数据结构

兵哥哥

2017 年 6 月 19 日

1 数据结构概念

数据结构的逻辑结构和物理结构（存储结构），我们主要研究的是存储结构。
数据结构两种存储方式：顺序存储和链式存储。
时间复杂度的概念，看 for 循环，$O(n)$

1.1 栗子

求整数的二分序列算法如下，该算法的渐近时间复杂度函数是$(O(log(n)))$。

```
void binary (int n) {
                while (n) {
                printf (“%d\n”, n);
                n=n/2;
        }
}
```

2 线性表

2.1 顺序表

顺序表（线性表的顺序表示）：用一组地址**连续的**存储单元依次存储线性表的数据元素。

2.1.1 顺序表的插入

线性表的插入需要移动元素的位置。

$$(a_1, a_2, a_3, \cdots, a_{i-1}, a_i, \cdots, a_n)$$
$$(a_1, a_2, a_3, \cdots, a_{i-1}, b, a_i, \cdots, a_n)$$

实现步骤

- 将第 n 至第 i 位的元素向后移动一个位置；
- 将要插入的元素写到第 i 个位置；
- 表长加 1。

实现主要程序

```
q=&(L.elem[i-1]);       //q赋为插入的位置
for(p=&(L.elem[L.length-1]));q<=p;--p)
*(p+1)=*p;//插入位置之后的元素后移
*q=e;//插入e
++L.length;//表长加1
return OK;
```

分析：插入一个元素需要移动的平均次数为 $\frac{n}{2}$，时间复杂度为 $O(n)$

2.1.2 顺序表的删除

$$(a_1, a_2, a_3, \cdots, a_{i-1}, a_i, \cdots, a_n)$$
$$(a_1, a_2, a_3, \cdots, a_{i-1}, a_{i+1}, \cdots, a_n)$$

实现步骤

- 将第 i+1 至第 n 位的元素向前移动一个位置；
- 表长减 1。

实现主要程序

```
p=&(L.elem[i-1]);     //p为被删除元素的位置
e=*p;                 //被删除的元素赋给e
q=L.elem+L.length-1     //表尾元素的位置
for（++p,p<=q;++p)
*(p-1)=*p;     //被删除元素之后的元素左移
--L.length;     //表长减1
return OK;
```

分析：插入一个元素需要移动的平均次数为 $\frac{n-1}{2}$，时间复杂度为 $O(n)$

2.1.3 顺序表的查找

```
int LocateElem_Sq(SqList L, ElemType e)
{
i=1;                  //i的初值为第1个元素的位序
```

```
p=L.elem; //p的初值为第1个元素的存储位置
while(i<=L.length && *p!=e)
{p++;          i++;}
if (i<=L.length)  return i; //找到满足条件的元素
else  return 0;  //没找到满足条件的元素
}
```

2.1.4 顺序表的优缺点

优点：可以快速地随机存取表中任一位置的元素。

缺点：插入和删除操作需移动大量元素

2.2 线性链表–单链表

线性链表：用一组任意的的存储单元存储线性表的数据元素（这组存储单元可以是连续的，也可以是不连续的）

结点包括数据域（存储数据元素信息的域）和指针域（存储直接后继存储位置的域）

2.2.1 单链表的插入

```
s->data=e;          //使新结点数据域的值为e
s->next=p->next; //将新结点插入L中
p->next=s;
```

2.2.2 单链表的删除

```
q=p->next;          //用一个指针q指向被删除结点
p->next=q->next;//删除第i个结点，即修改第i-1个结点的指针；
e=q->data;
free(q);          //释放第i个结点
```

单链表插入或删除一个结点时，仅需修改指针而不需要移动元素。时间复杂度为 $O(1)$

2.2.3 单链表优缺点

优点：插入、删除操作方便有效

缺点：需额外空间存储指针和额外的内存管理开销

	随机访问	插入操作	删除操作	内存空间
顺序表	yes,$O(1)$	$O(N)$	$O(n)$	受限
链表	no,$O(n)$	$O(1)$	$O(1)$	不受限

2.3 顺序表和单链表比较

2.4 栗子

1. 若某线性表中最常用的操作是取第 i 个元素和找第 i 个元素的前趋元素，则采用 (A) 存储方式最节省时间。

A．顺序表　　B．单链表

C．双链表　　D．单循环链表

2．设一个链表最常用的操作是在末尾插入结点和删除尾结点，则选用 (D) 最节省时间。

A. 单链表　　B. 单循环链表

C. 带尾指针的单循环链表　　D. 带头结点的双循环链表

3. 指针 p 指向循环单链表 L（带头结点）的首结点的条件是 (C)：

A.p==L　　B.p->next==L

C.L->next==p　　D.p->next==NULL

4. 指针 p 指向双向循环链表 L 的尾结点的条件是 (D)：

A.p==L　　B.p==NULL

C.p->prior==L　　D.p->next==L

3 栈和队列

3.1 栈

栈是限定仅在表尾进行插入和删除操作的线性表，因此，对栈来说，表尾断称为栈顶，表头端称为栈底。

栈的特点：后进先出 (LIFO)

栈在计算机中主要有两种基本的存储结构：顺序存储结构和链式存储结构。

利用一组地址连续的存储单元依次存放自栈底到栈顶的数据元素, 并用起始端作为栈底。

3.2 顺序栈的基本操作

```
/* ********** 元素入栈 push ********** */
//插入元素 e 为新的栈顶元素
*S.top++=e;    //*S.top=e;S.top++;
```

```
return OK;
/*****************读取栈顶值*******************/
if (S.base==S.top) return ERROR;
e=*(S.top-1);//注意要减1
return OK;
/*****************出栈*******************/
if (S.base==S.top) return ERROR;
e=*--S.top;      //S.top= S.top-1;e=*S.top;
return OK;
/*****************判空*******************/
if (S.base==S.top)   //栈空
/*****************判满*******************/
if (S.top-S.base>=S.stacksize)   //栈满
```

3.2.1 顺序栈的优缺点

优点：用顺序存储结构表示的栈并不存在插入删除数据元素时需要移动的问题。
缺点：栈容量难以扩充的弱点仍旧没有摆脱

3.2.2 栈的应用举例(了解记住就行)

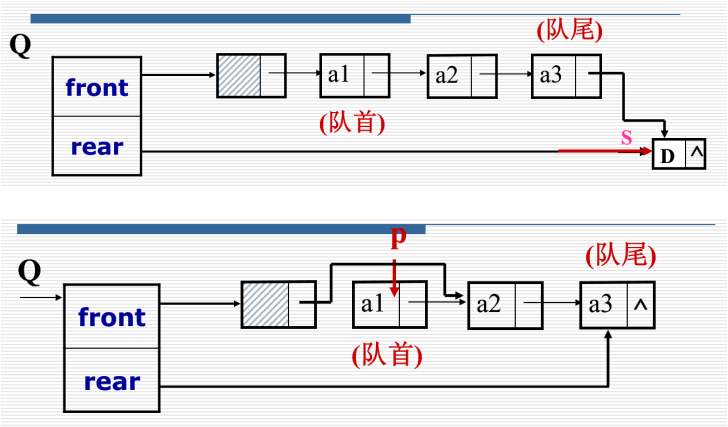
- 数制转换
- 括号匹配的检验
- 行编辑程序
- 迷宫求解
- 表达式求值
- 实施函数的调用（递归）

3.3 队列

队列是一种先进先出的线性表，它只允许在表的一端进行插入（队尾），而在另一端进行删除（队头）。

3.3.1 链队列–队列的链式表示与实现

```
/*****************插入*******************/
//插入元素e为Q的新的队尾元素
```




```
s= (QueuePtr)malloc(sizeof(Qnode));
s->data=D;                    s->next=NULL;
Q.rear->next = s;
Q.rear = s;
/*****************删除*******************/
//若队列不空，则删除Q的队头元素，并用e返回其值，并返回OK
if(Q.front==Q.rear)    return ERROR;
//如果队列为空则无法进行删除，返回ERROR
p=Q.front->next;          //令p指向队列Q的队头元素
e=p->data;                  //将队头元素的值取出并放入e
Q.front->next=p->next//删除队头指针元素
if(Q.rear==p) Q.rear=Q.front;
//如果删除的是队尾元素，则令队尾指针等于队头指针
free(p);          //释放队头元素所占空间
return OK;
```

3.3.2 循环队列–队列的顺序表示和实现

```
/***************入队操作**************/
//插入元素e为Q的新的队尾元素
//插入前应当先判断队列是否满？
if(Q.rear+1) % MAXQSIZE == Q.front) return ERROR;//队满
//插入动作分析
Q.base[Q.rear]=e;//将e插入到循环队列尾
Q.rear=(Q.rear+1) % MAXQSIZE;//修改列尾指针
return OK;
/***************出队操作**************/
//在删除前应当判断队列是否空？
```

```
11 if(Q.front == Q.rear) return ERROR;//队空
12 //删除动作分析;
13 e=Q.base[Q.front];//将循环队列头的元素值取出放入e
14 Q.front=(Q.front+1) % MAXQSIZE;//修改队头指针
15 return OK;
```

循环队列元素个数计算公式: $(N + rear - front)\%N$

3.3.3 队列的应用举例

- 离散事件的模拟

3.4 栗子

1. 一个队列的入队序列是 1，2，3，4，则出队顺序是 (B)

A 4,3,2,1　　B 1,2,3,4

C 1,4,3,2　　D 3,2,4,1

2. 循环顺序队列中是否可以插入下一个元素 (A)

A、与队头指针和队尾指针值有关

B、与队头指针有关，与队尾指针值无关

C、只与数组大小有关，与队头指针和队尾指针值无关

D、与曾经进行过多少次插入操作有关

提示：(rear+1)%M == front

3. 判断一个循环队列 QU（最大长度为 MaxSize) 为空的条件 (A)

A、QU.front==QU.rear

B、QU.front!=QU.rear

C、QU.front==(QU.rear+1)%MaxSize

D、QU.front!=(QU.rear+1)%MaxSize

4. 判断一个循环队列 QU（最大长度为 MaxSize) 为满的条件 (C)

A、QU.front==QU.rear

B、QU.front!=QU.rear

C、QU.front==(QU.rear+1)%MaxSize

D、QU.rear==(QU.front+1)%MaxSize

3.5 线性表的比较

线性表异同点汇总

	线性表	栈	队列
相同点	逻辑结构相同，都是线性的； 可以用顺序存储或链表存储； 栈和队列是两种特殊的线性表。		
不同点	任意合法位置插入、删除	**LIFO**	**FIFO**
	比较通用	函数调用 递归 …	用于离散事件模拟 OS作业调度 …

4 串

本章考 2-3 分，掌握定义和了解哪些应用即可。

串：是由零个或多个字符组成的有限序列，是数据元素为单个字符的特殊线性表。

$s = 'a_1a_2...a_n'$

子串在主串中的位置: 子串在主串中第一次出现时，子串的第一个字符在主串中的位置。

两个串相等：两个串的长度相等，并且各个对应位置的字符都相等时才相等。

4.1 串的应用举例

- 文本编辑
- 建立词索引表

4.2 栗子

1. 设有两个串 p 和 q，其中 q 是 p 的子串，求 q 在 p 中首次出现的位置的算法称为 (C)

A、求子串　　B、联接

C、匹配　　D、求串长

2. 串是一种特殊的线性表，其特殊性体现在 (B)

A、可以顺序存储　　B、数据元素是一个字符

C、可以链接存储　　D、数据元素可以是多个字符

3. 若串 S=“software”，其子串的个数是 (B)

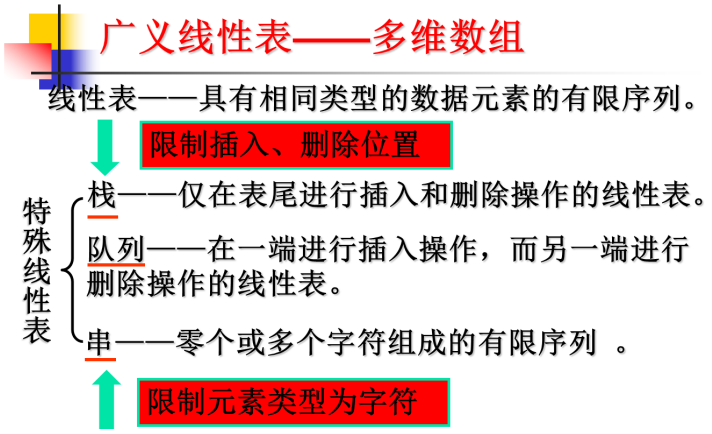
A、8　　B、37

C、36　　D、9

4. 当且仅当两个串的**长度**相等并且各个对应位置上的字符都**相等**时，这两个串相等。一个串中任意个连续字符组成的序列称为该串的**子**串，该串称为它所有子串的**主**串。

5.INDEX（‘DATASTRUCTURE’，‘STR’）=5

5 数组和广义表

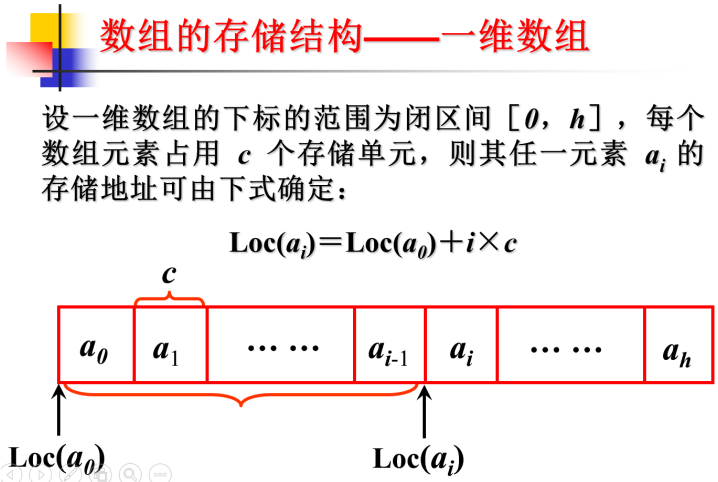


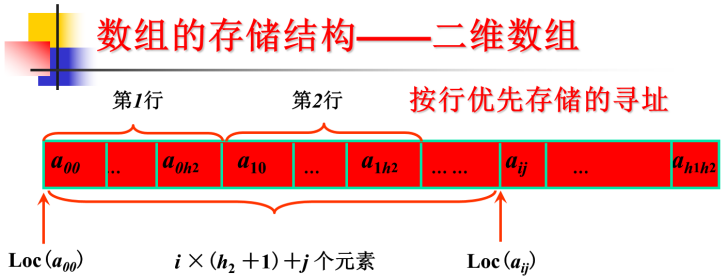
线性表——具有相同类型的数据元素的有限序列。

数组（线性表的推广）：是由一组类型相同的数据元素构成的有序集合，每个数据元素称为一个数组元素（简称为元素）。

数组没有插入和删除操作，不用预留空间，适合采用顺序存储。

5.1 数组的存储



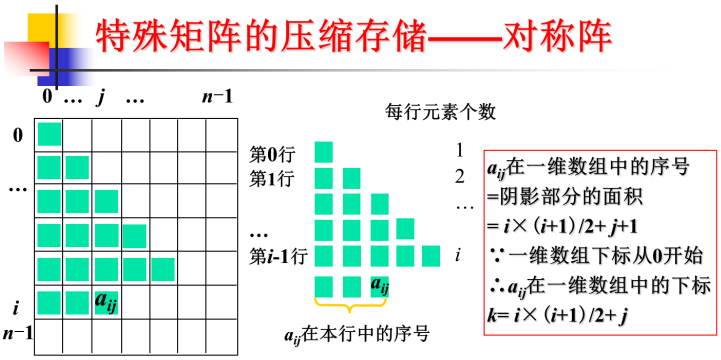


$$Loc(a_{ij}) = Loc(a_{00}) + (i \times (h2 + 1) + j) \times c$$

按列优先存储的寻址方法与此类似。

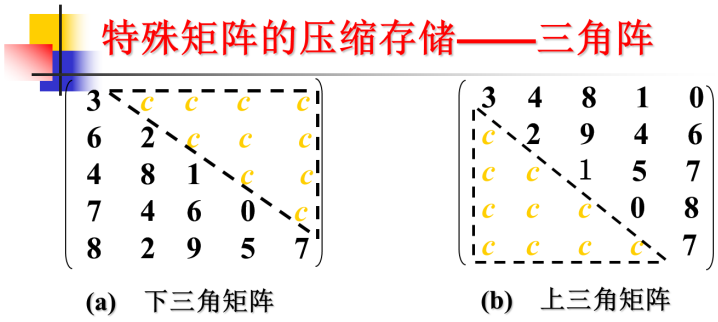
5.2 压缩存储

5.2.1 特殊矩阵的压缩存储——对称阵



(a) 下三角矩阵　　(b) 存储说明　　(c) 计算方法

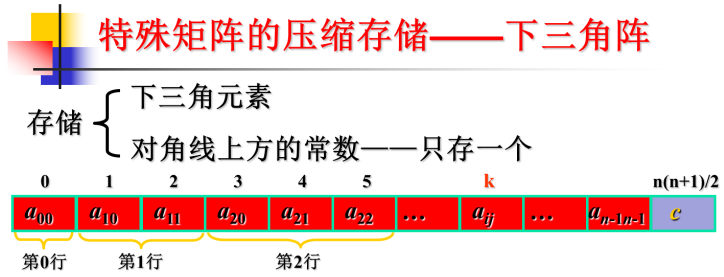
5.2.2 特殊矩阵的压缩存储——三角阵



(a)　下三角矩阵　　(b)　上三角矩阵

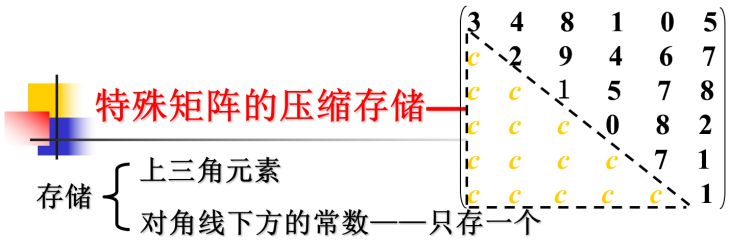
如何压缩存储？

只存储上三角（或下三角）部分的元素。



矩阵中任一元素a_{ij}在数组中的下标k与i、j的对应关系：

$$k=\begin{cases} i\times(i+1)/2+j & \text{当}i\geq j \\ n\times(n+1)/2 & \text{当}i<j \end{cases}$$



矩阵中任一元素a_{ij}在数组中的下标k与i、j的对应关系：

$$k=\begin{cases} i\times(2n-i+1)/2+j-i & \text{当}i\leq j \\ n\times(n+1)/2 & \text{当}i>j \end{cases}$$

5.2.3 矩阵的压缩存储——稀疏矩阵

将稀疏矩阵中的每个非零元素表示为：(行号，列号，非零元素值)——三元组

5.3 栗子

1. 已知数组 A[0..5,0..6] 的每个元素占 5 个字节，将其按列优先次序存储在起始地址为 1000 的内存单元中，则元素 A[5，5] 的地址是 (1175)

提示：$1000+(5\times6+5)\times5$

2. 有一个 100*90 的稀疏矩阵，非 0 元素有 10 个，设每个整型数占 2 字节，则用三元组表示该矩阵时，所需的字节数是 (B)。

A、60　　B、66

C、18000　　D、33

提示：$10\times3\times2+1\times3\times2=66$

3. 设有一个 10 阶的对称矩阵 A，采用压缩存储方式，以行序为主存储，a[1,1] 为第一元素，其存储地址为 1，每个元素占一个地址空间，则 a[8,5] 的地址为 (B)

A、13　　B、33

C、18　　D、40

提示：$7 \times (1+7)/2 + 5 = 33$

4. 设 A 是 $n \times n$ 的对称矩阵，将 A 的对角线及对角线上方的元素以列为主的次序存放在一维数组 B[1..n(n+1)/2] 中，对上述任一元素 a_{ij}(1≤i，j≤n，且 i≤j) 在 B 中的位置为 (B)

A、i(i-l)/2+j　　B、j(j-l)/2+i

C、j(j-l)/2+i-1　　D、i(i-l)/2+j-1

提示：特殊举例法

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 4 \\ 3 & 4 & 6 \end{bmatrix}$$

按照以列为主的的次序存放:$[1, 2, 5, 3, 4, 6]$，现在取 a_{23}, 它在其中的次序为：

$$\frac{j \times (j-1)}{2} + i = \frac{3 \times (3-1)}{2} + 2 = 5$$

5. 有一个二维数组 A[0:8,1:5], 每个数组元素用相邻的 4 个字节存储，存储器按字节编址，假设存储数组元素 A[0,1] 的第一个字节的地址是 0。若按行存储，则 A[3,5] 的第一个字节的地址是 (C)

A、28　　B、44

C、76　　D、92

提示：$0 + (3 \times 5 + 4) \times 4$

6. 数组 A[0..4,-1..-3,5..7] 中含有元素的个数 (B)

A、55　　B、45

C、36　　D、16

提示：$5 \times 3 \times 3 = 45$

7. 有一个二维数组 A[0:8,1:5], 每个数组元素用相邻的 4 个字节存储，存储器按字节编址，假设存储数组元素 A[0,1] 的第一个字节的地址是 0，若按列存储，则 A[2,4] 的第一个字节的地址是 (A)

A、116　　B、132

C、176　　D、184

提示：$0 + (3 \times 9 + 2) \times 4 = 116$

8. 有一个二维数组 A[1:6，0:7] 每个数组元素用相邻的 6 个字节存储，存储器按字节编址。假设存储数组元素 A[1，0] 的第一个字节的地址是 0，则存储数组 A 的最后一个元素的第一个字节的地址是 (B)

A、276　　B、282

C、283　　D、288

提示：$0 + (7 \times 6 + 5) \times 6 = 282$

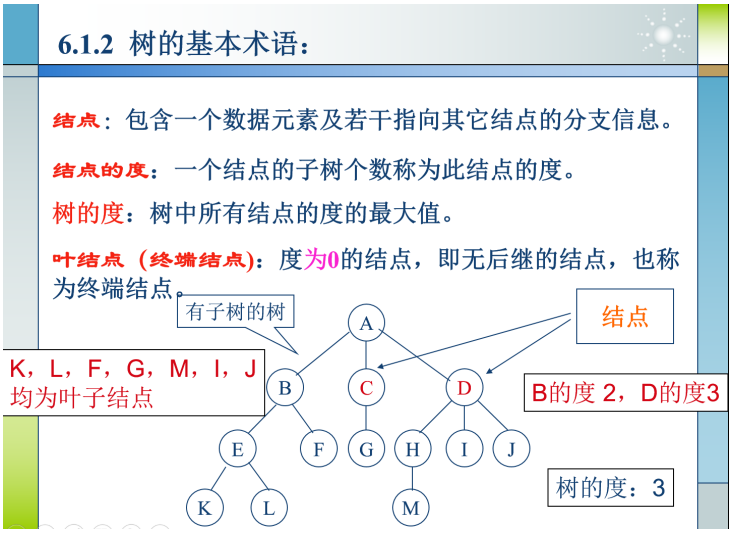
9. 假设以行序为主序存储二维数组 A=array[1..100，1..100]，设每个数据元素占 2 个存储单元，基地址为 10，则 LOC[5，5]=(B)

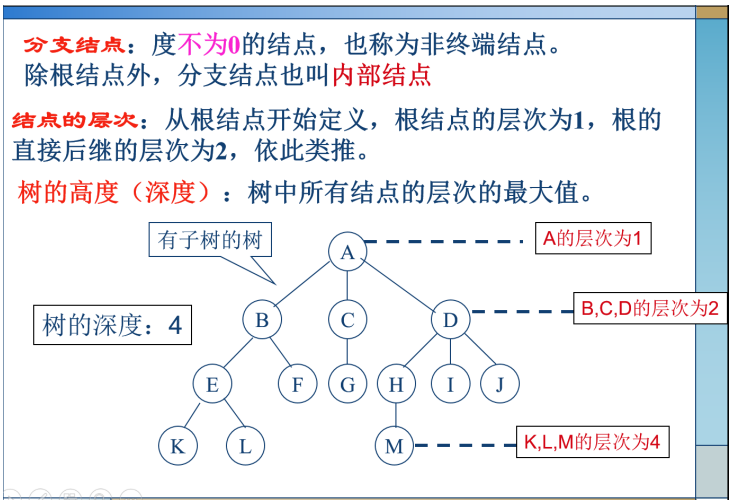
A、808　　B、818

C、1010　　D、1020

6　树和二叉树

本章是重点，应重点掌握





6.1　二叉树

二叉树的定义：二叉树是 $n(n \geq 0)$ 个结点的有限集，它或为空树 (n=0)，或由一个根结点和两棵分别称为左子树和右子树的互不相交的二叉树构成。

每个结点至多有二棵子树 (即不存在度大于 2 的结点)，二叉树的子树有左、右之分，且其次序不能任意颠倒

6.2　二叉树的性质

性质 1　在二叉树的第 i 层上至多有 2^{i-1} 个结点。

性质 2 深度为 k 的二叉树至多有 $2^k - 1$ 个结点。

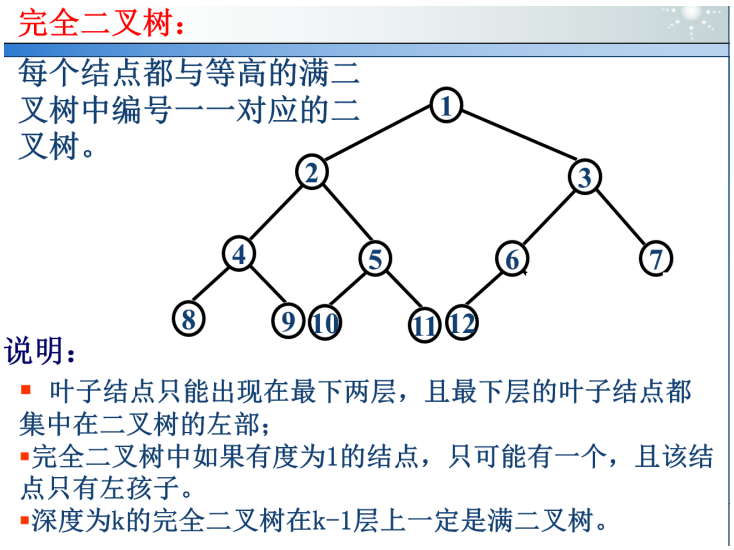
性质 3 对任何一棵二叉树，如果其终端结点（叶子结点）数为 n_0，度为 2 的结点数为 n_2，则 $n_0 = n_2 + 1$（即是度为 0 的结点数永远比度为 2 的结点数多 1）

6.3 满二叉树和完全二叉树

满二叉树：深度为 k 且有 $2^k - 1$ 个结点的二叉树

满二叉树的特点

- 满二叉树在同样深度的二叉树中结点个数最多
- 满二叉树在同样深度的二叉树中叶子结点个数最多



完全二叉树的性质

- 具有 n 个结点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor + 1$
- 对于具有 n 个结点的完全二叉树，如果按照从上到下和从左到右的顺序对二叉树中的所有结点从 1 开始顺序编号，则对于任意的序号为 i 的结点有：
  - 若 i>1, 则 i 的双亲结点为 $\lfloor i/2 \rfloor$
  - 若 2i≤n, 则 i 结点的左孩子结点为 2i
  - 若 2i+1≤n, 则 i 的右孩子结点为 2i+1

6.4 栗子

1. 若一棵二叉树有 10 个度为 2 的结点，5 个度为 1 的结点，则二叉树结点总数为 (B)

A.9　　B.26

C.15　　D. 不确定 2. 一棵完全二叉树上有 1001 个结点，其中叶子结点的个数为 (D)

A.250　　B.500

C.254　　D. 以上答案都不对

3. 一棵二叉树的高度为 h，所有结点的度为 0 或为 2，则这棵二叉树最少有 (B) 个结点

A.2h　　B.2h － 1

C.2h ＋ 1　　D.h ＋ 1

6.5　二叉树的存储结构（了解）

顺序存储和链式存储

6.6　二叉树的遍历–必须熟练掌握

先序遍历二叉树的操作定义为：

- 访问根结点；
- 先序遍历左子树；
- 先序遍历右子树。

中序遍历二叉树的操作定义为：

- 中序遍历左子树；
- 访问根结点；
- 中序遍历右子树。

后序遍历二叉树的操作定义为：

- 后序遍历左子树；
- 后序遍历右子树；
- 访问根结点。

说明：对于先序、中序、后序遍历序列中：

由先序和中序遍历结果可以唯一确定一棵二叉树 (DLR,LDR)。

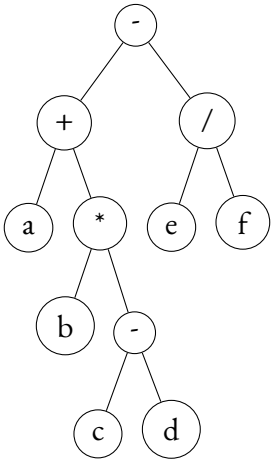
由后序和中序遍历结果可以唯一确定一棵二叉树 (LRD,LDR)。

由先序和后序遍历结果不能唯一确定一棵二叉树 (DLR,LRD)。

若一棵二叉树的先序、中序序列相同，则该二叉树的形态为：

空树、只有根结点、右单支

若一棵二叉树的后序、中序序列相同，则该二叉树的形态为：
空树、只有根结点、左单支
若一棵二叉树的先序、后序序列相同，则该二叉树的形态为：
空树、只有根结点



先序遍历：-+a*b-cd/ef　　中序遍历：a+b*c-d-e/f
后序遍历：abcd-*+ef/-　　层次遍历：-+/a*efb-cd

6.7 树和森林

6.7.1 树的三种存储结构

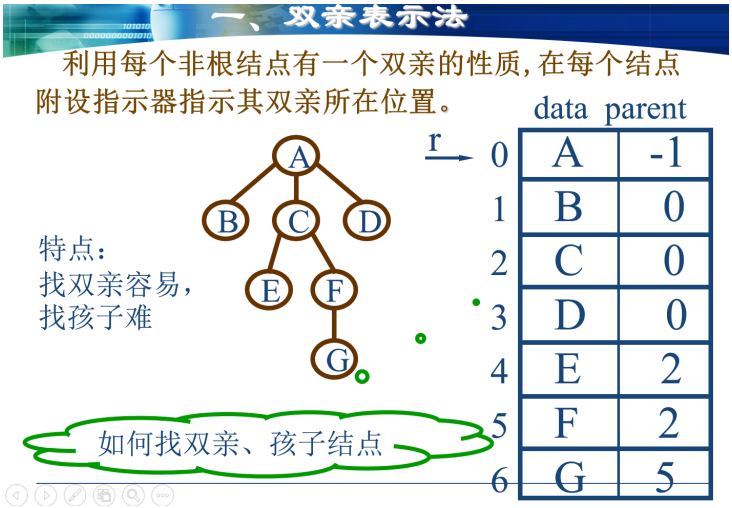


图 1: 双亲表示法

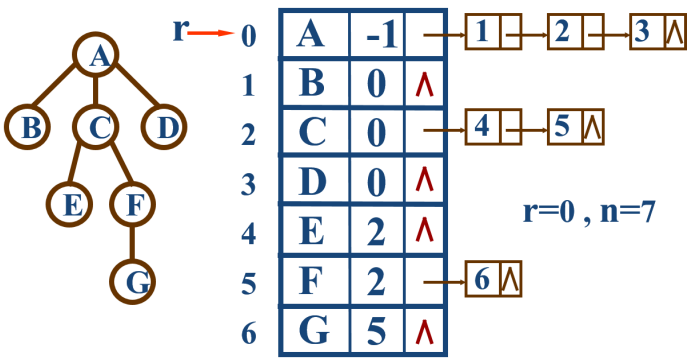


图 2: 孩子表示法

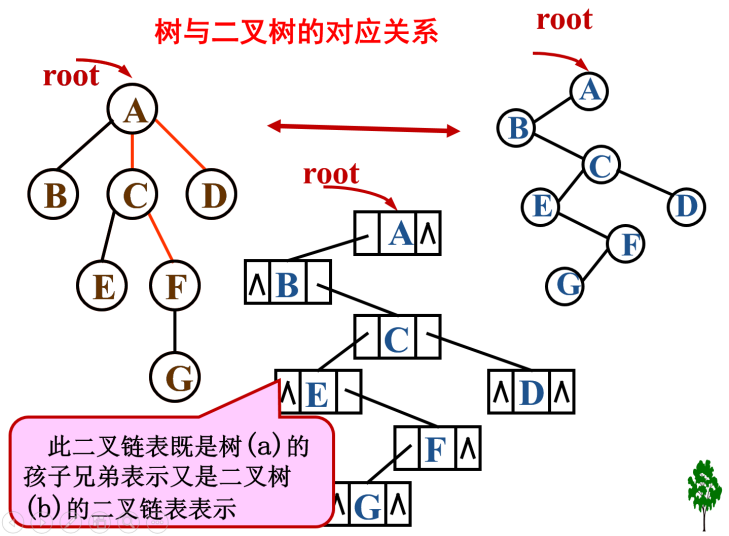


图 3: 树的二叉链表 (孩子-兄弟）存储表示法

链表中结点的两个链域分别指向该结点的第一个孩子结点和下一个兄弟结点。

6.8 森林与二叉树的转换

6.8.1 树转化为二叉树

1. (连兄弟) 在树中个兄弟之间加一连线；
2. (断父子) 对于任一结点，只保留它与最左孩子之间的连线；
3. (转一转) 分别以每个结点的第一个孩子结点为轴心，将其右边兄弟结点顺时针转 45°

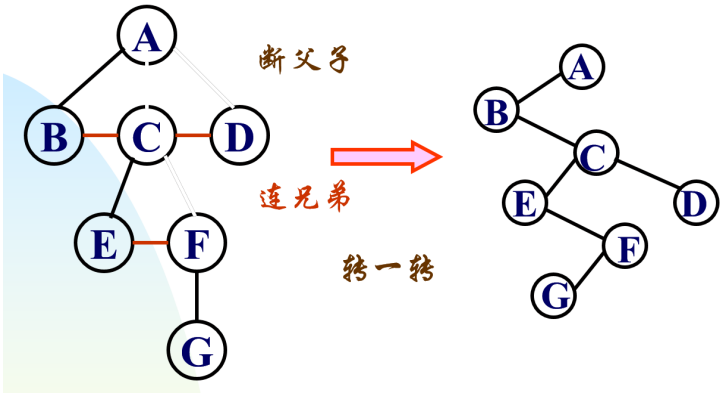


图 4: 树转换为二叉树

6.8.2 二叉树转换为树

1. (连祖孙) 将结点与其左孩子的右子孙连接；
2. (断父子) 对于任一结点，只保留它与左孩子之间的连线；
3. (抖一抖) 将结点按层次排列，形成树结构。

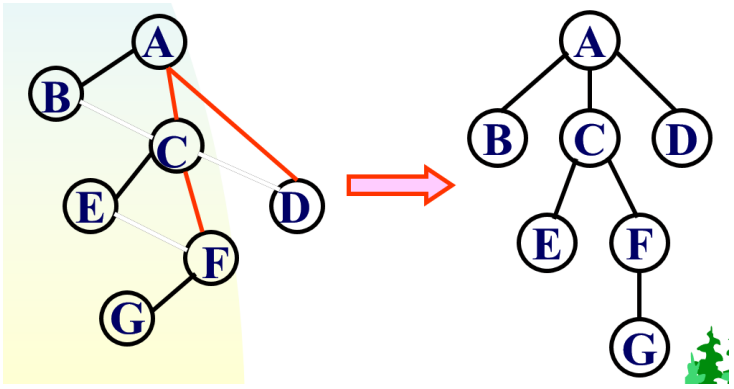


图 5: 二叉树转换为树

6.8.3 森林转化为二叉树

- 将森林中的每一棵树依次转换成相应的二叉树;
- 将第二棵作为第一棵二叉树的根结点的右子树连接起来，将第三棵又作为第二棵的右子树连接起来..., 直至把所有的二叉树连接成一棵二叉树。

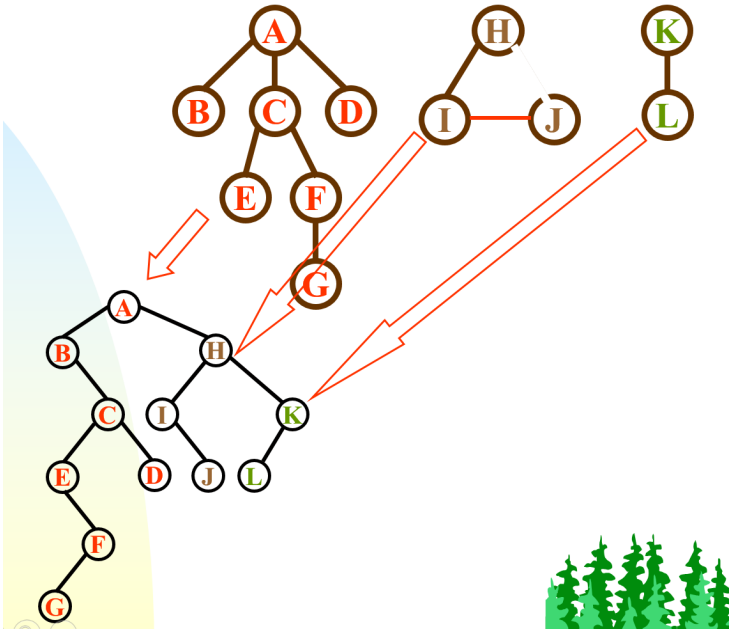


图 6: 森林转换成二叉树

6.8.4 二叉树转换成森林

- 将二叉树中根结点与其右孩子连线，及沿右分支搜索到的所有右孩子间连线全部抹掉，使之变成孤立的二叉树;
- 将森林中的每一棵二叉树依次转换成树。

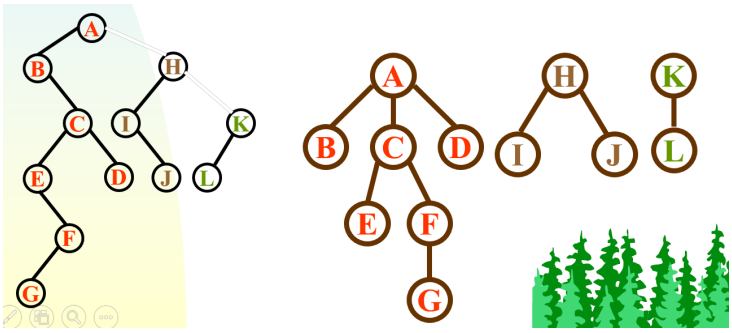


图 7: 二叉树转换成森林

6.9 树和森林的遍历

图 8: 树的遍历

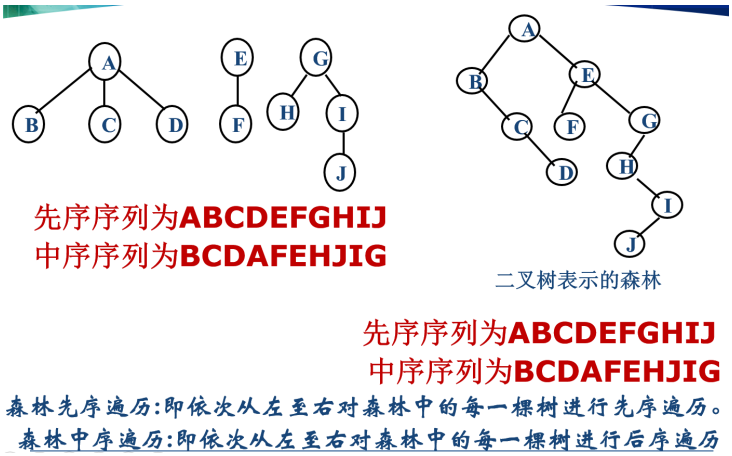


图 9: 森林的遍历

6.10 树与森林的遍历和二叉树的遍历的对应关系

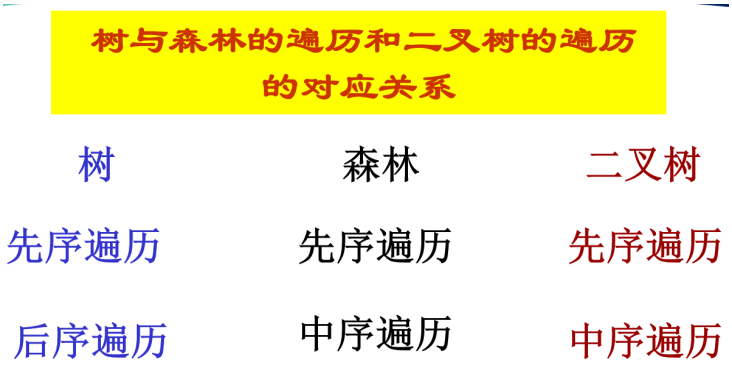


图 10: 森林的遍历

6.11 霍夫曼树及其应用

最优二叉树 (赫夫曼树) 的概念：带权路径长度 WPL 最小的二叉树称为霍夫曼树

6.12 霍夫曼树的构造

1. 在 F 中选取两棵根的权值最小的树作为左、右子树，构造一棵新二叉树，置新二叉树根的权值 = 左、右子树根结点权值之和；(每次新增一个结点)
2. 从 F 中删除这两棵树, 并将新树加入 F;(每次少一棵树)
3. 重复 2、3，直到 F 中只含一颗树为止。(要重复 n-1 次)

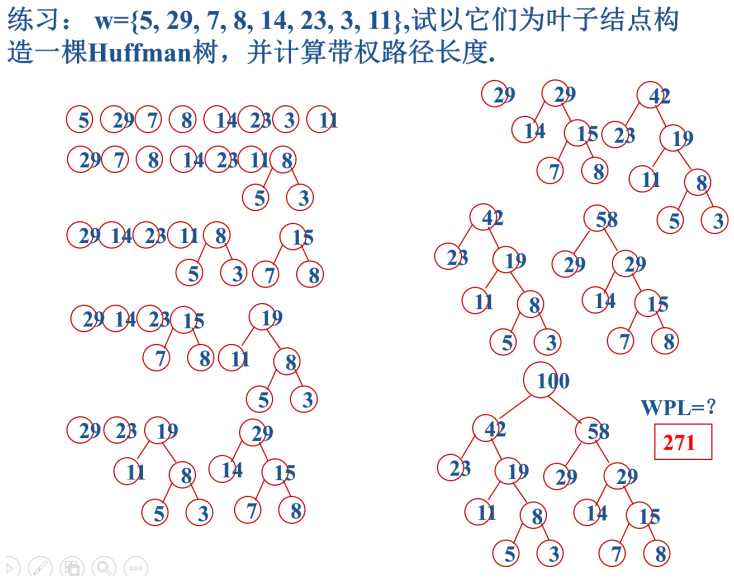


图 11: 霍夫曼树的构造

6.13 电报编码

某通讯系统只使用 8 种字符 a、b、c、d、e、f、g、h，其使用频率分别为 0.06, 0.28, 0.07, 0.08, 0.14, 0.23, 0.03, 0.11，利用赫夫曼树设计一种前缀编码：

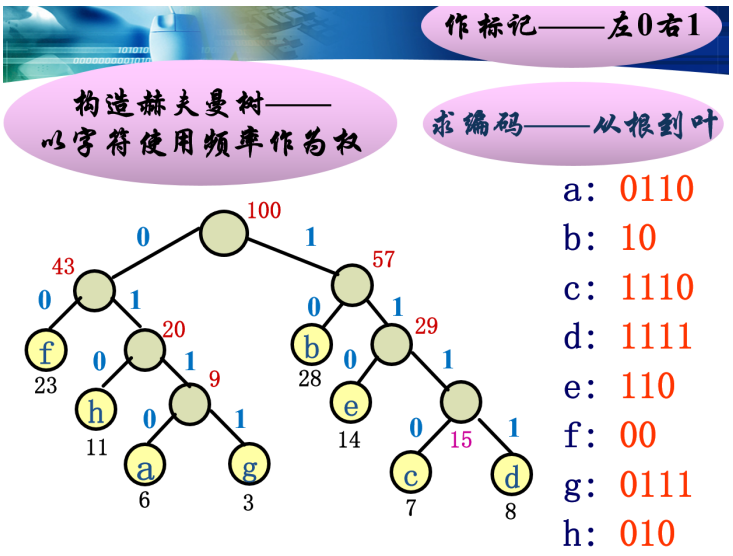


图 12: 霍夫曼树的应用–电报编码

7 图

7.1 图的基本术语

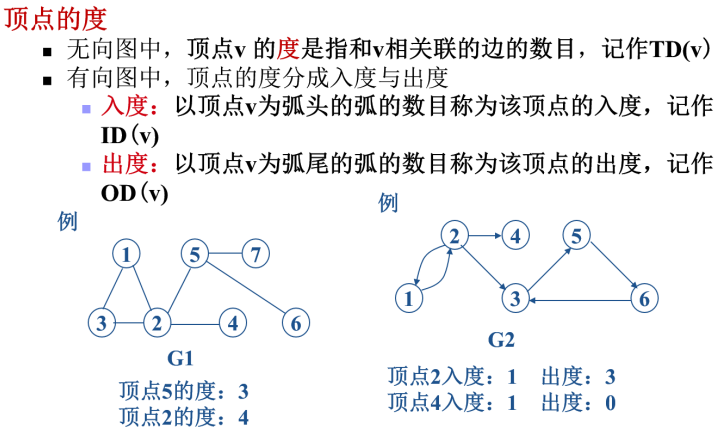


图 13: 度

在具有 n 个顶点、e 条边的无向图 G 中，各顶点的度之和是边数之和 e 的两倍

在具有 n 个顶点、e 条边的有向图 G 中，各顶点的入度之和等于各顶点的出度之和等于边数之和 e

无向图中，v_i 到 v_j 有路径，则称是连通的，无向图中极大连通子图称为连通分量

有向图中，任意的，从 v_i 到 v_j 和从 v_j 到 v_i 有路径，则称为强连通图，有向图的极大强连通子图称做有向图的强连通分量

注意：给一个无向图 (有向图) 能判别连通分量 (强连通分量)

7.2 图的存储结构

邻接矩阵，这个比较简单，能掌握怎么求入度、出度就 OK

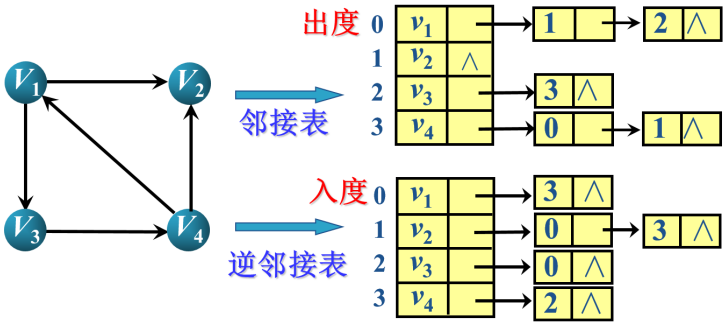


图 14: 邻接表

7.3 图的遍历

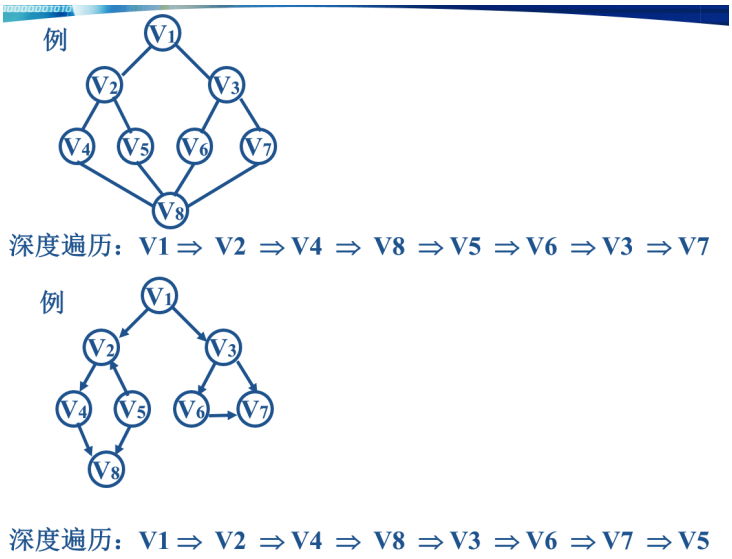


图 15: 深度优先遍历–递归

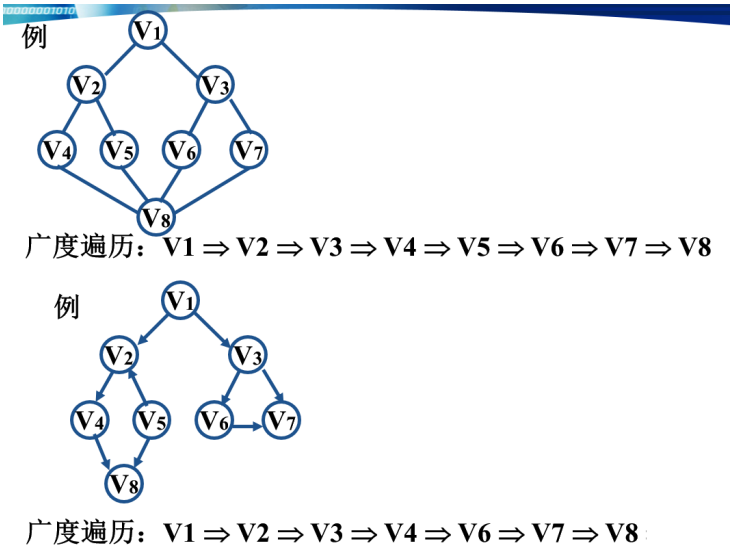


图 16: 广度优先遍历–队列

7.4 最小生成树

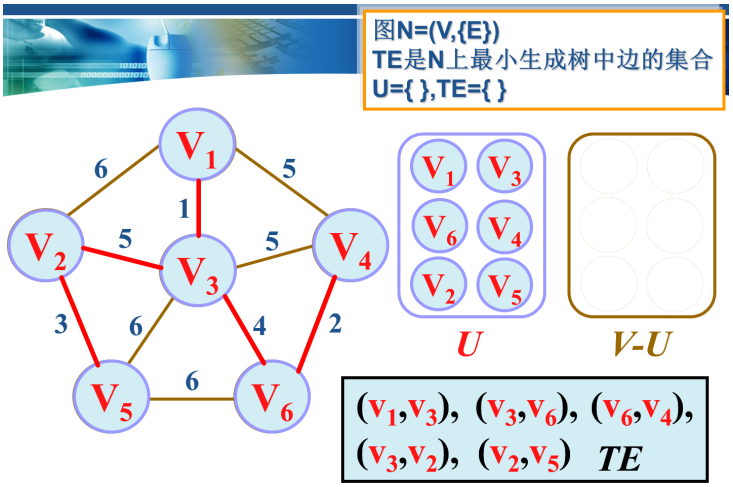


图 17: prim 算法–时间复杂度为 $O(n^2)$

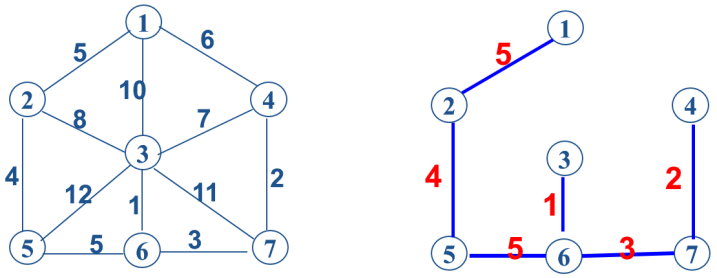


图 18: kruskal 算法–时间复杂度为诶 $O(eloge)$,(e 为网中边的数目)

7.5 有向无环图 (directed acycline graph，DAG 图) 应用–拓扑排序

AOV 网定义：顶点表示活动，弧表示活动间优先关系的有向图

拓扑排序：如果图中从 V 到 W 有一条有向路径，则 V 一定排在 W 之前，满足此条件的顶点序列称为一个拓扑序。获得一个拓扑序的过程就是拓扑排序。

1. 从有向图中选一个无前驱的顶点输出；
2. 将此顶点和以它为起点的弧删除；
3. 重复（1）、（2），直到不存在无前驱的顶点；
4. 若此时输出的顶点数小于有向图中的顶点数，则说明有向图中存在回路，否则输出的顶点的顺序即为一个拓扑序列。

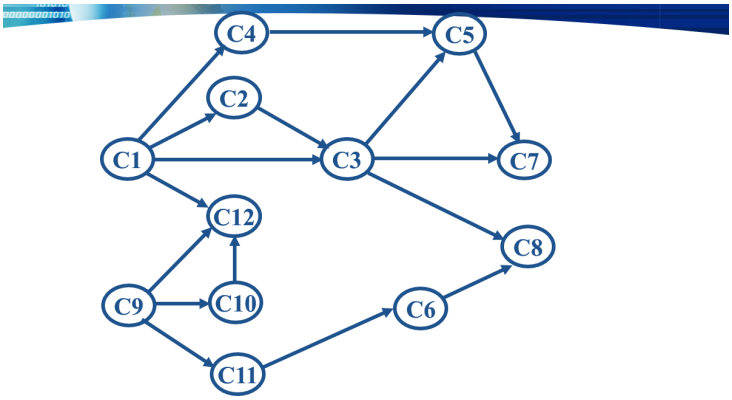


图 19: 拓扑排序

7.6 栗子

1. 以下说法正确的是 (B)

A. 连通分量是无向图中的极小连通子图。

B. 强连通分量是有向图中的极大强连通子图

C. 在一个有向图的拓扑序列中，若顶点 a 在顶点 b 之前，则图中必有一条弧 <a,b>

D. 对有向图 G，如果从任意顶点出发进行一次深度优先或广度优先搜索能访问到每个顶点，则该图一定是完全图。

2. 设无向图的顶点个数为 n，则该图最多有 (B) 条边

A.$n-1$ B.$n(n-1)/2$

C.$n(n+1)/2$ D.n^2

有向图最多有 $n(n-1)$ 条边

3. 要连通具有 n 个顶点的有向图，至少需要 (B) 条边

A.n-1　　B.n

C.n+1　　D.2n

4. 已知有向图 $G=(V,E)$, 其中 $V=\{V_1,V_2,V_3,V_4,V_5,V6,V_7\}$, $E=\{<V_1,V_2>,<V_1,V_3>,<V1,V4>,<V2,V5>,<V3,V5>,<V3,V6>,<V4,V6>,<V5,V7>,<V6,V7>\}$,G 的拓扑序列是 (A)

A.$V1,V3,V4,V6,V2,V5,V7$　　B.$V1,V3,V2,V6,V4,V5,V7$

C.$V1,V3,V4,V5,V2,V6,V7$　　D.$V1,V2,V5,V3,V4,V6,V7$

5. 在有向图 G 的拓扑序列中，若顶点 Vi 在 Vj 之前，则下列情形不可能出现的是 (D)

A.G 中有弧 <Vi,Vj>

B.G 中有一条从 Vi 到 Vj 的路径

C.G 中没有弧 <Vi,Vj>

D.G 中有一条从 Vj 到 Vi 的路径

8　查找

8.1　静态查找

8.1.1　顺序表查找

顺序查找适用于有序表和无序表，存储结构是顺序存储和线性链表均可

顺序查找过称为：从表中最后一个记录开始，逐个进行记录的关键字和给定值的比较，若某个记录的关键字和给定值比较相等，则查找成功，反之，若直到第一个记录，其关键字和给定值都不等，则表明表中没有所查记录，查找不成功。

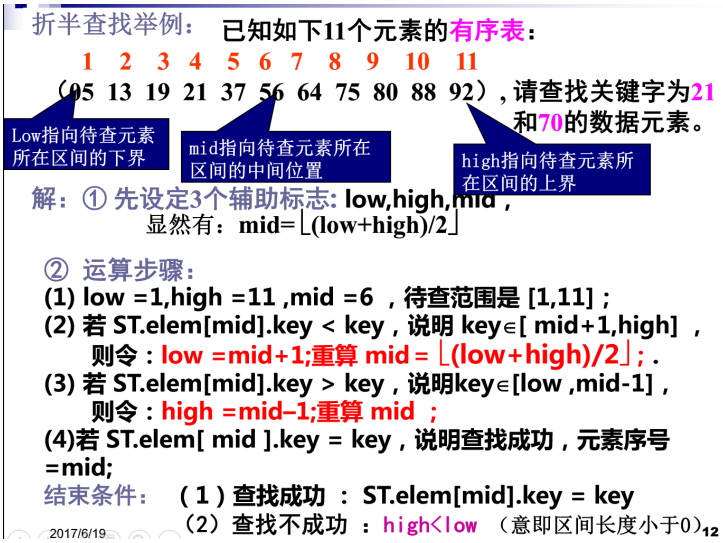
```
ST.elem[0].key =key; //“哨兵”
for( i=ST.length; ST.elem[i].key!=key;  --i ); //从后往前找
return i; //找不到时，i为0
```

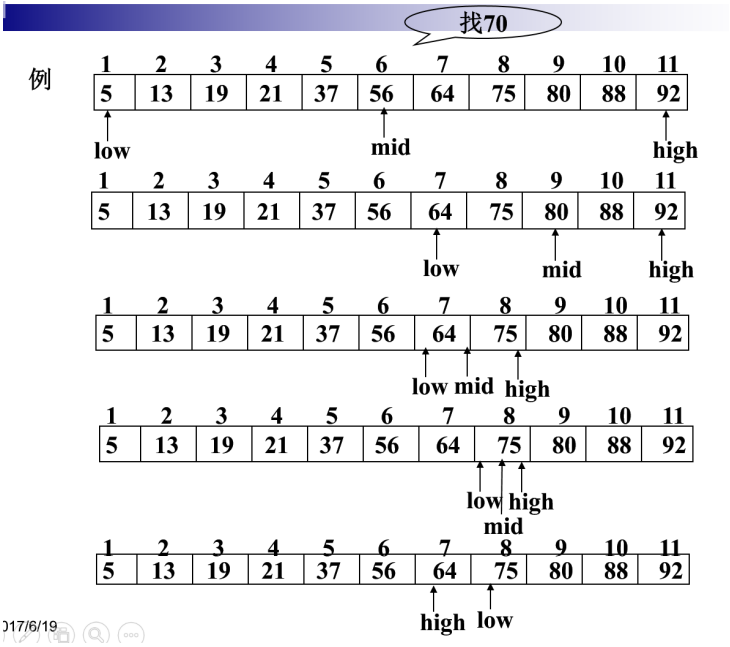
查找成功的平均查找长度：ASL=(1+2+3+...+n)/n=(1+n)/2，时间复杂度为 $O(n)$

8.1.2　有序表的查找-折半查找

查找效率较顺序查找高，但只适合有序表，且限于顺序存储结构 (对线性链表无法有效的进行折半查找)

折半查找的查找过程：先确定待查记录所在的范围，然后逐步缩小范围直到找到或找不到该记录为止。






```
int Search_Bin(SSTable ST, KeyType key){
low=1;  high=ST.length;  //置区间初值
while (low<=high){//循环查找
mid = (low+high)/2;
if (key== ST.elem[mid].key)  return mid;//找到待查元素
else if (key<ST.elem.[mid].key) high=mid-1
//继续在前半区间进行查找
else low = mid+1;//继续在后半区间进行查找
}
return  0;//顺序表中不存在待查元素
}
```

折半查找法在查找成功时和给定值进行比较的关键字个数至多为：$\lfloor \log_2 n \rfloor + 1$，假设有序

表的长度为 $n=2^h-1$(树的深度为 h) 查找成功的平均查找长度为:$ASL=\frac{1}{n}\sum_{j=1}^{h} j\cdot 2^{j-1}$，时间复杂度为 $O(\log_2{}^n)$

8.2 动态查找

二叉排序树或是一棵空树，或是具有下列性质的二叉树：

1. 若它的左子树不空，则左子树上所有结点的值均小于它的根结点的值
2. 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值
3. 若它的右子树不空，则右子树上所有结点的值均大于它的根结点的值

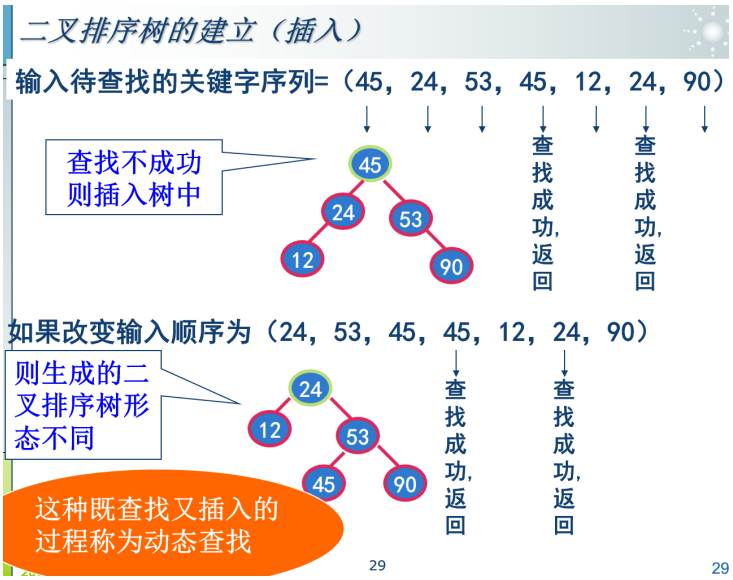
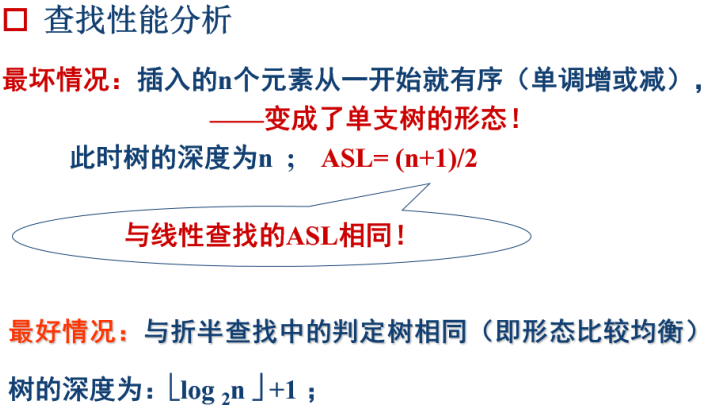


图 20: 二叉排序树的插入



二叉排序树的删除原则：在二叉排序树上删除一个结点后依旧要保持二叉排序树的特性

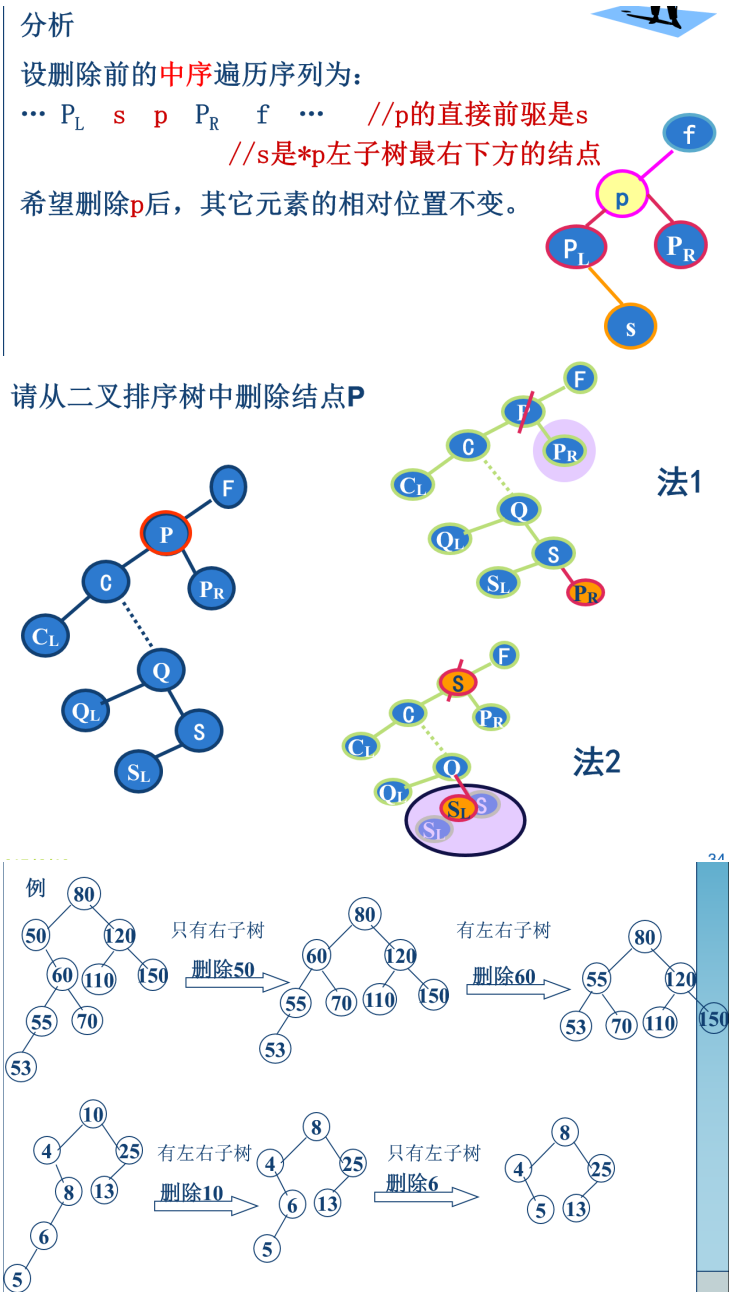


图 21: 二叉排序树的删除

8.3 哈希查找

8.3.1 开放定址法（开地址法）

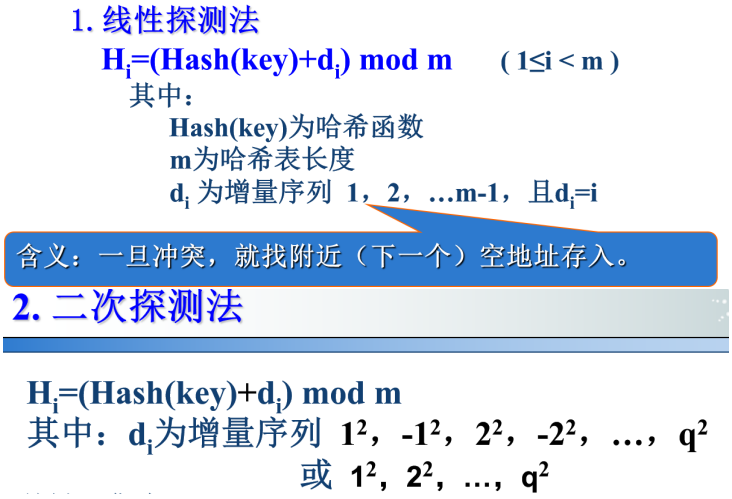


图 22: 开放地址法

8.3.2 链地址法（拉链法）

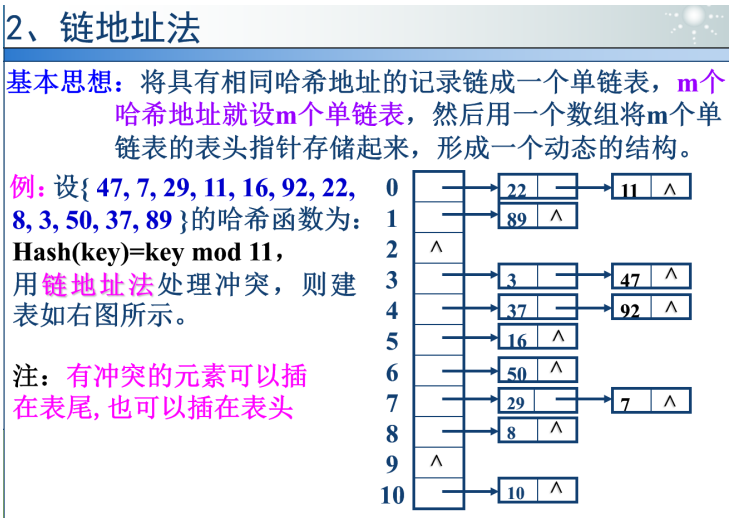


图 23: 链地址法

8.4 栗子

1. 已知一组关键字 (19,14,23,1,68,20,84,27,55,11,10,79)，哈希函数为：H(key)=key MOD 13，哈希表长为 m=16，设每个记录的查找概率相等，求查找成功时的 ASL。

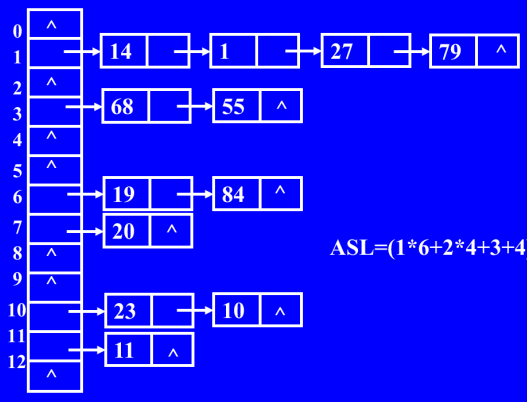
(1) 用线性探测再散列处理冲突，即Hi=(H(key)+di) MOD m

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	14	1	68	27	55	19	20	84	79	23	11	10			

H(19)=6
H(14)=1
H(23)=10
H(1)=1 冲突，H1=(1+1) MOD13=2
H(68)=3
H(20)=7
H(84)=6 冲突，H1=(6+1)MOD13=7
冲突，H2=(6+2)MOD13=8
H(27)=1 冲突，H1=(1+1)MOD13=2
冲突，H2=(1+2)MOD13=3
冲突，H3=(1+3)MOD13=4
H(55)=3 冲突，H1=(3+1)MOD13=4
冲突，H2=(3+2)MOD13=5
H(11)=11
H(10)=10 冲突，H1=(10+1)MOD13=11
冲突，H2=(10+2)MOD13=12
H(79)=1 冲突，H1=(1+1)MOD13=2
冲突，H2=(1+2)MOD13=3
冲突，H3=(1+3)MOD13=4
冲突，H4=(1+4)MOD13=5
冲突，H5=(1+5)MOD13=6
冲突，H6=(1+6)MOD13=7
冲突，H7=(1+7)MOD13=8
冲突，H8=(1+8)MOD13=9
ASL=(1*6+2+3*3+4+9)/12=2.5

关键字(19,14,23,1,68,20,84,27,55,11,10,79)

(2) 用链地址法处理冲突



ASL=(1*6+2*4+3+4)/12=1.75

2. 设哈希函数H（k）=3K mod 11，散列地址空间为0～10，对关键字序列（32,13,49,24,38,21,4,12）按下述两种解决冲突的方法构造哈希表。

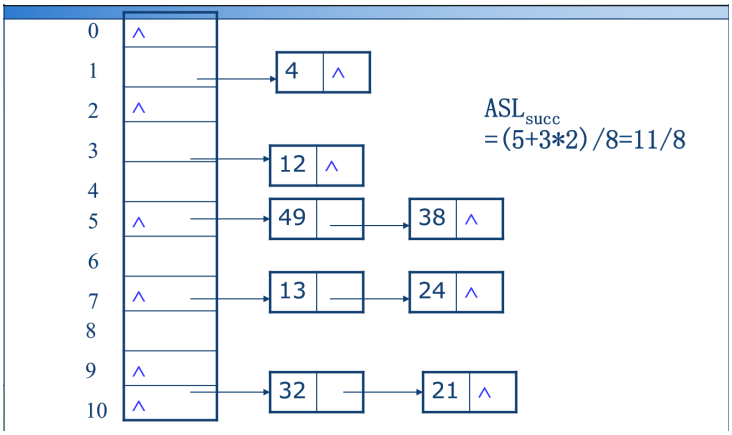
（1）线性探测再散列

（2）链地址法

并分别求出等概率下查找成功时的平均查找长度ASL。

散列地址	0	1	2	3	4	5	6	7	8	9	10
关键字		4		12	49	38	13	24	32	21	
比较次数		1		1	1	2	1	2	1	2	

ASL_{succ} =（1+1+1+2+1+2+1+2）/8=11/8

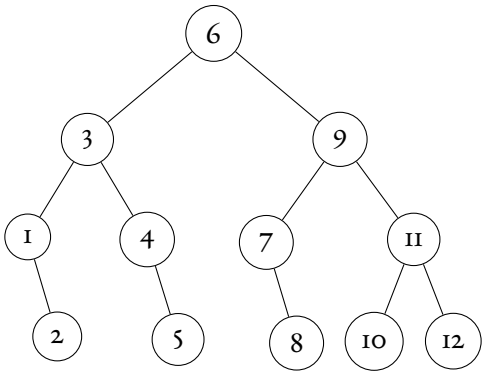


3. 具有 12 个关键字的有序表，折半查找的平均查找长度 (A)

A. 3.1　　B.4

C. 2.5　　D. 5

提示：$(1+2\times2+4\times3+5\times4)/12\approx3.1$



4. 折半查找的时间复杂性为 (D)

A. $O(n_2)$　　B. $O(n)$

C. $O(nlogn)$　　D. $O(logn)$

5. 顺序查找适合于存储结构为 (B) 的线性表

A. 散列存储　　B. 顺序存储或链式存储

C. 压缩存储　　D. 索引存储

6. 在顺序表（8，11，15，19，25，26，30，33，42，48，50）中，用二分查找法找关键字 20，需作比较4 次。

7. 设有一组记录的关键字为 {19，14，23，1，68，20，84，27，55，11，10，79}，用链地址法构造散列表，散列函数为 H（key）=key MOD 13, 散列地址为 1 的链中有 (D) 个记录。

A.1　　B.2

C.3　　D.4

提示：链地址为 1 的有：14，1，27，79

9 内排序

排序的时间开销可用算法执行中的数据比较次数与数据移动次数来衡量

9.1 插入排序

9.1.1 直接插入排序

```
void    InsertSort (SqList &L){
//对顺序表L作直接插入排序
for(i=2; i<=L.length; ++ i)
if (L.r[i] < L.r[i -1])
{  L.r[0]=L.r[i]; //“哨兵”
L.r[i]=L.r[i-1];
for(j=i-2; L.r[0] < L.r[j]; j--)//从后向前比较直到发现不比L.r[0]大的元素为
L.r[j+1]=L.r[j]; //记录后移
L.r[j+1]=L.r[0]; //插入到正确位置
}
}//InsertSort
```

9.1.2 希尔排序

先取一个正整数 $d_1<n$，把所有相隔 d_1 的记录放一组，组内进行直接插入排序；然后取 $d_2<d_1$，重复上述分组和排序操作；直至 $d_i=1$，即所有记录放进一个组中排序为止。

例 1：对下列序列采用希尔排序

49 38 65 97 76 13 27 49 55 04

第一趟希尔排序增量为 5

排序结果：13　27　49　55　04　49　38　65　97　76

第二趟希尔排序增量为 3

排序结果：13　04　49　38　27　49　55　65　97　76

第二趟希尔排序增量为 1

排序结果：04　13　27　38　49　49　55　65　76　97

例 2：已知序列 {70,83,100,65,10,32,7,9}, 请给出采用直接插入排序法对该序列进行升序排序时的每一趟的结果。解：初始:(70), 83,100,65,10,32,7,9

1 趟:(70,83), 100,65,10,32,7,9

2 趟:(70,83,100), 65,10,32,7,9

3 趟:(65,70,83,100),10,32,7,9

4 趟:(10,65,70,83,100), 32,7,9

5 趟:(10, 32,65,70,83,100), 7,9

6 趟:(7,10, 32,65,70,83,100), 9

7 趟:(7,9, 10, 32,65,70,83,100)

例 3. 已知序列 {70,83,100,65,10,32,7,9}, 采用希尔排序法（增量依次为 5，3，1），写出每一趟的结果。

第一趟结果 {32，7，9，65，10，70，83，100}

第二趟结果 {32，7，9，65，10，70，83，100}

第三趟结果 {7，9，10，32，65，70，83，100}

9.2　交换排序

9.2.1　冒泡排序

冒泡排序的基本方法是：设待排序对象序列中的对象个数为 n，最多作 n-1 趟排序。在第 i 趟中相邻两个元素进行比较，如果逆序就交换

原始	第 1 次	第 2 次	第 3 次	第 4 次	第 5 次
8	6	6	3	2	2
6	8	3	2	3	3
9	3	2	6	6	6
3	2	7	7	7	7
2	7	8	8	8	8
7	9	9	9	9	9

```
1 void BubbleSort(SqList &L) {
2 for (i=1;i<L.length;i++){
3 Exchange = false;
```

```
4  for (j = 1;  j <= L.length-i;  j++)
5  if (L.r[j+1] < L.r[j]) {
6  L.r[j+1] ⟷ L.r[j];
7  Exchange = true
8  }
9  if (Exchange == false) return;
10 }
11 }//BubbleSort
```

9.2.2 快速排序

快速排序方法的基本思想是任取待排序对象序列中的某个对象 (常取第一个对象) 作为枢轴 (pivot)，按照该对象的关键字大小，将整个对象序列划分为左右两个子序列：

1. 左侧子序列中所有对象的关键字都小于枢轴对象的关键字，右侧子序列中所有对象的关键字都大于枢轴对象的关键字
2. 枢轴对象则排在这两个子序列中间 (这也是该对象最终应安放的位置)。
3. 然后分别对这两个子序列重复施行上述方法，直到所有的对象都排在相应位置上为止。

例 1：对下列序列采用快速排序

{49 38 65 97 76 13 27 49}

第 1 趟快速排序:

28 38 13 49 76 97 65 49

第 2 趟快速排序:

13 27 38 49 49 65 76 97

第 3 趟快速排序:

13 27 38 49 49 65 76 97

例 2：一组记录的关键码为（46，79，56，38，40，84），则利用快速排序的方法，以第一个记录为基准得到的一次划分结果为（40，38，46，56，79，84）

例 3：在对一组记录（54,38,96,23,15,72,60,45,83) 进行直接插入排序时，当把第 7 个记录 60 插入到有序表时，为寻找插入位置需比较 3 次。

提示：当要插入 60 时，前 6 个元素已经有序，即为：15，23，38，54，72，96，需从后向前比较到 54 为止，故要比较 3 次。

9.3 选择排序

9.3.1 简单选择排序

排序过程:

1. 首先通过 n-1 次关键字比较，从 n 个记录中找出关键字最小的记录，将它与第一个记录交换；

2. 再通过 n-2 次比较，从剩余的 n-1 个记录中找出关键字次小的记录，将它与第二个记录交换；

3. 重复上述操作，共进行 n-1 趟排序后，排序结束。

```
void  SelectSort (SqList &L) {
//对顺序表L作简单选择排序
for(i=1;i<L.length; ++i)  {    //选择第i小的记录，并交换到位
Min=SelectMinKey(L, i);
//在L.r[i..L.length]中选择key最小的记录
if(i!=Min)  L.r[i] ⟷ L.r[Min]; //与第i个记录交换
}
}
```

栗子：

原始数据： 8 6 9 3 2 7

第一趟排序： 2 6 9 3 8 7

第二趟排序： 2 3 9 6 8 7

第三趟排序： 2 3 6 9 8 7

第四趟排序： 2 3 6 7 8 9

第五趟排序： 2 3 6 7 8 9

9.3.2 堆排序

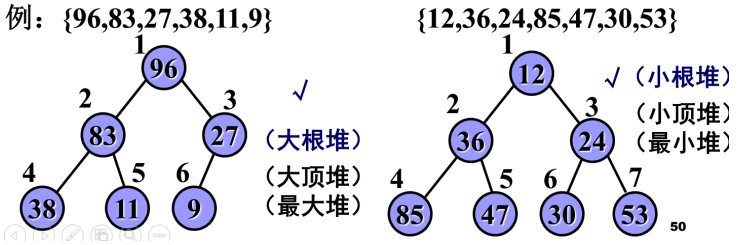


图 24: 大根堆和小根堆

基本思想：

1. 将无序序列建成一个堆，得到关键字最小（或最大）的记录；

2. 输出堆顶的最小（大）值后，使剩余的 n-1 个元素重又建成一个堆，则可得到 n 个元素的次小值；

3. 重复执行，得到一个有序序列，这个过程叫堆排序。

（初始建堆）的解决方法：从无序序列的第 $\lfloor n/2 \rfloor$ 个元素（即此无序序列对应的完全二叉树的最后一个非终端结点）起，至第一个元素止，进行反复筛选。

筛选方法：输出堆顶元素之后，以堆中最后一个元素替代之；然后将根结点值与左、右子树的根结点值进行比较，并与其中小者进行交换；重复上述操作，直至叶子结点，将得到新的堆，称这个从堆顶至叶子的调整过程为“筛选”。

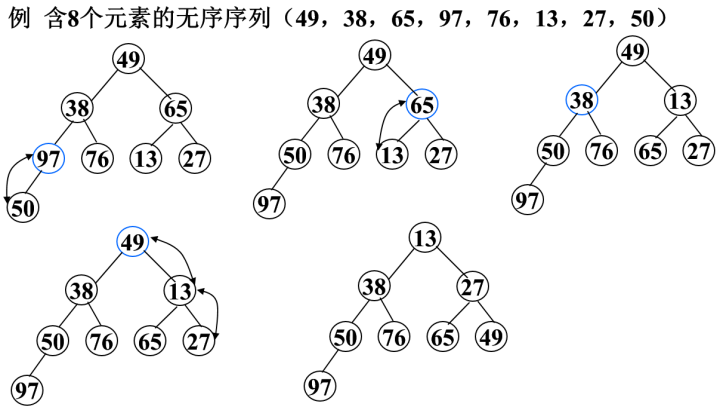
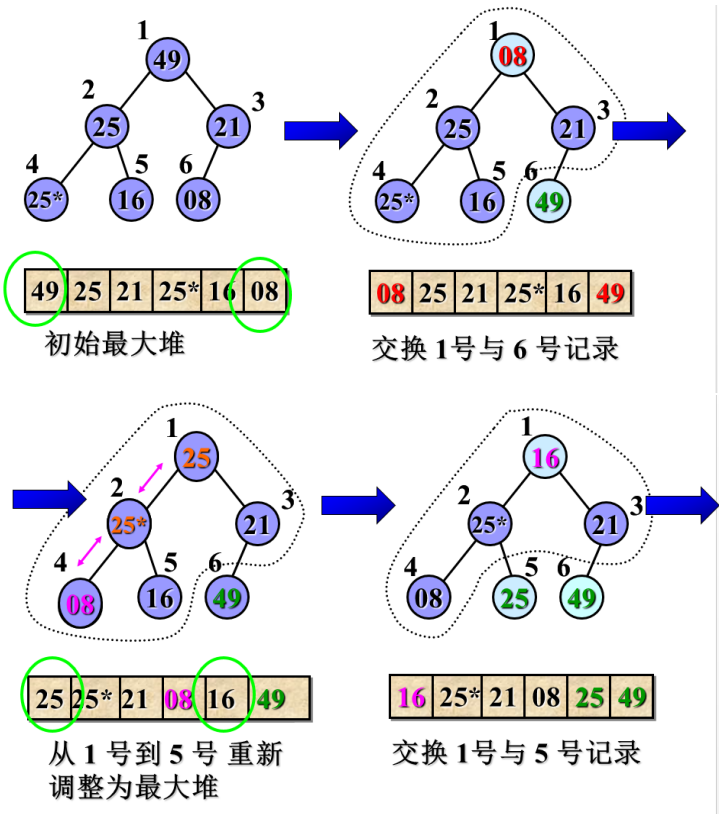
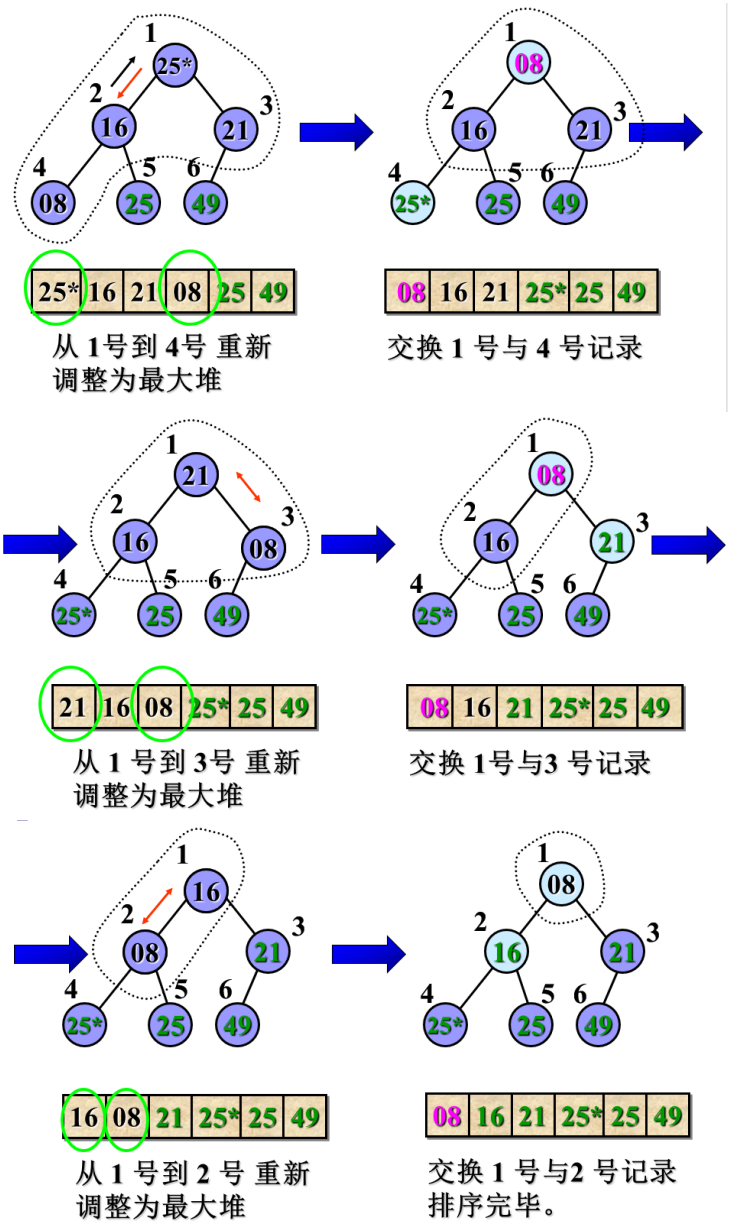


图 25: 初始建堆

例：关键字序列 T=(21，25，49，25*，16，08)，请给出采用堆排序法对该序列进行升序排序时的每一趟的结果。





9.4 总结

稳定的排序有：直接插入排序，冒泡排序

时间复杂度为 n^2 的有：直接插入排序，冒泡排序，简单选择

时间复杂度为 $nlog(n)$ 的有：快速排序和堆排序