复习重点

第一章　绪论

1.1 数据、 数据元素、 数据项、 数据结构等基本概念

1. 数据（data）：客观事物的符号表示，在计算机科学中指所有能输入计算机中并被计算机处理的符号总称。整数、浮点数、字符串、声音、图像。
2. 数据元素（data element）：数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。
3. 一个数据元素可能由若干个数据项（data item）组成。数据元素是一个数据整体中相对独立的单位。但它还可以分割成若干个具有不同属性的项（字段）。故不是组成数据的最小单位。数据项是构成数据的最小单位。
4. 数据对象（data object）：性质相同的数据元素的集合，是数据的一个子集。
5. 数据结构（data structure）：数据元素以及数据元素之间存在的关系。
6. 数据结构主要描述：数据元素之间的逻辑关系、数据在计算机系统中的存储方式和数据的运算，即数据的逻辑结构、存储结构和数据的操作集合

1.2 数据结构的逻辑结构、 存储结构的含义及其相互关系

1. 数据的逻辑结构：用形式化方式描述数据元素间的关系。数据的逻辑结构独立于计算机，是数据本身所固有的。用于算法的设计。
2. 有两大类逻辑结构：线性结构（线性表、栈、队列、数组和串），非线性结构（树和图）。
3. 数据的物理结构（也称存储结构）：数据在计算机中的具体表示。包括数据元素的表示和关系的表示。存储结构是逻辑结构在计算机存贮器中的映像，必须依赖于计算机。用于算法的实现。
4. 数据的存储方式可分为如下两类：顺序存储、链接存储。

1.3 算法

1. 算法的定义：算法是对特定问题求解步骤的一种描述， 是指令的有限序列。
2. 算法的特性：

有穷性——算法必须在执行有穷步之后结束，而且每一步都可在有穷时间内完成

确定性——每条指令无二义性。并且，相同的输入只能得到相同的输出；

可行性——算法中描述的每一操作，都可以通过已实现的基本运算来实现。

输入——算法有零至多个输入。

输出——算法有一个至多个输出

3. 算法效率的度量：时间复杂度和空间复杂度及计算。

第二章　线性表

2.1 线性表的逻辑结构特征

存在唯一的一个被称作第一个的数据元素；存在唯一的一个被称作最后一个的数据元素；除第一个元素之外，集合中的每个数据元素均只有一个前驱；除最后一个元素之外，集合中的每个数据元素均只有一个后继。

2.2 线性表的顺序存储结构

1. 用一组连续的存储单元依次存储线性表的数据元素。在线性表的顺序存储表示中，只要确定了线性表的起始位置，线性表中任一数据元素都可随机存取。线性表的顺序存储结构是一种随机存取的存储结构。

   $LOC(a_{i+1}) = LOC(a_i) + l$

   $LOC(a_{i+1}) = LOC(a_1) + i*l$

   $LOC(a_i)$表示元素 a_i 的存储位置； $LOC(a_1)$ 表示第一个数据元素的存储位置，通常称为线性表的起始位置或基地址每个数据元素占用 l 个存储单元。

2. 线性顺序表上的插入是指在第 i(1≤i≤n+1)个位置插入一个新的数据元素，需将第 i 至第 n 共(n-i+1)个元素后移

   注意：

   ➢ 顺序表中数据区域有 listSize 个存储单元,所以在向顺序表中做插入时先检查表空间是否满了，在表满的情况下不能再做插入，否则产生溢出错误。

   ➢ 要检验插入位置的有效性，这里 i 的有效范围是：

   1 <= i <= n + 1，其中 n 为原表长。

   ➢ 注意数据的移动方向

   算法时间复杂度：

   移动元素个数：n-i+1

   平均移动元素个数：n/2

   T（n）=O（n）；

3. 线性顺序表上的删除是指第 i (1≤i≤n)个数据元素删除掉，需将第 i+1 至第 n 共（n-i)个元素前移

   注意：

   ➢ 删除第 i 个元素，i 的取值为 1 <= i <= n ,否则第 i 个元素不存在，因此，要检查删除位置的有效性。

   ➢ 当表空时不能做删除。

   ➢ 删除 a_i 之后，该数据已不存在，如果需要，先取出 a_i，再做删除。

   算法时间复杂度：

   移动元素个数：n-i

   平均移动元素个数：（n-1）/2

   T（n）=O（n）；

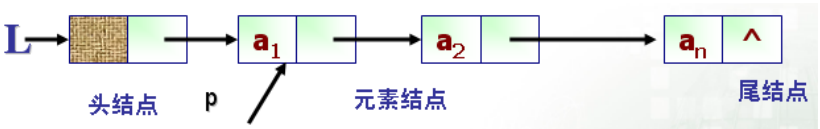
4. 线性表的顺序存储。

优点：逻辑相邻，物理相邻可以实现数据元素的随机存取；

缺点：在作插入或是删除操作时，需要移动大量数据元素

2.3 线性表的链式存储结构

1. 线性表链式存储结构的特点：用一组任意的存储单元存储线性表的数据元素。在线性表的链式存储中，在进行插入或是删除操作时，不需要进行数据元素的移动，但不能实现数据元素的随机存取。
2. 线性链表的表示：数据元素、数据元素之间的关系；
   数据域存储数据元素，指针域存储数据元素之间的关系：直接后继的存储位置，
   线性链表：每个节点只包含一个指针域



3. 假定指针 p 指向线性链表中的第 i 个数据元素，则 p->next 为指向线性链表中第 i+1 个数据元素的指针。即 p->data 为 ai， p->next ->data 为 ai+1。

   (*p)表示 p 所指向的结点

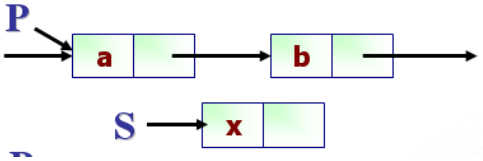
   (*p).data⇔p->data 表示 p 指向结点的数据域

   (*p).next⇔p->next 表示 p 指向结点的指针域

4. 在单链表中查找第 i 个元素

```
Status getElem_L(LinkList L, int i, ElemType &e)
{ //获取线性链表中的第 i 个数据元素
   p = L->next;
   j = 1;
   while (p && j < i)
   {
      p = p -> next;
      ++ j;
   }
   if ( !p || j > i) return ERROR;
   return p -> data;
} // GetElem_L
```

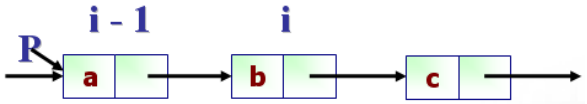
5. 在单链表中插入数据元素




```
S->next = P->next;
P->next = S;
```

```
Status listInsert_L(LinkList &L, int i, ElemType e){
  p = L;j = 0;
  while (p && j < i - 1){
     p = p -> next; ++ j;
  }
  if ( !p ‖ j > i - 1) return ERROR;
  s = (LinkList) malloc ( sizeof (LNode));
  s -> data = e;
  s -> next = p -> next;
  p -> next = s;
  return OK;
}
```

6. 在单链表中删除数据元素



P -> next = P -> next -> next; 或

```
q=p->next;
p->next=q->next;
free(q);
```

```
Status listDelete_L(LinkList &L, int i){
    p = L; j = 0;
    while (p -> next && j < i - 1){
       p = p -> next; ++j;
    }
    if ( !(p -> next) ‖ j > i - 1)return ERROR; // 删除位置不合理
    q = p -> next ;
    p -> next = q -> next;
    free(q); // 删除并释放结点
    return OK;
} // ListDelete_L
```

7. 循环链表：表中最后一个结点的指针域指向头结点，整个链表形成一个环。

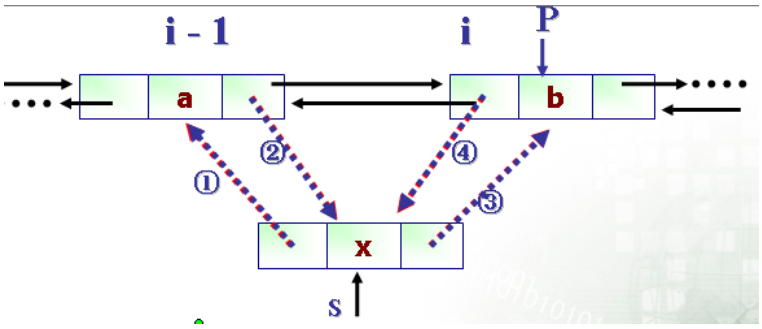
循环链表的操作和单链表基本一致，差别仅在于，判别链表中最后一个结点的条件不再是"后继是否为空"，而是"后继是否为头结点"。

(1) 单链表 p 或 p->next==NULL

(2) 循环链表 p->next==L

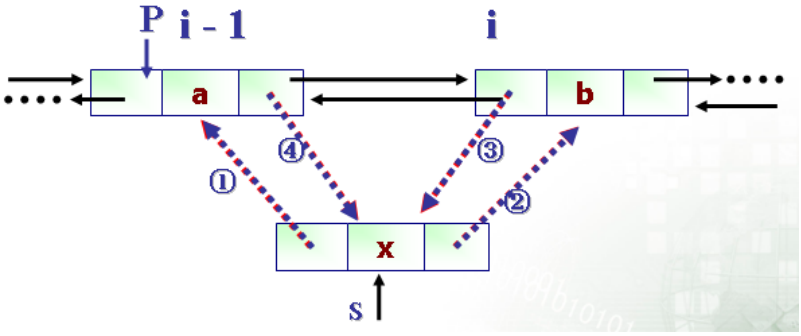
8. 双向链表有两个指针域，一个指向直接前驱，一个指向直接后继。

1）向双向链表中插入一个结点：



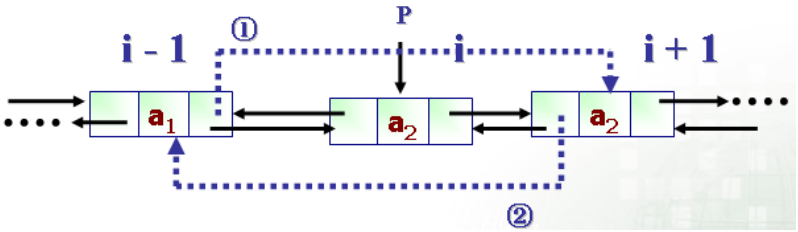

```
s->prior = p->prior;
p->prior->next = s;
s->next = p;
p->prior = s;
```

2）向双向链表中插入一个结点：：




```
s->prior = p;
s->next = p->next;
p->next->prior = s;
p->next = s;
```

3）从双向链表中删除一个结点



① p->prior->next = p->next;

② p->next->prior = p->prior;

第三章　栈和队列

3.1 栈和队列的逻辑结构特征

1. 栈（stack）和队列（queue）是两种重要的线性结构，特殊性在于其基本操作是线性表操作的子集，是操作受限的线性表（操作限定在两个端点进行），为具有限定性的数据结构。栈按“后进先出”的规则进行操作，队列按“先进先出”的规则进行操作。

2. 栈是限定在表尾进行插入和删除操作的线性表。允许插入,删除的一端称为栈顶(top),另一端称为栈底(bottom)。
3. 栈的基本运算主要有两个：Push(S, e)，进栈，插入（压入）元素 e 为新的栈顶元素，Pop(S)，出栈，删除（弹出）S 的栈顶元素。如：若元素入栈的顺序为 1234，为了得到 1342 出栈顺序，操作序列为：Push(S, 1)，Pop(S)，Push(S, 2)，Push(S, 3)，Pop(S)，Push(S, 4)，Pop(S)，Pop(S)。

3.2 栈的顺序存储结构

1. 顺序栈：利用一组地址连续的存储单元一次存放从栈底到栈顶的数据元素，用指针 top 指示栈顶元素在顺序栈中的位置。

	top 指向栈顶元素的下一个位置	top 指向栈顶元素
初始化：	S.top = S.base	S.top = S.base-1
判栈空：	S.base == S.top	S.base == S.top-1
入栈：	*s->top++= e（先压后加）	*++s->top= e（先加后压）
栈满：	S.top - S.base >= S.stacksize	S.top - S.base >= S.stacksize-1
出栈：	e = *--S.top（先减后弹）	e = *S.top--（先弹后减）

- 入栈时，首先判栈是否满了，栈满的条件为：S.top - S.base >= S.stacksize，栈满时，不能入栈; 否则出现空间溢出，引起错误，这种现象称为上溢。
- 出栈和读栈顶元素操作，先判栈是否为空，为空时不能操作，否则产生错误。通常栈空时常作为一种控制转移的条件。

2. 用数组的索引值表示栈底和栈顶

	top 指向栈顶元素的下一个位置	top 指向栈顶元素
初始化：	top=0	top=-1
判栈空：	top==0	top==-1
入栈：	v[top++]=x（先压后加）	v[++top]=x（先加后压）
栈满：	top-base>=stacksize;	top-base>=stacksize-1;
出栈：	y=v[--top])（先减后弹）	y=v[top--])（先弹后减）

3. 两栈共享空间:

top[0]表示第一个栈的栈顶; top[1]表示第二个栈的栈顶

栈空:top[0]=-1; top[1]=n

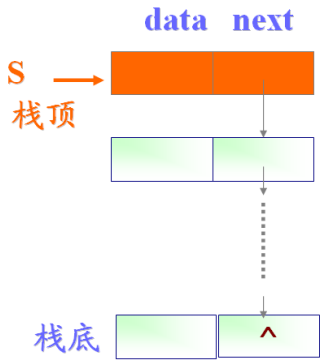
入栈: a[++top[0]]=e; a[--top[1]]=e

栈满:top[0]+1=top[1]

出栈: e= a[top[0] --] ; e=a[top[1]++]

4. 关于顺序栈的说明:入栈时，首先判栈是否满了，栈满时，不能入栈; 否则出现空间溢出，引起错误，这种现象称为上溢。 出栈和读栈顶元素操作，先判栈是否为空，为空时不能操作，否则产生错误。通常栈空时常作为一种控制转移的条件。

3.3 栈的顺序链式存储



入栈：

```
p = new LNode ; // 建新的结点
if(!p) exit(1) ; // 存储分配失败
p -> data = e;
p->next=S->top ;// 链接到原来的栈顶
S->top = p;    // 移动栈顶指针
```

出栈：

```
if ( !S->top )return NULL;
  else
  {  e=S->top->data; // 返回栈顶元素
     q = S->top;
     S->top=S->top->next; //修改栈顶指针
     free(q); // 释放被删除的结点空间
     return e;
   }
```

3.4 栈的应用举例

1.数制转换

```
#define NUM  10
void conversion(int N, int r){
    int  s[NUM],top;      /*定义一个顺序栈*/
    int   x;
    top = -1;          /*初始化栈*/
    while ( N ){
        s[++top] = N % r; /*余数入栈 */
        N = N / r ;       /* 商作为被除数继续 */
   }
   while (top != -1){
   x = s[top --];
```

```
        printf(“%d”,x);
        }
    }
```

2. 括号匹配的检验：

3. 表达式求值 ：熟悉前缀、中缀和后缀表达式，表达式求值时栈的状态变化。

4. 栈与递归的实现：熟悉使用递归解决

3.5 队列的逻辑结构特征

队列：只允许在一端进行插入，而在另一端删除元素。允许插入的一端为队尾(rear)，允许删除的一端为队头(front) 。

3.6 队列的顺序存储结构

1. 循环队列的顺序存储结构：队列存放数组被当作首尾相接的表处理。队头、队尾指针加 1 时用语言的取模(余数)运算实现。

队列初始化：front = rear = 0;

队空条件：front == rear;

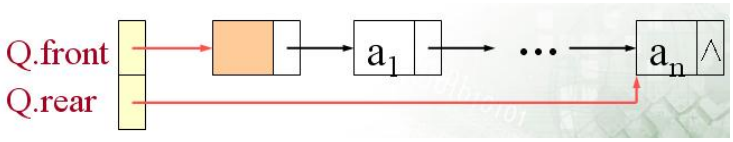
队满条件：(rear+1) % MAXQSIZE == front

队头指针进 1: front = (front+1) % MAXQSIZE;

队尾指针进 1: rear = (rear+1) %MAXQSIZE;

队中元素个数：（rear-front+ MAXQSIZE）% MAXQSIZE

2. 链式队列：



进队：

```
p = (QueuePtr) malloc (sizeof (QNode));
if (!p)return 0;     //存储分配失败
p->data = e;     p->next = NULL;
Q->rear->next = p;       Q.rear = p;
```

出队：

```
if (Q->front == Q->rear)       return NULL;
p = Q->front->next;     e = p->data;
Q->front->next = p->next;
if (Q->rear == p)    Q->rear = Q->front;
free (p);          return e;
```

第四章　串、数组和广义表

4.1 串相关术语

串即字符串，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。

串长：串中字符个数（n≥0）. n=0 时称为空串 ∅。

空白串：由一个或多个空格符组成的串。

子串：串 s 中任意个连续的字符序列叫 s 的子串; s 叫主串。

子串位置：子串的第一个字符的序号。

字符位置：字符在串中的序号

串相等：串长度相等，且对应位置上字符相等。

串的逻辑结构和线性表极为相似，区别仅在于串的数据对象约束为字符集；串的基本操作和线性表有很大差别。在线性表的基本操作中，大多以“单个元素”作为操作对象；在串的基本操作中，通常以“串的整体”作为操作对象。

4.2 串的基本操作

熟悉以下操作的意义：

StrAssign (&T, chars)

StrCopy (&T, S)

DestroyString (&S)

StrEmpty (S)

StrCompare (S, T)

StrLength (S)

Concat (&T, S1, S2)

SubString (&Sub, S, pos, len)

Index (S, T, pos)

Replace (&S, T, V)

StrInsert (&S, pos, T)

StrDelete (&S, pos, len)

ClearString (&S)

4.3 数组

1. 二维数组的顺序存储结构及地址计算方式。

$$A_{mn}=\begin{bmatrix} a_{c1,c2} & \cdots & a_{c1,d2} \\ \cdots & a_{i,j} & \cdots \\ a_{d1,c2} & \cdots & a_{d1,d2} \end{bmatrix}$$

设一般的二维数组是 A[c1..d1, c2..d2]，这里 c1,c2 不一定是 0。L：单个元素长度

则行优先存储时的地址公式为：

LOC(aij)=LOC(c1，c2)+[(i-c1)*(d2-c2+1)+(j-c2)]*L

二维数组列优先存储的通式为：

LOC(aij)=LOC(ac1，c2)+[(j-c2)*(d1-c1+1)+(i-c1)]*L

2. 对称矩阵的压缩存储：在对称矩阵中，只需存储对称矩阵的下半部分。所需空间数为：n×（n+1）/2。

设一般的二维数组是 A[c1..d1, c2..d2]，这里 c1,c2 不一定是 0，对应一维存储空间 SA 的起始值是 C3。

A=

3	6	4	7	8
6	2	8	4	2
4	8	1	6	9
7	4	6	0	5
8	2	9	5	7

SA	3	6	2	4	8	1	7	4	6	0	8	2	9	5	7

则行优先存储时的地址公式为：

$$k=\begin{cases}\dfrac{(i-c_1)\times(i-c_1-1)}{2}+j-c_2+c_3 & i\geq j\\ \dfrac{(j-c_1)\times(j-c_1-1)}{2}+i-c_2+c_3 & i<j\end{cases}$$

3. 三角矩阵：若 n 阶方阵中下(上)三角(不包括对角线)中的元均为常量 c 或 0，则称为上(下)三角矩阵；下三角矩阵：主队角线以上均为同一个常数；上三角矩阵，主队角线以下均为同一个常数。与对称矩阵类似，不同之处在于存完下三角中的元素之后，紧接着存储对角线上方的常量，因为是同一个常数，所以存一个即可，这样一共存储了 n*(n+1)/2 + 1 个元素，设存入数组：SA[n*(n+1)/2 + 1]中，这种的存储方式可节约 n*(n-1)/2 个存储单元。

4. 理解下、上三角矩阵：SAk 与 ai, j 的对应关系。

5. 稀疏矩阵：将每个非零元素用一个三元组（i, j, aij）来表示，将三元组按行优先的顺序，同一行中列号从小到大的规律排列成一个线性表，称为三元组表，每个稀疏矩阵可用一个三元组表来表示。

	i	j	value
0	6	6	8
1	1	2	12
2	1	3	9
3	3	1	-3
4	3	5	14
5	4	3	24
6	5	2	18
7	6	1	15
8	6	4	-7

0	12	9	0	0	0
0	0	0	0	0	0
-3	0	0	0	14	0
0	0	24	0	0	0
0	18	0	0	0	0
15	0	0	-7	0	0

注意：为更可靠描述，通常再加一行“总体”信息：即总行数、总列数、非零元素总个数

31

4.4 广义表

1. 广义表是递归定义的线性结构，是线性表的推广，也称为列表(lists)

记为： LS = (1, 2,..., n) 。

2. 广义表与线性表的区别和联系：广义表中元素既可以是原子类型，也可以是列表；

当每个元素都为原子且类型相同时，就是线性表。

3. 广义表 LS =(α1,α2,…,αn)的的性质：

1) 广义表中的数据元素有相对次序；

2)广义表的长度定义为最外层包含元素个数；

3)广义表的深度定义为所含括弧的最大重数；

注意:“原子”的深度为 0;“空表”的深度为 1

4)广义表是一种多层次的数据结构。广义表的元素可以是单元素，也可以是子表，而子表的元素还可以是子表，…。

5)广义表可以是递归的表。广义表的定义并没有限制元素的递归，即广义表也可以是其自身的子表。

6)广义表可以为其他表所共享。

7) 任何一个非空广义表 LS = (1, 2, …, n)

均可分解为: 表头 Head(LS) = 1 和 表尾 Tail(LS) =(2, …, n) 两部分.

任何一个非空表，表头可能是原子，也可能是列表；但表尾一定是列表

4. 广义表的基本运算：广义表有两个重要的基本操作,即取头操作(Head)和取尾操作(Tail)。要熟悉这个两个操作，正确给出一个广义表的这两个操作的结果。

第五章 树及二叉树

5.1 树结构及基本概念

1. 树具有下面两个特点：

1) 树的根结点没有前驱结点，除根结点之外的所有结点有且只有一个前驱结点。

2) 树中所有结点可以有零个或多个后继结点。

2.基本术语：

结点(node)： 表示树中的元素，包括数据项及若干指向其子树的分支

结点的度(degree)：结点拥有的子树数称为~

叶子(leaf)：度为 0 的结点

孩子(child)： 结点子树的根称为该结点的孩子

双亲(parents)： 孩子结点的上层结点

兄弟(sibling)： 同一双亲的孩子

树的度： 一棵树中最大的结点度数

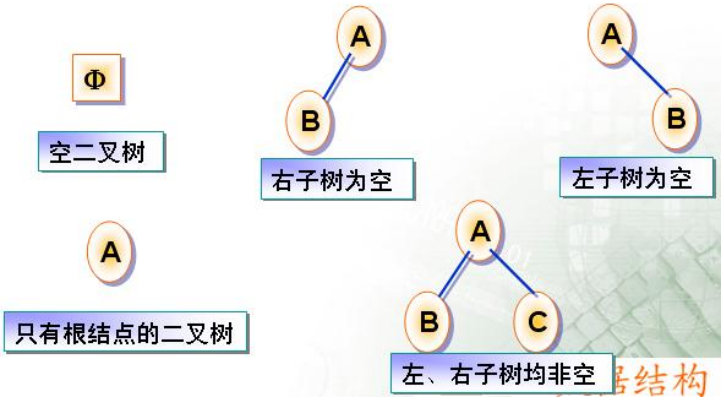
结点的层次(level)： 从根结点算起，根为第一层，它的孩子为第二层……

深度(depth)： 树中结点的最大层次数

森林(forest)： m(m≥0)棵互不相交的树的集合

5.2 二叉树结构

1. 定义：二叉树是n(n 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
2. 特点：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
3. 基本形态：



4. 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：对任意二叉树 BT ，若叶结点数为 n_0 ，度为 2 的结点数为 n_2，则：$n_0 = n_2 + 1$

性质 4：具有n个结点的完全二叉树的深度为$\lfloor \log_2 n \rfloor + 1$

性质 5：如果对一棵有 n 个结点的完全二叉树的结点按层序编号，则对任一结点 i(1≤i≤n)，有： (1) 如果 i=1，则结点 i 是二叉树的根，无双亲；如果 i>1，则其双亲是结点⌊i/2⌋

(2) 如果 2i>n，则结点 i 无左孩子；如果 2i≤n，则其左孩子是结点 2i

(3) 如果 2i+1>n，则结点 i 无右孩子；如果 2i+1≤n，则其右孩子是结点 2i+1

5. 几种特殊形式的二叉树：

满二叉树：一棵深度为k且有2^k-1个结点的二叉树称为~

特点：每一层上的结点数都是最大结点数

完全二叉树：深度为 k，有 n 个结点的二叉树当且仅当其每一个结点都与深度为 k 的满二叉树中编号从 1 至 n 的结点一一对应时，称为~

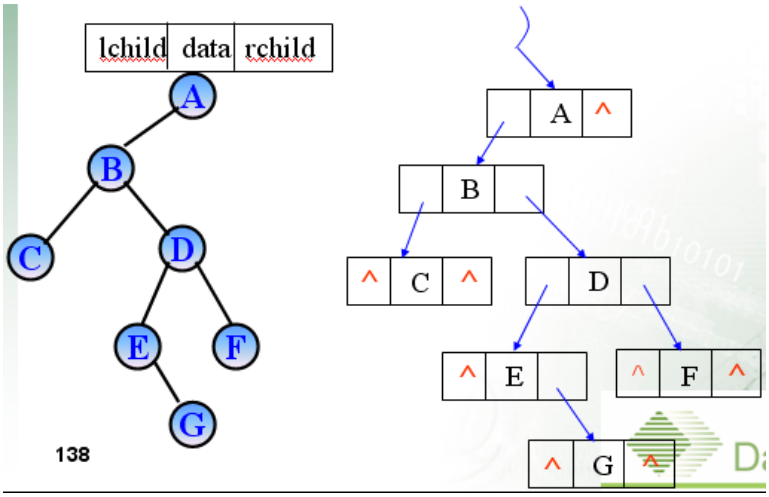
特点：叶子结点只可能在层次最大的两层上出现；对任一结点，若其右分支下子孙的最大层次为 l，则其左分支下子孙的最大层次必为 l 或 l+1

5.3 二叉树存储

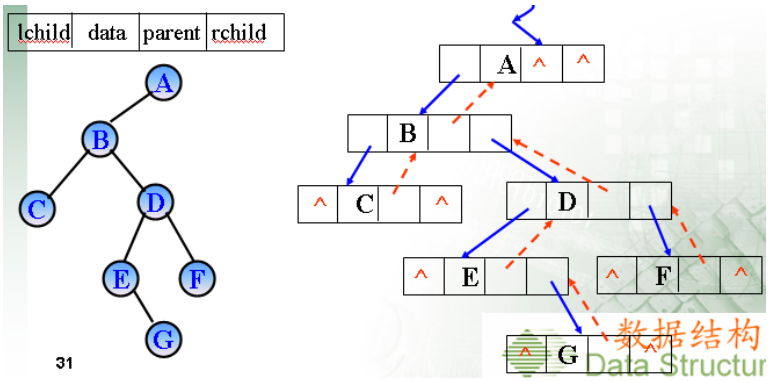
1. 二叉树的顺序存储结构：按满二叉树的结点层次编号，依次存放二叉树中的数据元素

特点：结点间关系蕴含在其存储位置中；浪费空间，适于存满二叉树和完全二叉树

2. 二叉树的链式存储结构（二叉链表）：在 n 个结点的二叉链表中，有 n+1 个空指针域



3. 二叉树的链式存储结构（三叉链表）



5.4 二叉树遍历

1. 二叉树的遍历：

先序遍历(DLR) ：先访问根结点,然后分别先序遍历左子树、右子树

中序遍历(LDR)：先中序遍历左子树，然后访问根结点，最后中序遍历右子树

后序遍历(LRD)：先后序遍历左、右子树，然后访问根结点

2. 遍历的递归算法：

```
void    preOrder(bt){ /*先序遍历二叉树 bt*/
 if (bt) {  /*递归调用的结束条件为 bt 为空*/
     visit(bt->data);  /*访问结点的数据域*/
     preorder(bt->lchild); /*先序递归遍历 bt 的左子树*/
     preorder(bt->rchild); /*先序递归遍历 bt 的右子树*/
  }
}
void    inOrder(bt){ /*中序遍历二叉树 bt*/
     if (bt) {  /*递归调用的结束条件为 bt 为空*/
          inOrder(bt->lchild);  /*中序递归遍历 bt 的左子树*/
```

```
            visit(bt->data);   /*访问结点的数据域*/
            inOrder(bt->rchild);  /*中序递归遍历 bt 的右子树*/
        }
    }
    void   postOrder(bt) { /*后序遍历二叉树 bt*/
        if (bt) { /*递归调用的结束条件为 bt 为空*/
            postOrder(bt->lchild); /*后序递归遍历 bt 的右子树*/
            postOrder(bt->rchild); /*后序递归遍历 bt 的右子树*/
            visit(bt->data);          /*访问结点的数据域*/
        }
    }
```

5.5 线索二叉树

1. 线索二叉树的定义

前驱与后继：在二叉树的先序、中序或后序遍历序列中两个相邻的结点互称为~

线索：指向前驱或后继结点的指针称为~

线索二叉树：加上线索的二叉链表表示的二叉树叫~

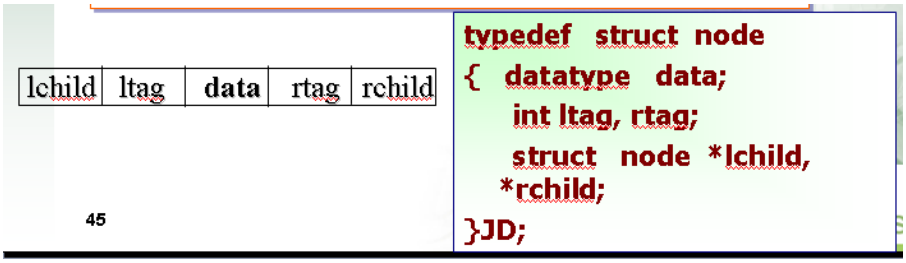
线索化：对二叉树按某种遍历次序使其变为线索二叉树的过程叫~

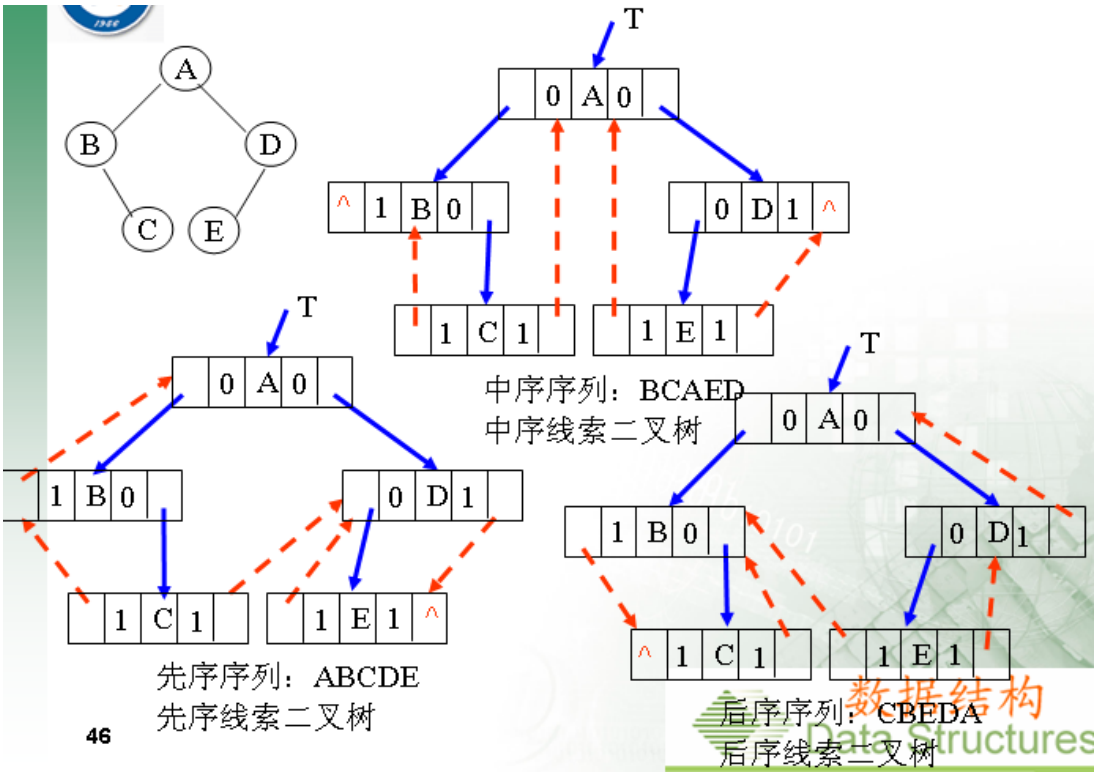
2. 线索二叉树的实现

在有 n 个结点的二叉链表中必定有 n+1 个空链域。在线索二叉树的结点中增加两个标志域

ltag :若 ltag =0, lchild 域指向左孩子；若 ltag =1, lchild 域指向其前驱

rtag :若 rtag =0, rchild 域指向右孩子；若 rtag =1, rchild 域指向其后继





3. 在中序线索二叉树中找结点后继的方法

若 rt=1，则 rc 域直接指向其后继

若 rt=0，则结点的后继应是其右子树的左链尾（lt=1)的结点

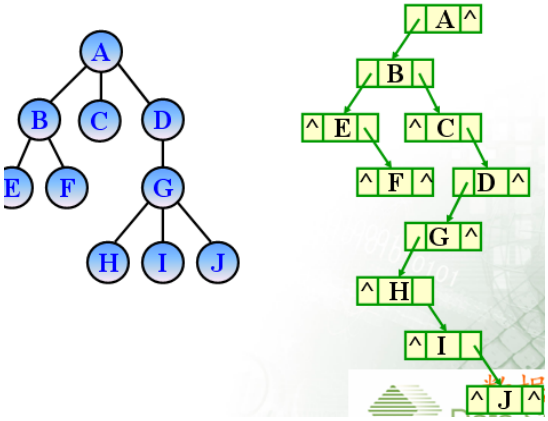
4. 在中序线索二叉树中找结点前驱的方法：

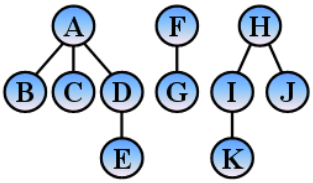
若 lt=1，则 lc 域直接指向其前驱

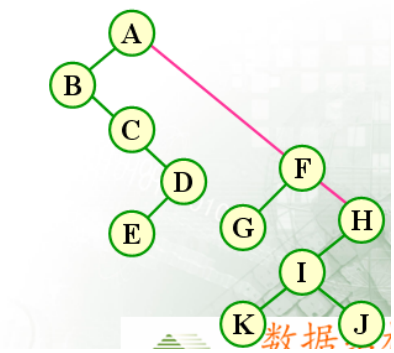
若 lt=0，则结点的前驱应是其左子树的右链尾（rt=1)的结点

5.6 树和森林

1. 树和森林与二叉树之间的转换方法：　孩子兄弟表示法







5.7 赫夫曼树

1. 赫夫曼树(Huffman)——带权路径长度最短的树
2. 赫夫曼算法

   1)根据给定的 n 个权值构成 n 棵二叉树的集合 F,其中每棵二叉树中只有一个带权值的结点；

   2）在 F 中选取两棵根结点的权值最小的树作为左右子树构造一棵新的二叉树，且置新的二叉树的根结点的权值为其左、右子树上根结点的权值之和；

   3）在 F 中删除这两棵树，同时将新得到的二叉树加入到 F 中；

   4）重复 2)和 3)，直到 F 中只含一棵树为止
3. Huffman 编码：数据通信用的二进制编码

   1）思想：根据字符出现频率编码，使电文总长最短

   2）编码：根据字符出现频率构造 Huffman 树，然后将树中结点引向其左孩子的分支标“0”，引向其右孩子的分支标“1”；每个字符的编码即为从根到每个叶子的路径上得到的 0、1 序列

   3）译码：从 Huffman 树根开始，从待译码电文中逐位取码。若编码是“0”，则向左走；若编码是“1”，则向右走，一旦到达叶子结点，则译出一个字符；再重新从根出发，直到电文结束

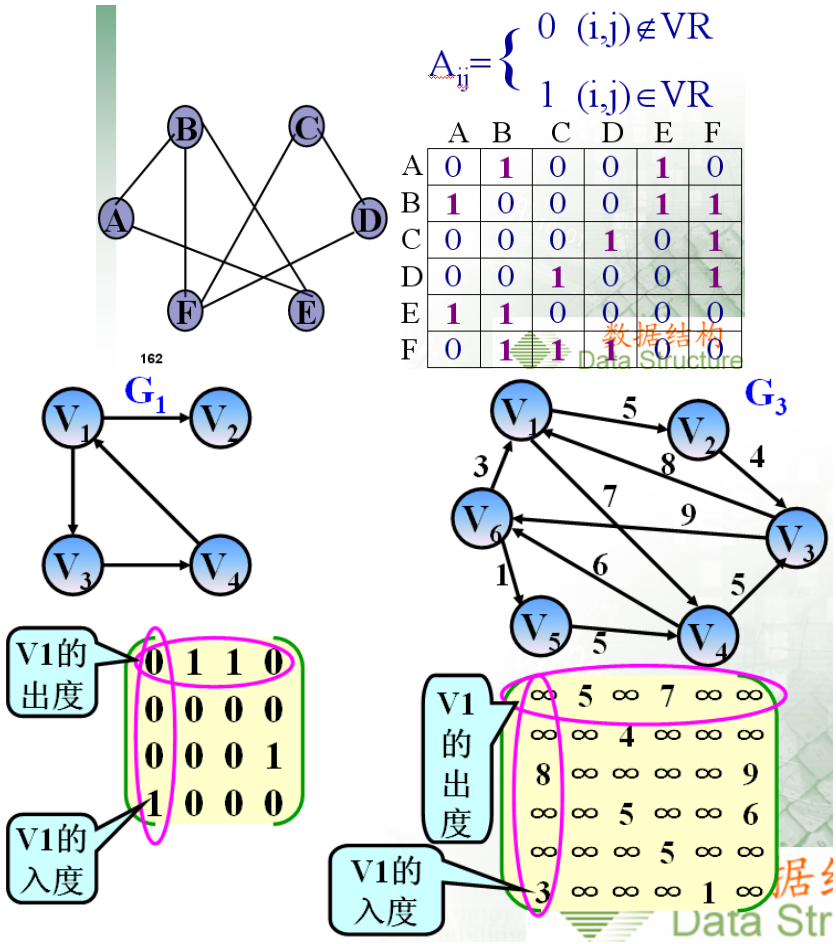
第六章　图

6.1 图的术语

1. 图的常用术语及含义:有向图、无向图、完全图、有向完全图、稀疏图、稠密图、网、邻接点、路径、简单路径、回路或环、简单回路、连通、连通图、强连通图、生成树。用 n 表示图中顶点数目,用 e 表示图中边或弧的数目:0 ≤ e ≤ ½*n(n-1)(无向图),0 ≤ e ≤ n(n-1)（有向图）

6.2 图的邻接矩阵存储表示

1. 图的数组(邻接矩阵)存储表示



2. 图的邻接矩阵存储方法具有以下特点：

（1）无向图的邻接矩阵一定是一个对称矩阵。因此，在具体存放邻接矩阵时只需存放上（或下）三角矩阵的元素即可。

（2）对于无向图，邻接矩阵的第 i 行（或第 i 列）非零元素的个数正好是第 i 个顶点的度 TD(vi)。

（3）对于有向图，邻接矩阵的第 i 行（或第 i 列）非零元素的个数正好是第 i 个顶点的出度 OD(vi)（或入度 ID(vi)）。

3. 图的邻接矩阵存储方法的优点：用邻接矩阵方法存储图，很容易确定图中任意两个顶点之间是否有边相连

4. 图的邻接矩阵存储方法的局限性：要确定图中有多少条边，则必须按行、按列对每个元素进行检测，所花费的时间代价很大。存储空间为 O(n2)，适用于稠密图。

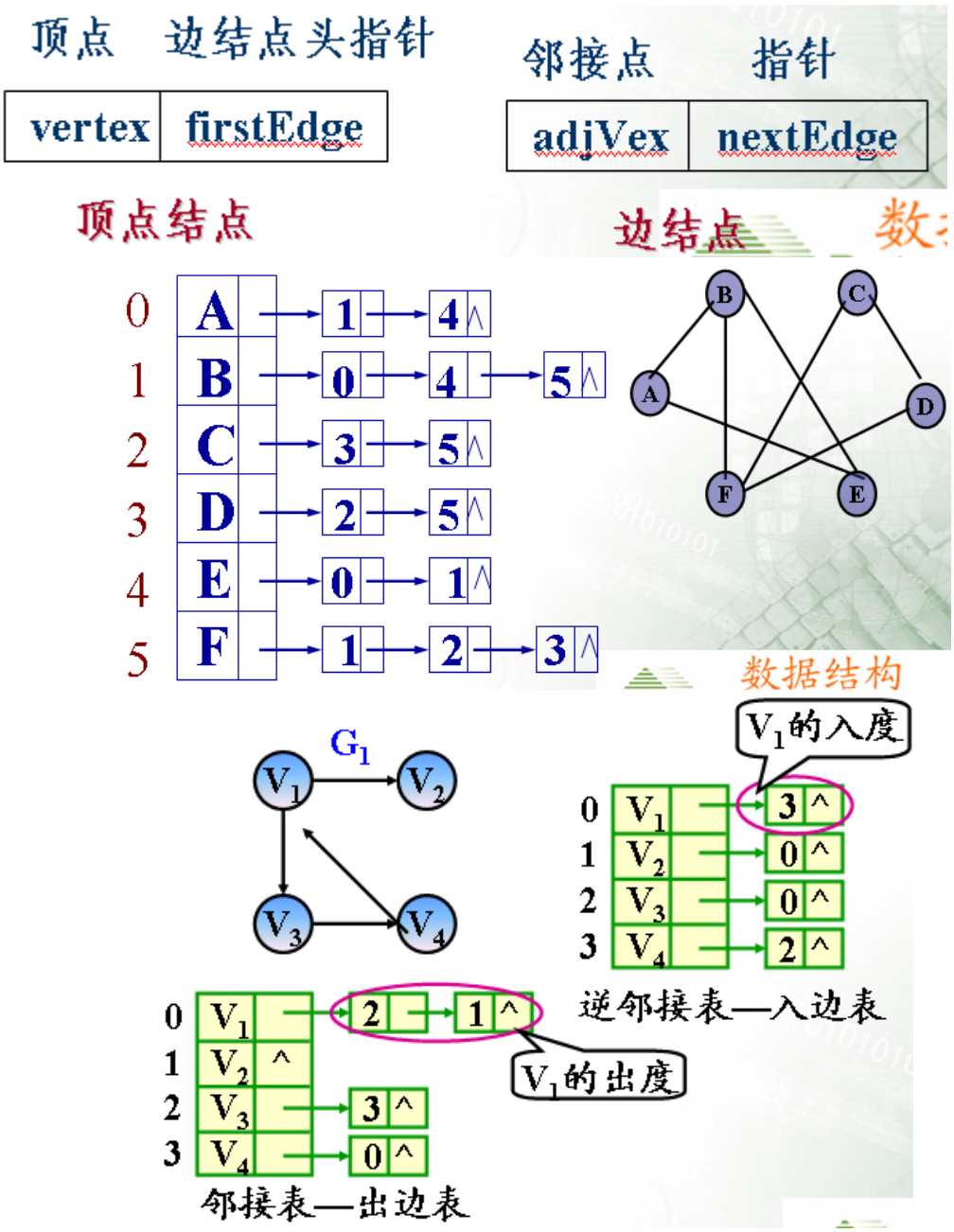
图的邻接表存储表示

6.3 图的邻接表存储表示

1. 邻接表(Adjacency List)是图的一种顺序存储与链式存储结合的存储方法。

对于图 G 中的每个顶点 vi，将所有邻接于 vi 的顶点 vj 链成一个单链表，这个单链表就称为顶点 vi 的邻接表，再将所有顶点的邻接表表头放到数组中，就构成了图的邻接表。

2. 邻接表表示中包括两种结点结构：



3. 邻接表表示法的优点：在邻接表上容易找到任一顶点的第一个邻接点和下一个邻接点

4. 邻接表表示法的局限性：要判定任意两个顶点（vi 和 vj）之间是否有边或弧相连，则需搜索第 i 个或第 j 个链表，因此，不及邻接矩阵方便

6.4 图的遍历

1. 图的深度优先遍历(depth-first search，DFS)：假设初始状态是图中所有顶点未曾被访问，则深度优先搜索可从图中某个顶点 v 出发，访问此顶点；然后依次从 v 的未被访问的邻接点出发深度优先遍历图；直至图中所有和 v 有路径相通的顶点都被访问到；若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。

2.图的广度优先遍历(breadth-first search，BFS)：假设从图中某顶点 v 出发，在访问了 v 之后依次访问 v 的各个未曾访问过的邻接点；然后分别从这些邻接点出发依次访问它们的邻接点，并使“先被访问的顶点的邻接点”先于“后被访问的顶点的邻接点”被访问；直至图中所有已被访问的顶点的邻接点都被访问到。若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。

6.5 最小生成树

1.生成树

1)一个连通图的生成树是由 n-1 条边且包含 G 的所有顶点的树组成。

2)可按深度或广度优先遍历来创建生成树。

3)由深度优先遍历得到的为深度优先生成树；

4)由广度优先遍历得到的为广度优先生成树。

5)对于非连通图，通过这样的遍历，将得到的是生成森林。

2.最小生成树：如果无向连通图是一个网，那么，它的所有生成树中必有一棵边的权值总和最小的生成树，我们称这棵生成树为最小生成树，简称为最小生成树

3.普里姆算法的基本思想： 取图中任意一个顶点 v 作为生成树的根，之后往生成树上添加新的顶点 w。在添加的顶点 w 和已经在生成树上的顶点 v 之间必定存在一条边，并且该边的权值在所有连通顶点 v 和 w 之间的边中取值最小。之后继续往生成树上添加顶点，直至生成树上含有 n-1 个顶点为止。

4.克鲁斯卡尔算法：先构造一个只含 n 个顶点的子图 SG，然后从权值最小的边开始，若它的添加不使 SG 中产生回路，则在 SG 上加上这条边，如此重复，直至加上 n-1 条边为止。

5.普里姆算法：时间复杂度：$O(n^2)$，适应范围：稠密图

克鲁斯卡尔算法：时间复杂度：O(eloge)，适应范围：稀疏图

6.6 拓扑排序

1.拓扑排序:

按照有向图给出的次序关系，将图中顶点排成一个线性序列，对于有向图中没有限定次序关系的顶点，则可以人为加上任意的次序关系。由此所得顶点的线性序列称之为拓扑有序序列

2.检查有向图中是否存在回路的方法之一，是对有向图进行拓扑排序。

3.AOV 网： 用顶点表示活动,边表示活动间的先后关系的有向图称为顶点活动网,简称 AOV 网

4.拓扑排序算法

1)从有向图中选取一个没有前驱的顶点，并输出之;

2)从有向图中删去此顶点以及所有以它为尾的弧;

重复上述两步，直至图空，或者图不空但找不到无前驱的顶点为止。

6.7 关键路径

1.若在带权的有向图中，以：顶点表示事件；有向边表示活动；边上的权值表示活动的开销（如该活动持续的时间）。则此带权的有向图称为 AOE (activity on edge)网。

2. 由于 AOE 网中的某些活动能够同时进行，故完成整个工程所必须花费的时间应该为：源点到终点的最大路径长度（这里的路径长度是指该路径上的各个活动所需时间之和）。

1）具有最大路径长度的路径称为**关键路径**。

2）关键路径上的活动称为关键活动。

3）关键路径长度是整个工程所需的最短工期。这就是说，要缩短整个工期，必须加快关键活动的进度。

4）利用 AOE 网进行工程管理时要需解决的主要问题是：确定关键路径，以找出哪些活动是影响工程进度的关键活动。

3. AOE 网具有以下两个性质：

①只有在某顶点所代表的事件发生后，从该顶点出发的各有向边所代表的活动才能开始。

②只有在进入一某顶点的各有向边所代表的活动都已经结束，该顶点所代表的事件才能发生。

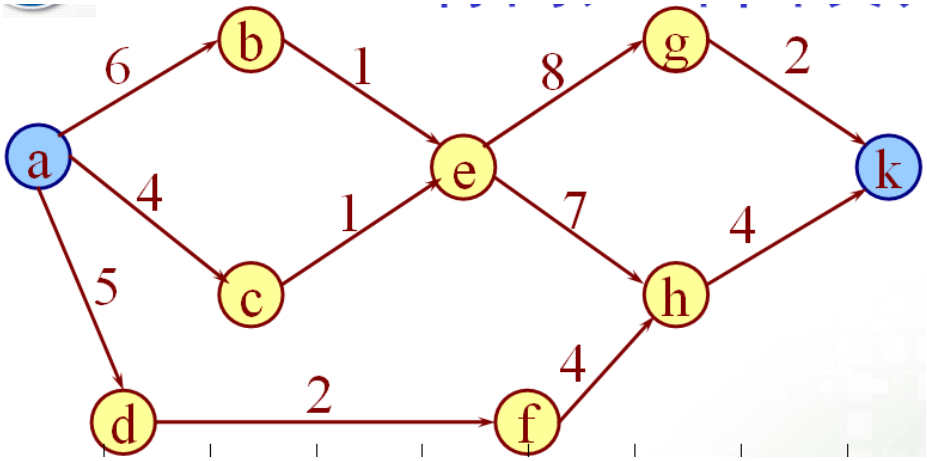
4. 求关键路径的算法讨论

ve(0)=0, ve(k)=maxj{ve(j)+ dut(<j, k>)}--拓扑有序

vl(n-1)=ve(n-1), vl(j)=mink{vl(k)- dut(<j, k>)}－逆拓扑有序

e(i)=ve(j)

l(i)=vl(k)- dut(<j, k>)



	a	b	c	d	e	f	g	h	k
ve	0	6	4	5	7	7	15	14	18
vl	0	6	6	8	7	10	16	14	18

	ab	ac	ad	be	ce	df	eg	eh	fh	gk	hk
权	6	4	5	1	1	2	8	7	4	2	4
e	0	0	0	6	4	5	7	7	7	15	14
l	0	2	3	6	6	8	8	7	10	16	14
	✓			✓				✓			✓

第七章　查找

7.1 顺序查找

1. 顺序查找又称线性查找，是最基本的查找方法之一。
2. 查找表结构：以顺序表或线性链表表示
3. 基本思想：从一端开始向另一端，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，若直至另一端，其关键字和给定值比较都不等，则表明表中没有所查记录，查找不成功。
4. 哨兵的作用：免去查找过程中每一步都要检测整个表是否查找完毕。
5. 平均查找长度(ASL)：

   查找成功时：$\frac{(n+1)}{2}$

   查找不成功时：n+1

   平均查找长度：$\frac{3(n+1)}{4}$

6. 顺序查找缺点：当 n 很大时，平均查找长度较大，效率低

顺序查找优点：对表中数据元素的存储没有要求。另外，对于线性链表，只能进行顺序查找。

7.2 折半查找(二分查找)

1. 查找表结构：以顺序表且有序表表示
2. 基本思想：查找区间逐步缩小(折半)
3. 能够画出其判定树：
4. 平均查找长度：$ASL_{bs} \approx \log_2(n+1)-1$

7.3 二叉排序树(二叉查找树)

1.定义(递归)：或者是一棵空树，或者是具有如下特性的二叉树：若它的左子树不空，则左子树上所有结点的值均小于根结点的值；若它的右子树不空，则右子树上所有结点的值均大于根结点的值；它的左、右子树也分别是二叉排序树

2.对二叉排序树进行中序遍历，便可得到一个按关键字有序的序列，

3.查找方法与算法

① 若查找树为空，查找失败。

② 查找树非空，将给定值 key 与查找树的根结点关键字比较。

③ 若相等，查找成功，结束查找过程，否则：

a. 当 key 小于根结点关键字，查找将在以左孩子为根的子树上继续进行，转①

b. 当 key 大于根结点关键字，查找将在以右孩子为根的子树上继续进行，转①

4.二叉排序树的插入

新插入的结点一定是一个新添加的叶子结点,并且是查找不成功时查找路径上访问的最后一个结点的左孩子或右孩子结点.

5.二叉排序树的删除

假设被删结点为*p，其双亲结点为*f，

1)*p 为叶子结点：删去*p，并修改*f 的孩子域。

2)*p 只有左子树 PL 或只有右子树 PR：令 PL 或 PR 直接成为*f 的子树

3)*p 的左子树 PL 和右子树 PR 均不为空

方法 1、令*p 的中序遍历的直接前驱替代*p，再从二叉排序树中删去它的直接前驱。

方法 2、与方法 3 对称，令*p 的中序遍历的直接后继替代*p，再从二叉排序树中删去它的直接后继。

6.二叉排序树的查找分析:与给定值比较的关键字个数不超过二叉排序树的深度

7.4 平衡二叉树(AVL 树)

1.定义(递归):

或者是一棵空树，或者是具有如下特性的二叉排序树：

左子树和右子树的深度之差的绝对值不超过 1；

它的左、右子树也分别是平衡二叉树。

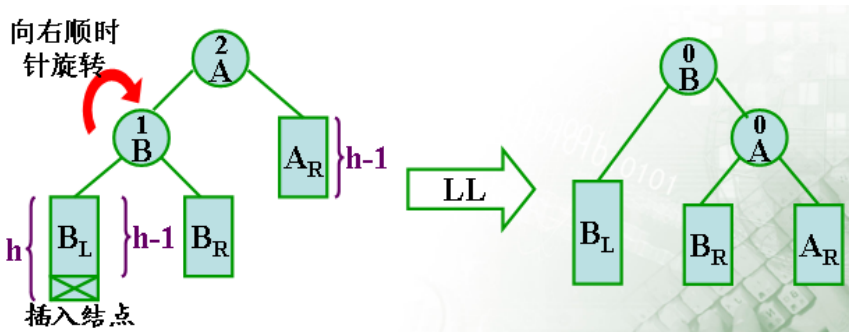
2.二叉树上结点的平衡因子 BF：该结点的左子树的深度减去它的右子树的深度。

3.AVL 树的深度和 logN 是同数量级的(其中的 N 为结点个数)。

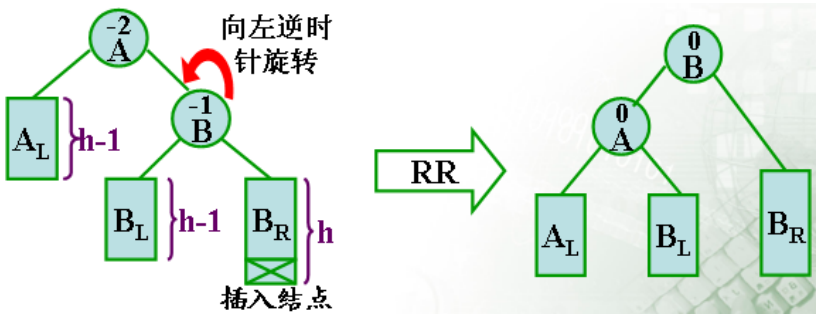
4.失去平衡后进行调整的规律

假设 a 是由于在二叉排序树上插入结点而失去平衡的最小子树根结点的指针(a 是离插入结点最近,且平衡因子绝对值超过 1 的祖先结点)

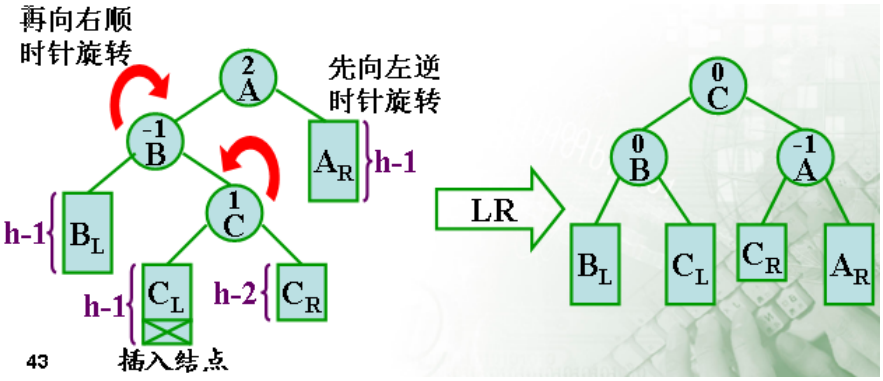
1.单向右旋平衡处理(LL 型)



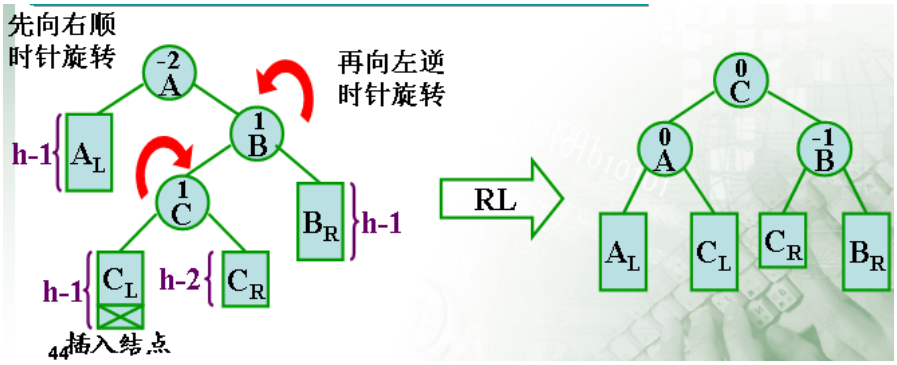
2. 单向左旋平衡处理(RR 型)



3. 双向旋转(先左后右)平衡处理(LR 型)



4. 双向旋转(先右后左)平衡处理(RL 型)



7.5 哈希表

1. 哈希(Hash)函数是一个映象,即：将关键字的集合映射到某个地址集合上,它的设置很灵活,只要这个地址集合的 大小不超出允许范围即可;
2. 由于哈希函数是一个压缩映象，因此，在一般情况下，很容易产生“冲突”现象，即： key1≠key2，而 f(key1) = f(key2)。

3. 在构造这种特殊的“查找表” 时，除了需要选择一个“好”(尽可能少产生冲突)的哈希函数之外;还需要找到一种“处理冲突” 的方法。
4. “处理冲突” 的实际含义是：为产生冲突的地址寻找下一个哈希地址。
5. 开放定址法:为产生冲突的地址 H(key) 求得一个地址序列：

   H0, H1, H2, …, Hs 1≤s≤m-1

   Hi = (H(key) + di) MOD m，其中： i=1, 2, …, s

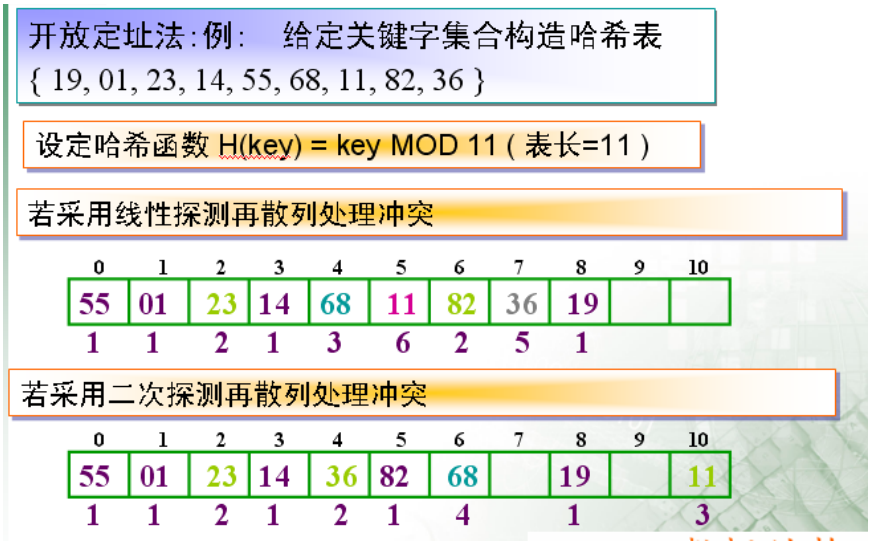
   H(key)为哈希函数;

   m 为哈希表长;

   di 为增量序列;
6. 开放定址法:对增量 di 的两种取法：

   线性探测再散列:di = c× i，最简单的情况 c=1

   平方探测再散列: di = 12, -12, 22, -22, …,



第八章 排序

8.1 基本概念

1. 排序算法的稳定性：如果在对象序列中有两个对象 r[i]和 r[j]，它们的关键字 k[i] == k[j]，且在排序之前，对象 r[i]排在 r[j]前面。如果在排序之后，对象 r[i]仍在对象 r[j]的前面， 则称这个排序方法是稳定的，否则称这个排序方法是不稳定的。

8.2 插入排序

1. 直接插入排序的基本思想以及在最好、最坏和平均情况下的时间性能分析

   (1) 当待排序序列为正序时，比较次数：n-1 次，交换次数：0 次

   (2) 当待排序序列为反序时，比较次数：n（n-1）/2 次，交换次数：n（n-1）/2 次

   (3) 时间复杂度：T（n）=O(n^2)

   (4) 空间复杂度：S（n）=O(1)

(5) 直接插入排序是一种稳定的排序

2.针对给定的输入实例，要能写出直接插入排序的排序过程

3.熟练写出直接插入排序的算法

8.3 冒泡排序

1.冒泡排序的基本思想以及在最好、最坏和平均情况下的时间性能分析

(1) 当待排序序列为正序时，比较次数：n-1 次，交换次数：0 次

(2) 当待排序序列为反序时，比较次数：n（n-1）/2 次，交换次数：n（n-1）/2 次

(3) 时间复杂度：T（n）=0(n^2)

(4) 空间复杂度：S（n）=0(1)

(5) 冒泡排序是一种稳定的排序

2.判断冒泡排序结束的条件

3.针对给定的输入实例，要能写出冒泡排序的排序过程

4.熟练写出冒泡排序的算法

8.4 选择排序

1.简单选择排序的基本思想和算法实现，以及时间性能分析

(1) 比较次数与待排序序列的原始状态无关。无论何时，比较次数：n（n-1）/2 次，

(2) 当待排序序列为正序时，交换次数： 0 次

(3) 最差情况（不是反序）：交换次数： n-1 次

(4) 时间复杂度：T（n）=0(n^2)

(5) 空间复杂度：S（n）=0(1)

(6) 直接选择排序是一种不稳定的排序

2.针对给定的输入实例，写出直接选择排序的排序过程

3.熟练写出简单选择排序的算法

8.5 快速排序

1.快速排序的基本思想和算法实现，以及在最坏和平均情况下的时间性能分析，

(1) 平均情况下，快速排序时间复杂度为 0(nlogn)，其平均性能最好

(2) 若初始序列有序或基本有序时，快速排序蜕化为起泡排序。时间复杂度为 0(n^2)

(3) 时间复杂度：T（n）=0(nlogn)

(4) 空间复杂度：S（n）=0(logn)

(5) 快速排序是一种不稳定的排序

2.针对给定的输入实例，能写出快速排序的排序过程

3.熟练写出快速排序的算法

8.6 堆排序和归并排序

1.堆的概念、堆排序和归并排序的基本思想和算法实现，以及性能分析，

1.针对给定的输入实例，能写出堆排序和归并排序的排序过程