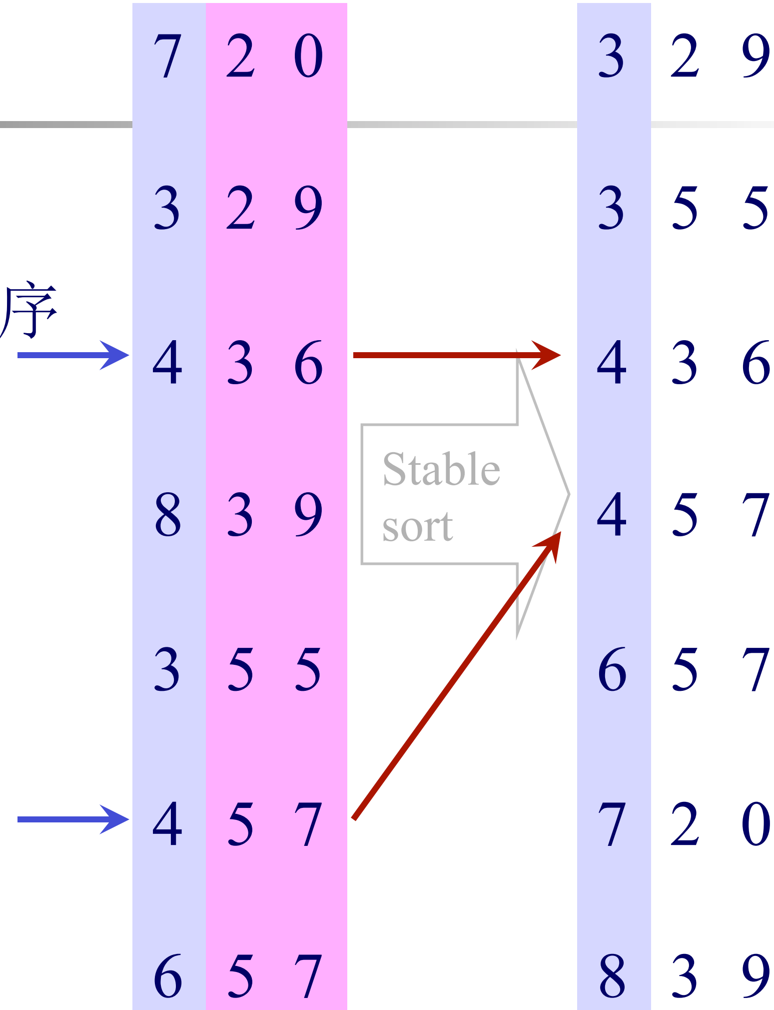
看一下基数排序的正确性

- 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。。。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为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

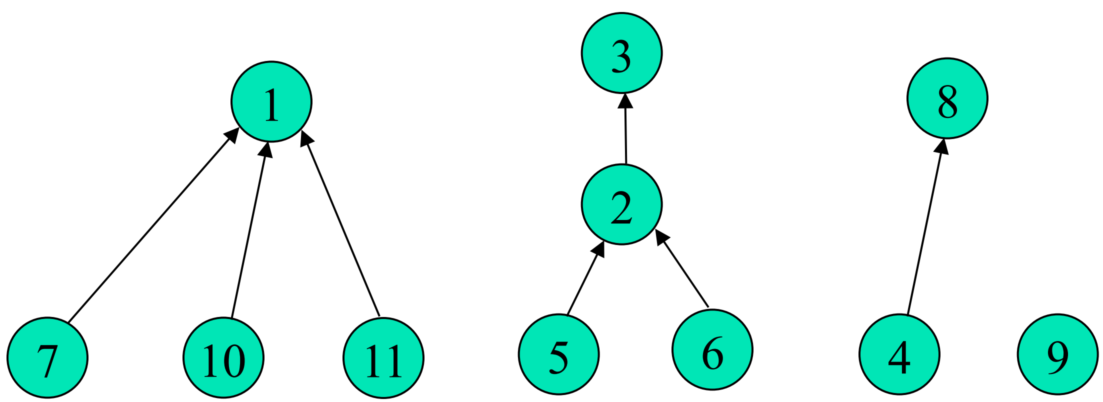
- 算法时间复杂度(按迭代次数计算)：$\Theta(kn)=\Theta(n)$
- 算法空间复杂度（十个表，每个表都是n）：$\Theta(10n)=\Theta(n)$

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。

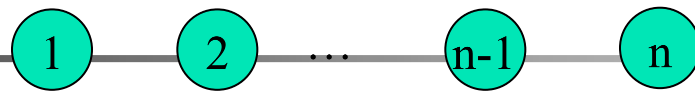


假如要执行如下计算任务：

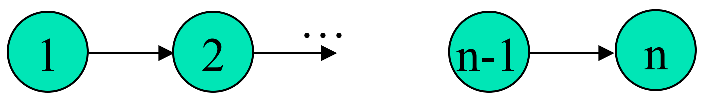
- FIND(x): 寻找包含元素x的集合的名字
- UNION(x,y): 将包含元素x和y的两个集合合并，重命名。

- 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},…,{n}



执行合并序列：UNION(1,2),UNION(2,3),…UNION(n-1,n).我们得到的结果是：

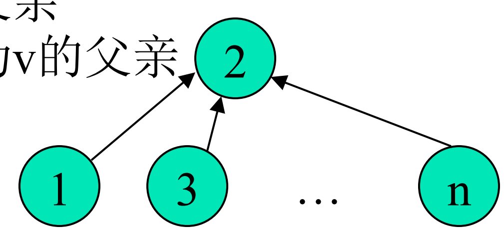


执行查找序列：FIND(1), FIND(2),…, FIND(N).需要比较的次数是：

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

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



武漢大學 WUHAN UNIVERSITY

```
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
```

引理 4.1 包括根节点 x 在内的树中节点的个数至少是 $2^{rank(x)}$。

参考书籍，采用归纳法证明

合并运算的时间复杂性和寻找运算的时间复杂性相同，都是 $O(\log n)$

m 次合并和寻找指令的交替执行序列的时间复杂性是 $O(m \log n)$

路径压缩

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)。

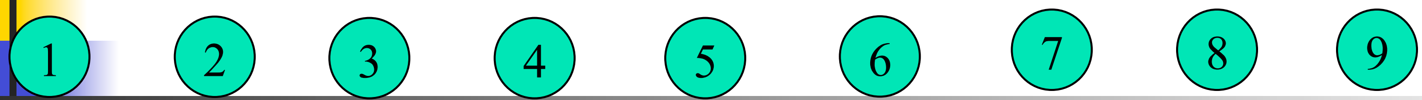
Algorithm: FIND

输入： 节点x

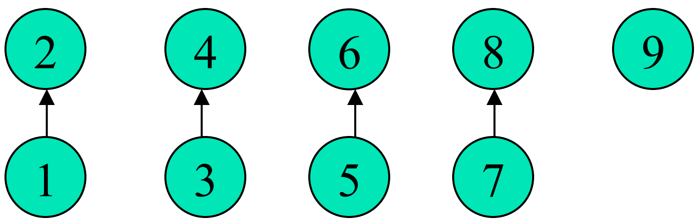
输出： root(x)和路径压缩后的树

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

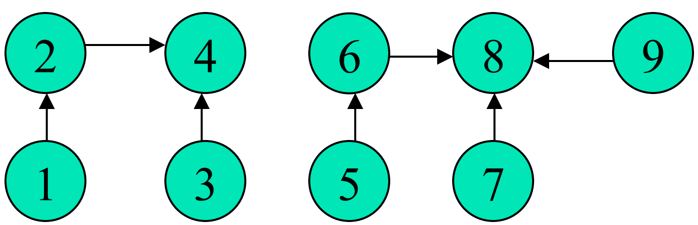
例：初始状态：{1},{2},…,{9}



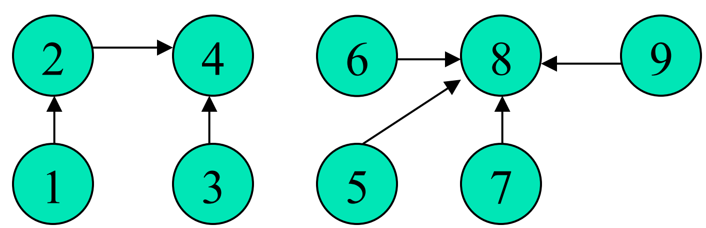
执行合并序列：UNION(1,2),UNION(3,4),UNION(5,6) ,UNION(7,8),得到的结果是：



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



继续执行：FIND(5)得到的结果是：

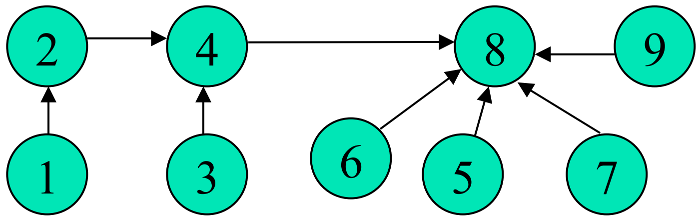


继续执行UNION(4,8)呢?

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：