**线性表顺序存储**

函数声明：Status Insert ( SqList &va, ElemType x )

表长：va.length

访问表元素：va.elem[i]

**线性表链式存储**

函数声明：Status Delet ( Linklist &L)

空表：L->next == null

访问元素：p = L->next;

p = p->next;

p->data;

while(p)

**堆栈顺序存储**

函数声明：Status Judge ( char a[] )

堆栈使用：

Stack st;

InitStack(st);

Push(st,a[i])

ElemType x;

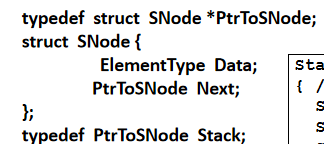
Pop(st,x);

GetTop(st,x);

StackEmpty(st);

DestroyStack(st);

**堆栈的链式存储（默认有头结点，直接用S->next）**



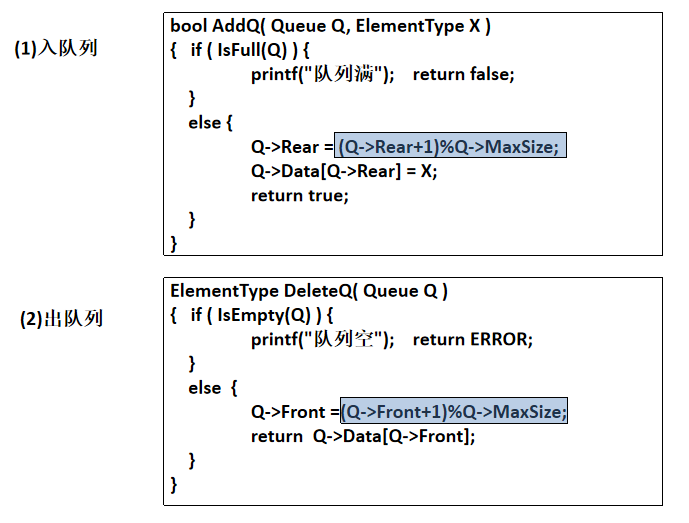
**队列的顺序存储（循环队列）**

1. >front

Q>rear

Q->Data[Q->rear]

Q->MaxSize



**队列做题使用：**

Queue q;

InitQueue(q,k) // k 是队列长度

ElemType x;

EnQueue(q,x);

DeQueue(q,x);

**广义表存储：**

函数声明： int PrintAtom ( GList L , int layer)

访问tag: L->tag(原子为0，表为1）

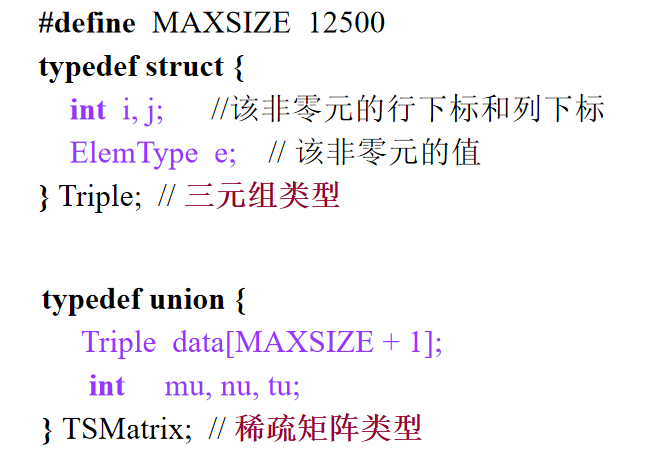
判断空表：if( !L )

访问其他：L->tag L->atom L->ptr.hp L->ptr.tp

注意递归表头要+1

**三元组存储：**

行优先，（i, j, aij） nXm的矩阵， 1 ≤ i ≤ n , 1 ≤ j ≤ m



**二叉树**

函数声明：Status CaculateLeafNodeNumber ( int &i, BiTree &T )

使用：T->lchild

1. >rchild

i是递归次数标志

判空：!T

**孩子-兄弟链表表示的树：**

int LeafNum( CSTree &T )

1. >firstchild
2. >nextsibling

**邻接表表示的图**

int FInd\_BFS ( ALGraph G )

常用循环：

for ( p = G.vertices[i].firstarc ; p ; p = p->nextarc )

k = p->adjvex;

顶点数G.vexnum

边数G.arcnum

访问顶点G.vertices[i]

常用全局变量int visited[MAXSIZE] 来表示顶点是否被访问过 visited[i] = 1表示i号顶点已被访问

**顺序表**

SSTable ST

ST.length

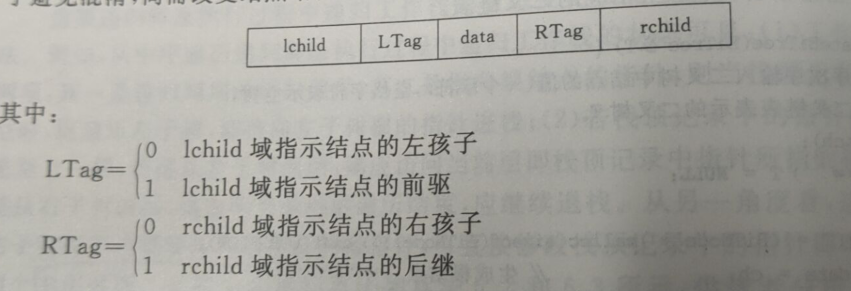
ST.elem[i]

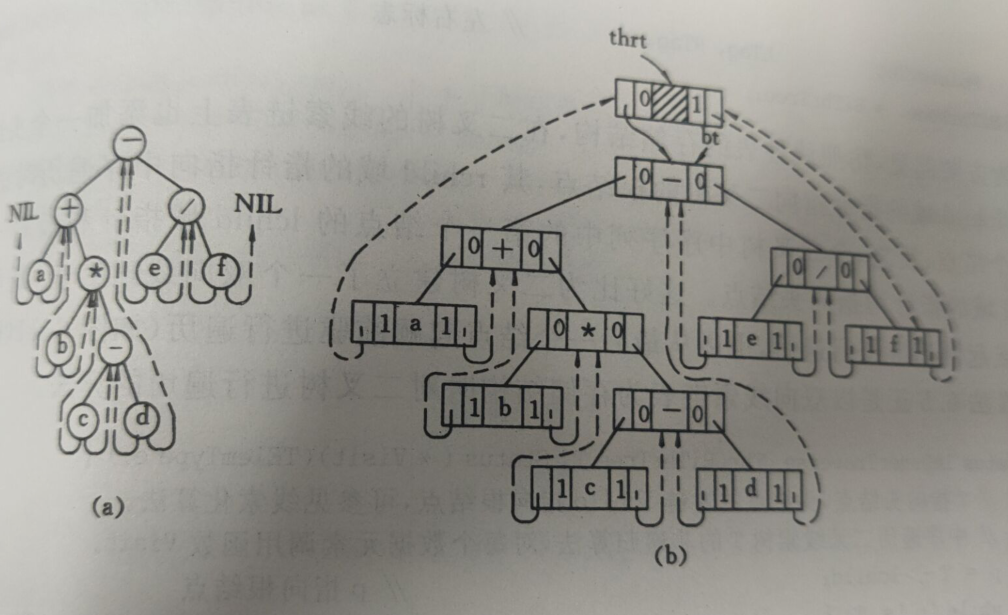
**第十章常用函数声明**

void sort(int a[], int n)

void sort(int a[],int low, int high)

**线索二叉树**





**堆：**

完全二叉树，从根到任意结点单增或者单减

插入：按照完全二叉树的要求进行插入，然后调整

删除：删除堆顶，然后用完全二叉树最后一个替代，然后进行调整

构建：从完全二叉树的最后一个非叶子节点开始调整

**哈夫曼树：**

WPL带权路径长度=叶子权值 X 根到叶子长度 求和

**关键路径：**



顶点表示状态，边表示活动，权值表示活动持续时间，上述是求顶点的

求边的最早最晚：比如求AB的最早=AB的最早开始=A的最早到达

AB的最晚=AB的最晚结束=B的最晚到达 - AB边权值

**迪杰斯特拉**

初始化

DIST PATH

0 0

无穷 无穷

无穷 无穷

.....

.....

找最小且未访问，更新

重复

**弗洛伊德**

先写D-1，自己到自己是0，自己直接到不了别人是无穷，到的了是权值。

再写P-1，第一行全是0，第二行全是1，.....

框出D-1的第一列第一行，更新，P-1的相应位置也更新，第一次更新填0，第二次填1，。。。

最后D[v][w]代表从v到w最短路径的权值，P[v][w]代表从v到w的最短路径中，w的前一个结点。

比如，P最终结果的第4行是 2 3 1 3

P[3][0]是2，说明④......①的前一个是③，④.....③.....①

再看P[3][P[3][0]]也就是P[3][2],是1，也就说④....③的前一个是②，也就是④....②.....③....①

再看P[3][P[3][2]] （注意这里第一个始终是3），也就是P[3][1]，是3，也就是④..②的前一个是④，这就对起来了

说以结果就是④...②...③...①

**顺序查找**

等概率情况下：

ASL=(n+1)/2

查找成功可能性与不可能相同，每个记录查找概率相等：

ASL=3/4 ×(n+1)

监视哨作用：

免去查找过程中每一步都要检测整个表是否检查完毕。当查找表长度很长时可以减少查找时间。

**折半查找**

调整时low+1或者high-1

mid = (low + high)/2下取整

ASL = (n+1)/n log2(n+1) -1

**二叉排序（查找、搜索）树构建、插入、删除：**

构建、插入实际上是先搜索然后在对应位置插入

删除：1.叶子直接删除2.一个孩子，删除+连接3.两个孩子，删除，用左子树最大（或右子数最小）替代，转变为2

**平衡二叉树插入：**

四种旋转，对于LR和RL，转两次， 第一次是关于-1，第二次关于-2，每次都有子树的挪动

**B-树**

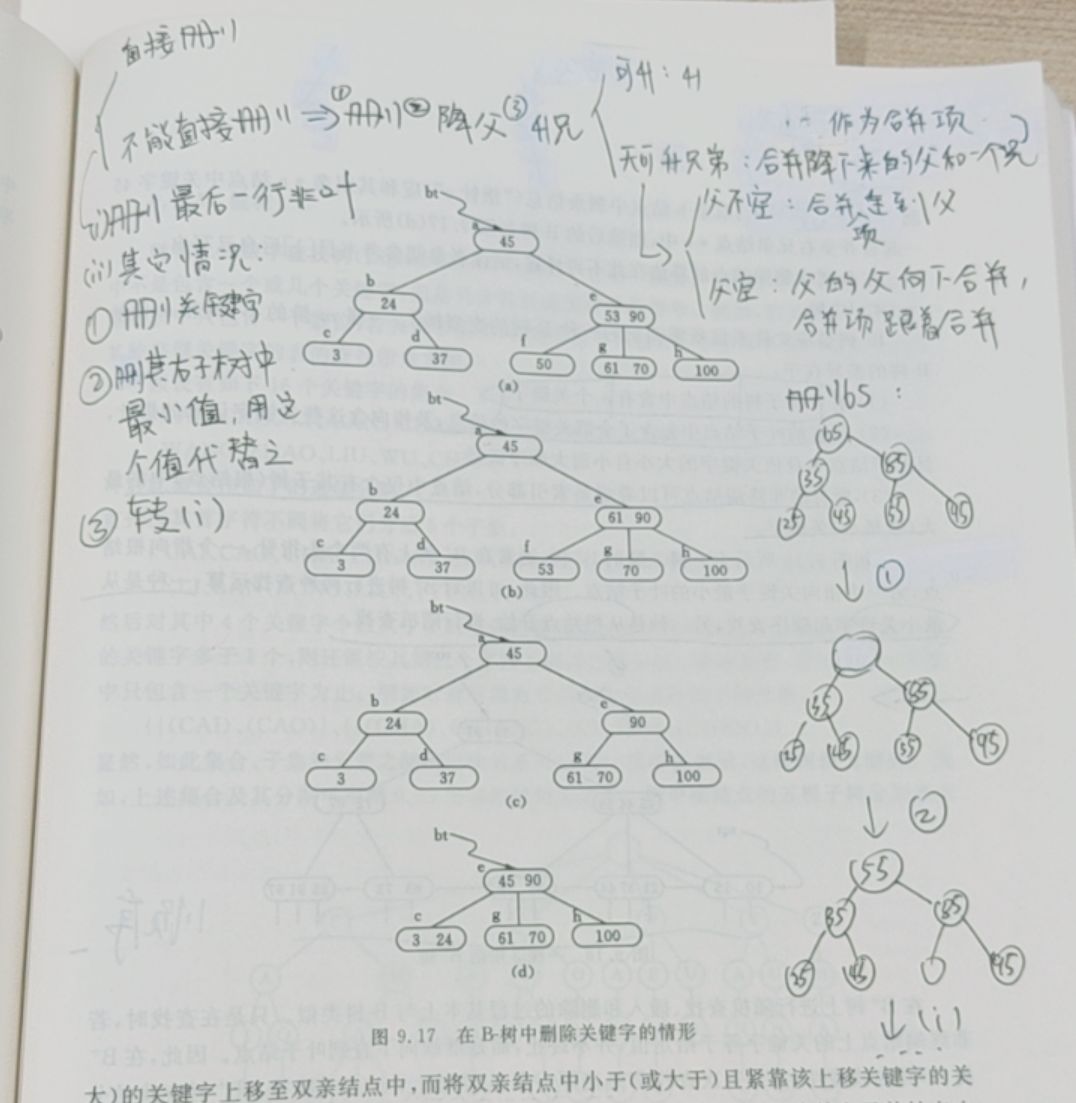
m阶的B-数，每个结点至多有m棵子树

若根节点不是叶子结点，则至少有两颗子树

除根之外所有非终端结点至少m/2山取整棵子树

求深度时注意还有叶子那一层

插入：升兄，连接

删除：

**注意是删除关键字而不是删除结点！**

ASL=1/n （∑结点层数×该层结点个数）

**哈希表：**

复杂度与装填因子有关，是α的函数，不是n的函数

**散列：**

散列函数：

直接定址法h(key) = a × key + b

除留余数法h(key) = key mod p

**处理冲突的方法：**

开放定址法（线性探测，二次探测、双散列）

Hi = (H(key) + di) **mod n**

线性探测再散列：d=1,2,....

平方探测（二次探测）再散列：d = 1^2,-1^2,2^2,-2^2......

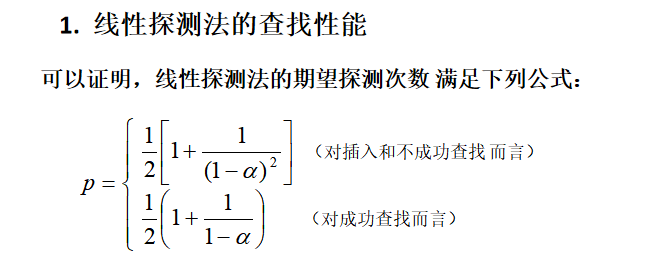
伪随机探测再散列：

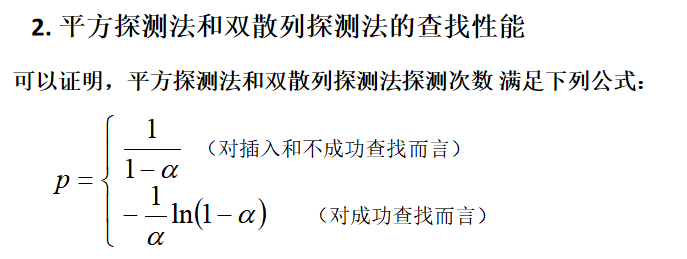
双散列探测法：di = i \* h2(key),h2是另一个散列函数

= h2(key), 2h2(key), 3h2(key)

再散列：当装填因子过大时，解决的方法是加倍扩大散列表，这样α可以减小一半，这个过程叫做“再散列（Rehashing）”。当然，装填因子过小时，比如 α<0.3，会浪费空间，此时散列表大小可以减半。

分离链接法是解决冲突的另一种方法，其做法是将所有关键词为同义词的数据对象通过结点链接存储在同一个单链表中





10.7各种排序方法讨论（很重要，参照PPT）

