一．绪论

数据元素是数据的基本单位（结点，顶点）

数据项是数据的不可分割的最小单位

数据对象是由性质相同的数据元素组成的数据的子集

数据的逻辑结构：集合，线性结构，树形结构，网状结构

数据结构：

1. 线性结构：线性表，栈，队列，数组，字符串
2. 非线性结构：树，二叉树，图

数据的存储结构：1.顺序存储结构（向量，一维数组） 2.非顺序存储结构（链接表，链式存储结构） 3.索引存储结构 4.哈希存储结构

数据类型：原子类型（int,char,float），结构类型（数组，结构，联合体）

算法的五个特征：有穷性，确定性，可行性，输入，输出

算法设计要求：正确性，可读性，健壮性，高效率与低存储量

O(1)<O(logn)<O(n)<O(nlogn)<O(n^2)<O(n^k)<O(2^n)<O(n!)<O(n^n)

空间复杂度是对一个算法在运行过程中临时占用存储空间的大小的量度

二．线性表

typedef struct线性表的静态分配（所占空间和表长）

{

ElemType elem[maxleng];

int length;

}SqList;

typedef struct线性表的动态分配

{

ElemType \*elem; //存储空间基地址

int length; //表长

int listsize; //当前分配的存储容量(以sizeof(ElemType)为单位)

}SqList;

ai的存储地址是LOC(i)=LOC(1)+(i-1)\*p

插入，判断i值是否合法，顺序栈是否溢出，从最后一个往后覆盖，最后长度要加一

带头结点的链表1.第一元素插入 2.一般插入和尾元素插入

用前后双指针生成带头结点的递增有序单链表

free释放的是指针指向的空间

带表头结点的空循环单链表H->next=H,H!=NULL;

所有表插入新结点都要让新结点的前后指针指对地方

多项式的链表表示方法

|  |  |  |
| --- | --- | --- |
| coef | expn | next |

系数 指数 下一结点

三．栈和队列

顺序栈top指向栈元素上一空位置

typedef struct

{

ElemType elem[maxleng]; //栈元素空间

int top;顶指针

}sqstack;

typedef struct

{

ElemType \*base;指向栈元素空间

int top;顶指针

int stacksize;当前分配的栈空间大小

}SqStack;

当前算符栈顶元素的优先级大于要进栈的算符，进行出栈运算

typedef struct Qnode存放元素的结点类型

{

ElemType data;

struct Qnode \*next;

}Qnode,\*QueuePtr;

typedef struct由头尾指针组成的结点类型

{

|  |
| --- |
| front-> |
| rear-> |

Qnode \*front;

Qnode \*rear;

}LinkQueue;

进队前测试，如果r+1==f表明是满队列，r指向最后一个元素的前一个

(Q.rear-Q.front+maxlength)%maxlength计算队列长度

四．串

一个结点放四个字符的串

struct node4

{

char data[4];

struct node4 \*next;

}\*ps2;

朴素匹配算法O(m+n)

while(i<=S[0]&&j<=T[0]) {i=i-j+2;j=1;}

if(j>T[0]) return i-T[0];

ababaaaba前面有n个重复的就1+n

011234223

while(i<S[0]&&j<=T[0]){j=next[j];}

if(j>T[0]) return i-T[0];

五．数组和广义表

m行n列，以行序为主序，a[i][j]的地址为Loc(I,j)=Loc(1,1)+(n\*(i-1)+j-1)\*s

typedef struct

{  
 int I,j;

ElemType e;

}Triple; //定义三元组

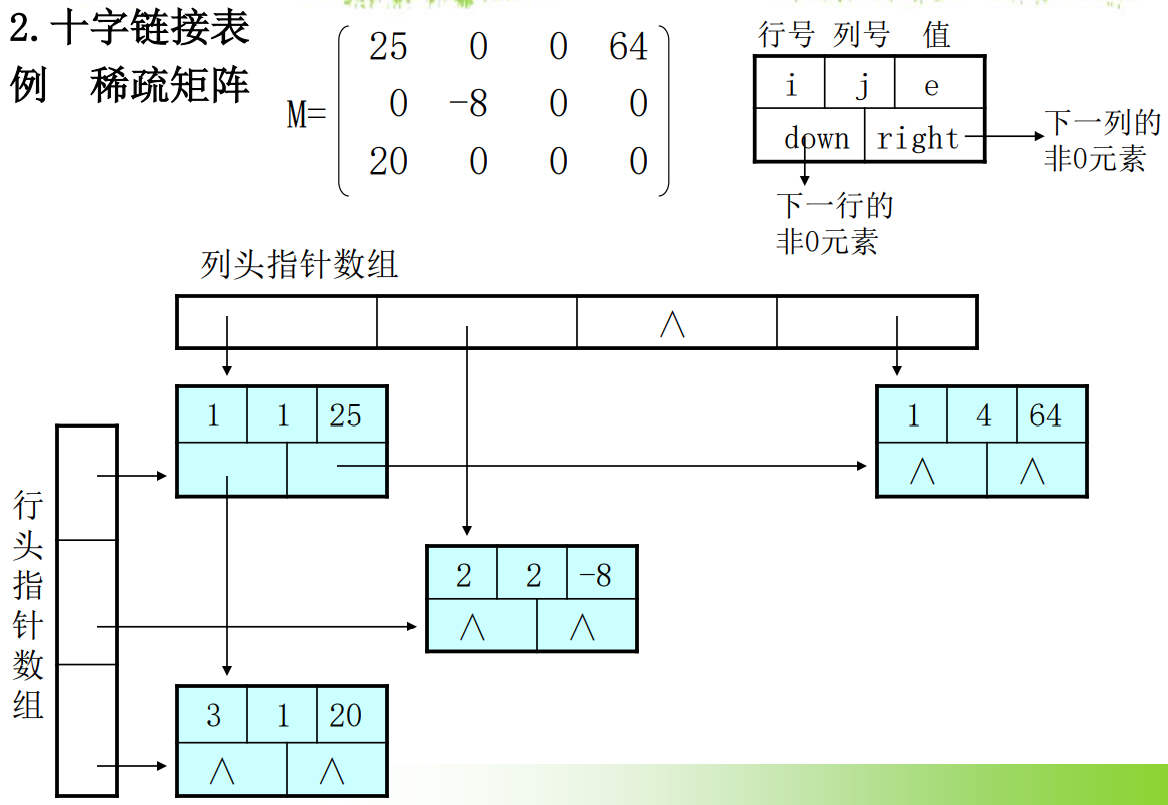
typedef struct

{

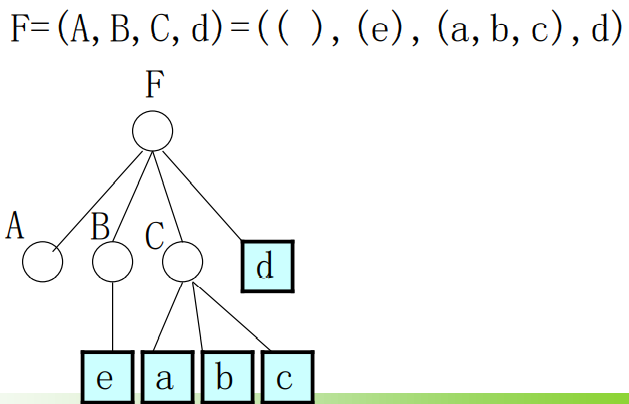
Triple data[MAXSIZE+1];

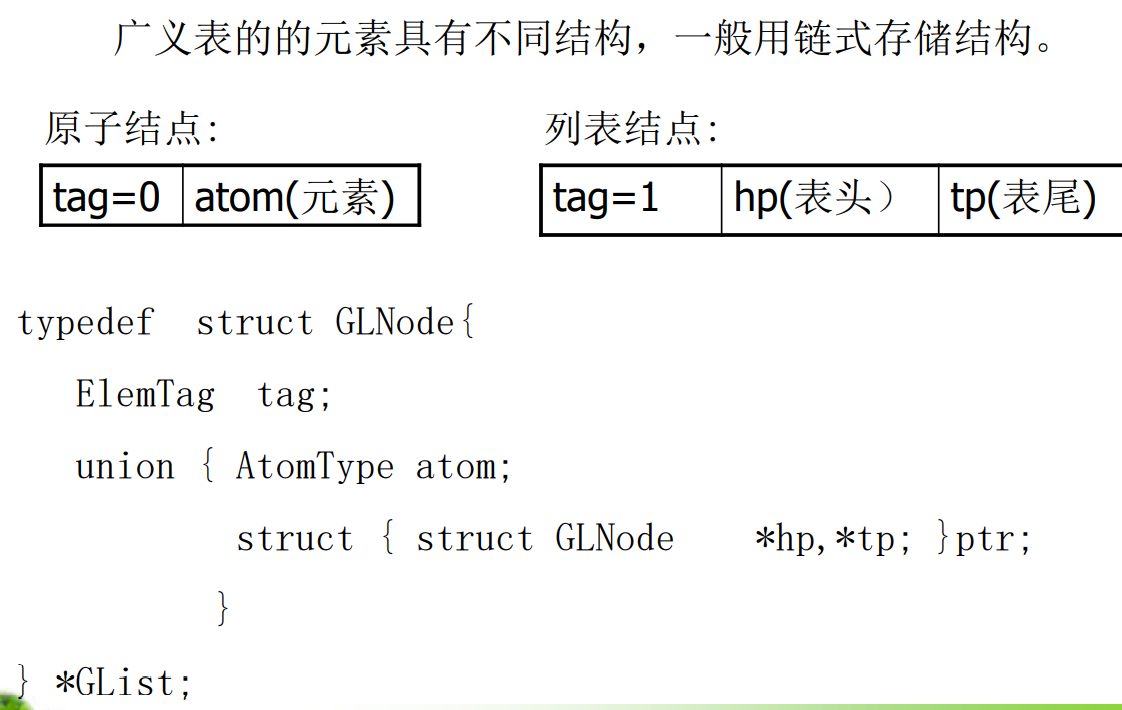
int mu,nu,tu; //tu非零元个数

}TSMatrix; //定义三元组表



广义表的表头只有一个，表尾有很多，多少层就数最多有多少层括号（一般在最前面或者最后面）

广义表的图形表示



六．树

根的层为1

结点i的双亲是i/2取下整，左孩子是2i

结点I的层号是log2i取下整+1或者log2(i+1)取上整

struct BiTNode //三叉链表结点类型

{

struct BiTNode \*lchild,\*rchild,\*parent; //左孩子指针

ElemType data;

}\*root,\*p;

struct bnode2 //静态链表

{

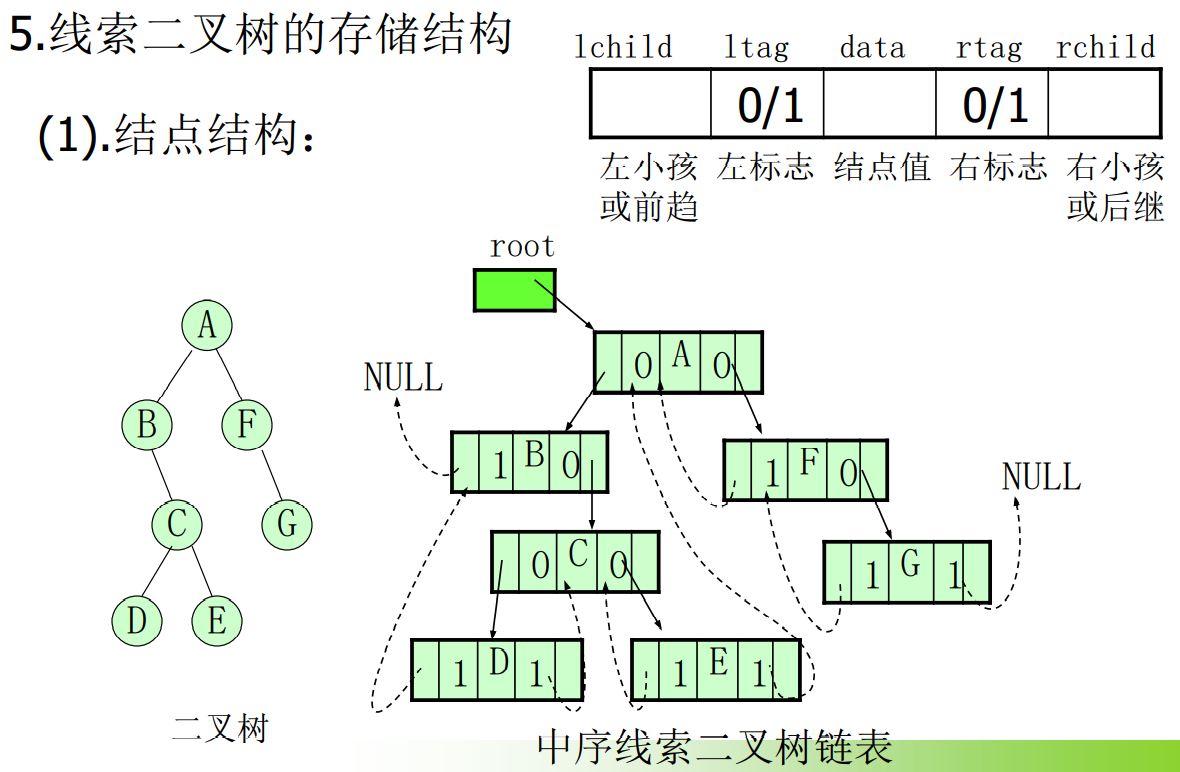
ElemType data;

int lchild,rchild;

}t[n+1];

中序遍历非递归，左子树遍历结束后，第二次遇到根结点，就将根结点退栈并访问，然后用中序遍历它的右子树，如果栈为空则结束 do{while(t); if(top)t=t->rchild;}

先序遍历非递归，while{p=St[--top];St[top++]=p->rchild;}

0是原始的是左孩子，1是变化的是左前趋

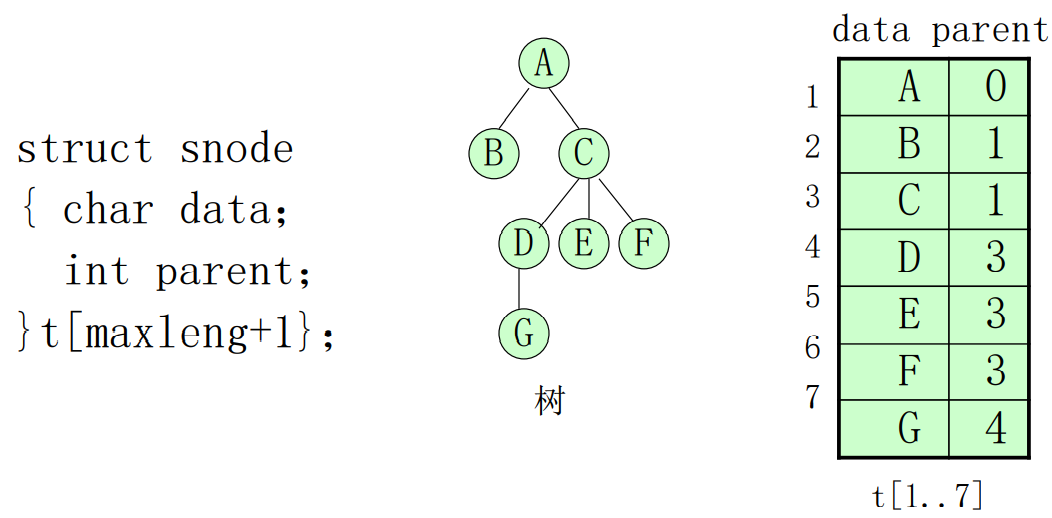
先序线索二叉树的遍历if(p->ltag==0)p=p->lchild; else p=p->rchild;如果没有左孩子就移向右线索（往后遍历）或者右孩子

中序线索二叉树的遍历while(p->rchild!=NULL){ //p有后继结点

if(p->rtag==1)p=p->rchild; //p无右孩子时为线索

else{p=p->rchild;while(p->ltag!=1)p=p->lchild} //如果p有右孩子，取右子树最左结点}

求二叉树中结点个数，if(root){n=1;n+=preorder(root->lchild);}



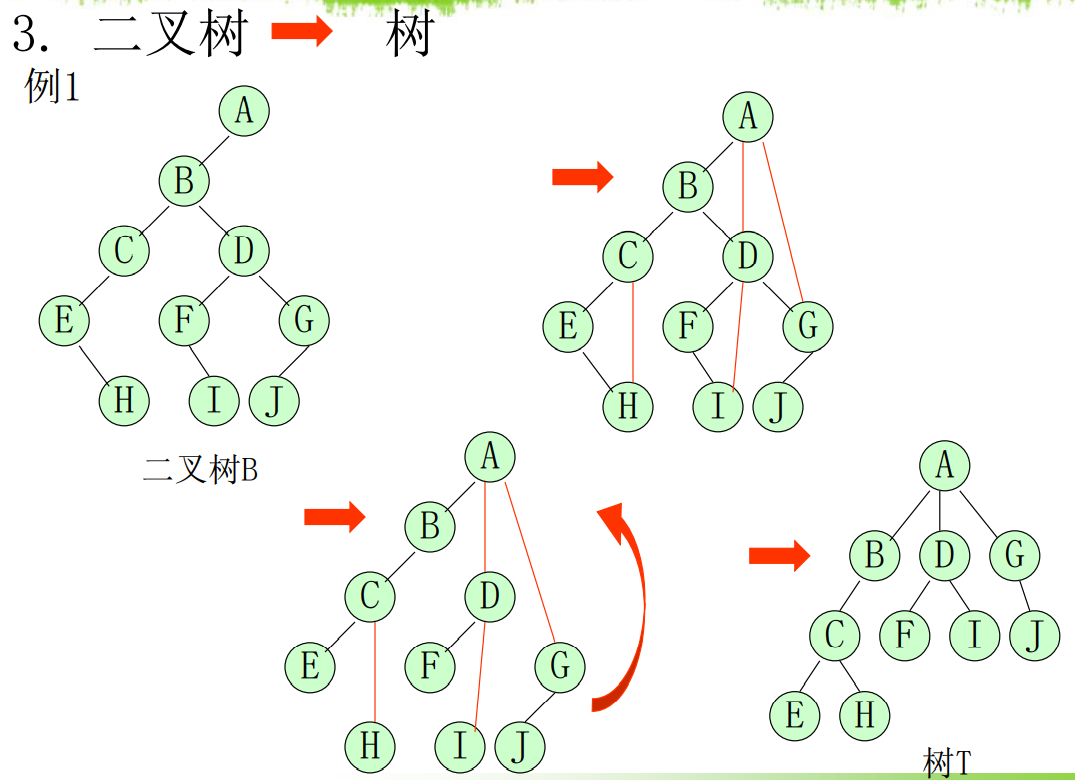
带双亲的孩子链表表示法

|  |  |  |
| --- | --- | --- |
| 结点值data | parent | first->第一个孩子 |

上面是表头结点，下面是孩子结点/表结点（被first指向）

|  |  |
| --- | --- |
| 序号值 | next下一个孩子 |

树->二叉树：1.在兄弟之间加线 2.保留根与最左孩子之间的连线，删除与其他孩子之间的连线



树T的路径长度PL(T)—从树T的根到其余每个结点的路径长度之和(根到自己是0)

树的带权路径长度—每个叶子的权与根到该叶子的路径长度的的乘积之和WPL(T)

哈夫曼树，合并权最小的二叉树，并排序，编码左0右1

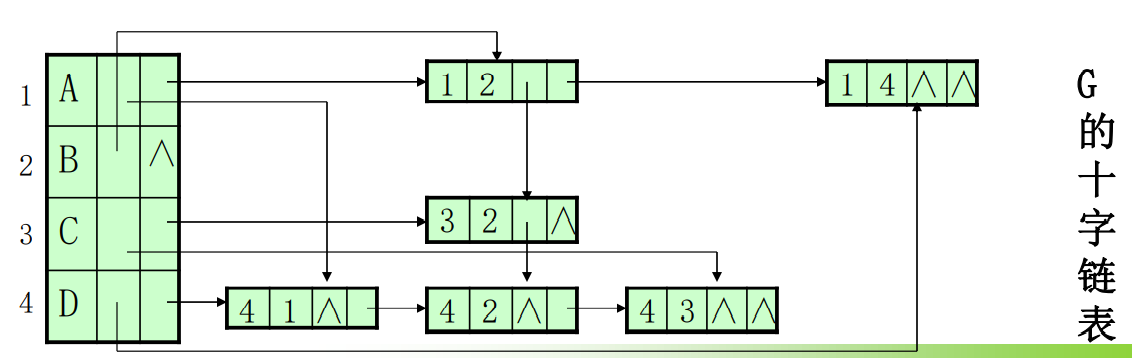
七．图

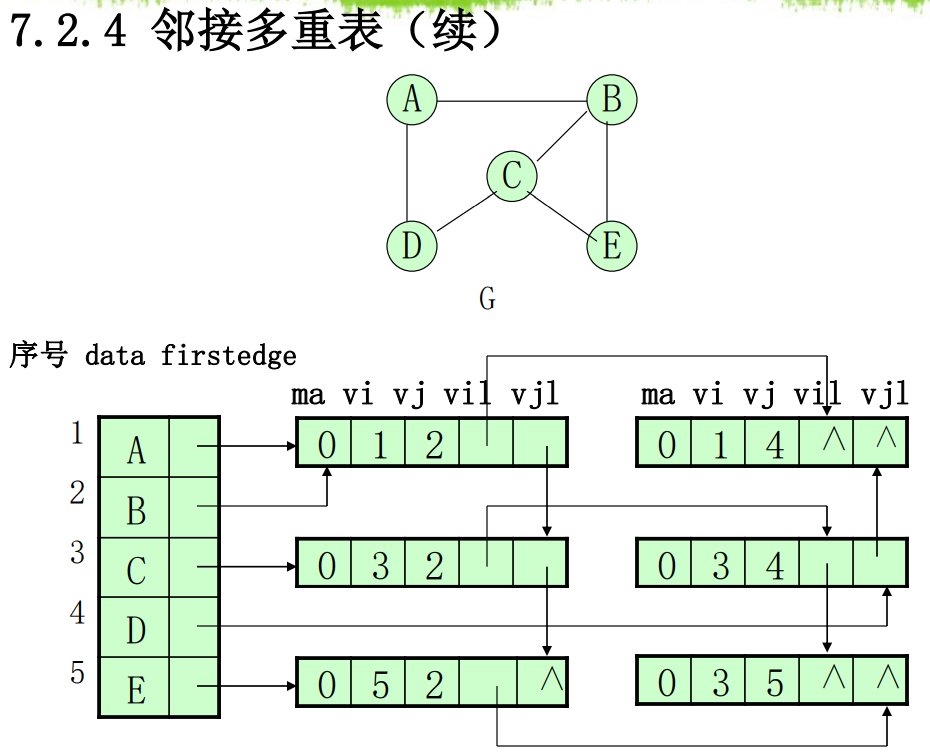
<a,b>a是弧尾，b是弧头

网是边上加权的图

强连通图（每对顶点之间都有双向路径）是对有向图而言，生成树是极小连通子图

邻接矩阵每列元素之和是入度ID，每行元素之和是出度OD



mark用来标记这条边是否被访问过

DFS回溯到前面搜索过的点，BFS用栈，先进的下一层要先遍历完

拓扑排序1.选一个没有前趋的顶点输出（选择入度为0的顶点） 2.从图中删除该顶点和所有以它为尾的弧

路径长度最长的路径叫做关键路径，有多个前趋选择前趋加路径最长的，从前往后顶点选最大的，从后往前顶点选最小的，边的最迟开始时间l==边的最早开始时间e，就是关键路径（也可以直接比较每一条路加到最后一个点的路径最长的）

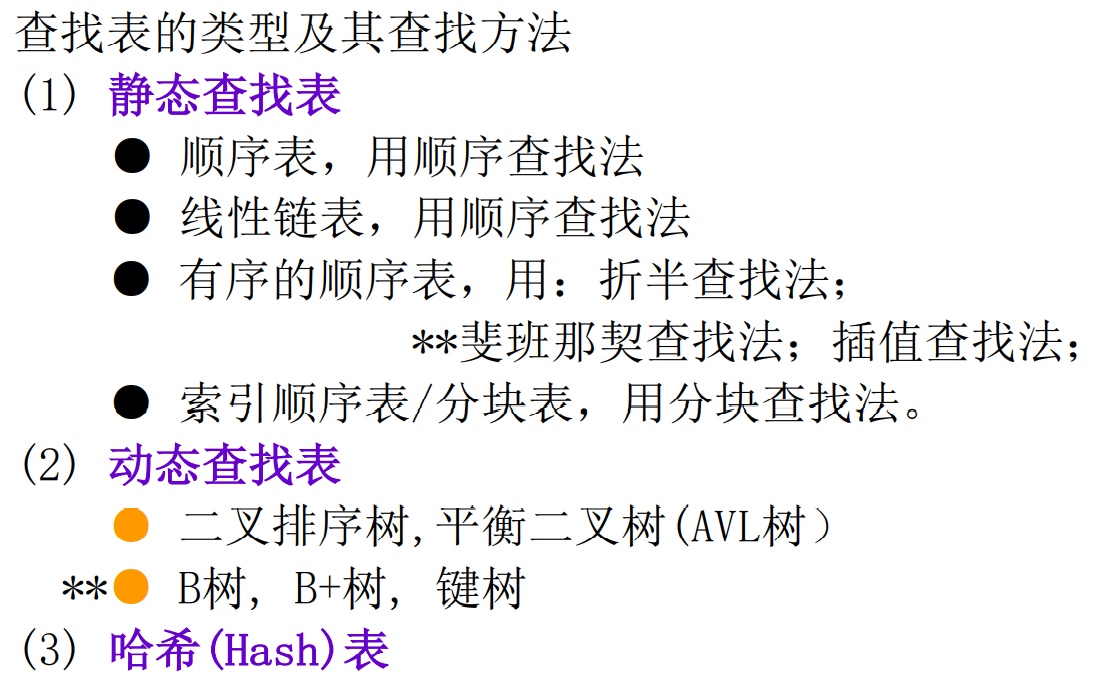
单源最短路径，狄克斯特拉算法，选最短的顶点后更新到源头顶点的最短路径

Floyd多元最短路径

for(k=0;k<N;k++)for(i=0;i<N;i++)for(j=0;j<N;j++) if(cost[i][j]>cost[i][k]+cost[k][j]) cost[i][j]=cost[i][k]+cost[k][j];

九．查找

主关键字：可以唯一地辨识一个记录的数据项 次关键字：可以识别若干记录的数据项

二叉排序树不需要插入相同结点

平均查找长度ASL等于概率(除以个数)乘以查找所需比较关键字的次数（和NULL进行的比较也要算一次）

判定树是描述折半查找过程顶点二叉树ASL=(n+1)/n \*log2(n+1)-1

最佳分块：n个记录分成/--n块 ASL=/--n+1

被删结点左右子树都存在，寻找右子树的最左孩子，用它的值填补到被删结点中，再来处理结点的删除问题，删除的最好时间复杂度：ASL=(n+1)/n \*log2(n+1)-1，最坏情况(n+1)/2

平衡因子：左右子树的深度之差，平衡二叉树：任意结点的平衡因子的绝对值小于等于1

typedef struct node //平衡二叉树的结点

{

DataType data;

int balance; //多了结点平衡因子域

struct node \*leftChild,\*rightChild;

}AVLNode;

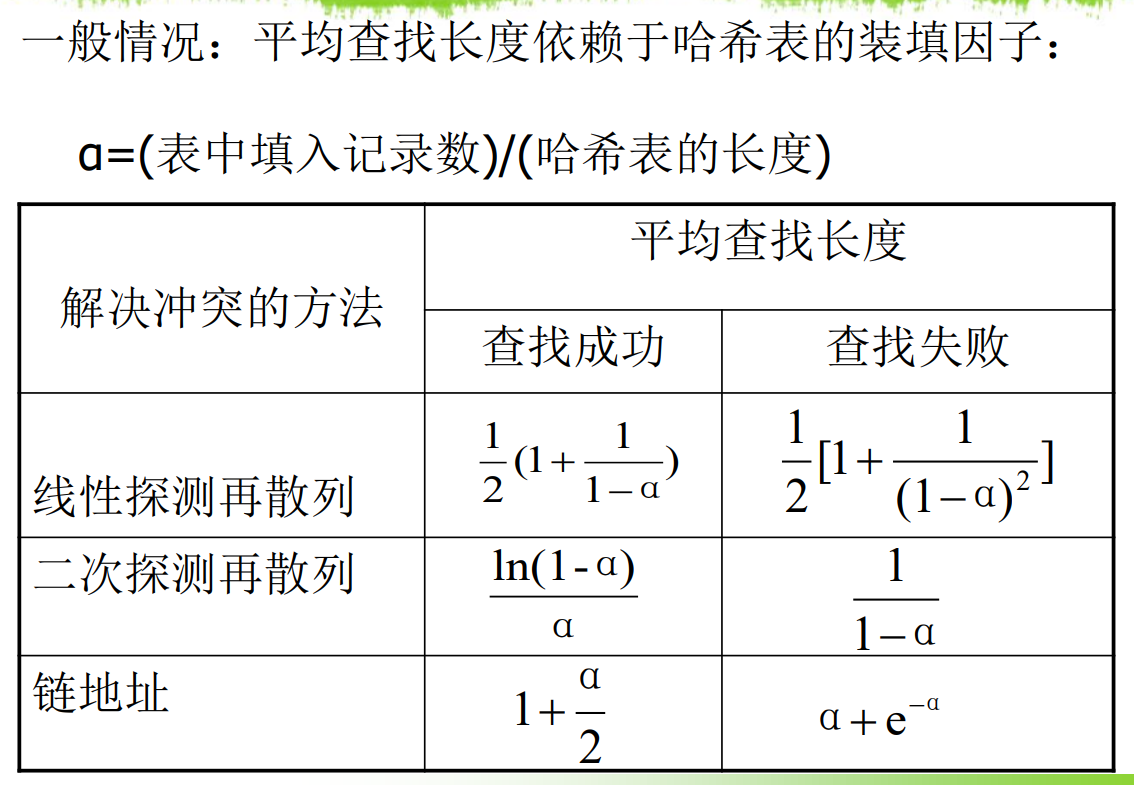
从发生不平衡的结点起，沿着刚才回溯的路径取直接下两层的结点，双旋转是把折线先转成在一条直线上

数字分析法，取关键字的若干分布均匀的位组成哈希地址

链地址法，将关键字为同义词的所有记录存入同一链表中（头插）

二次探测再散列：存入q+i^2中

查找失败是从哈希地址一直到空白，查找成功是从哈希地址到存放地址（和关键字的比较次数）



十．内部排序

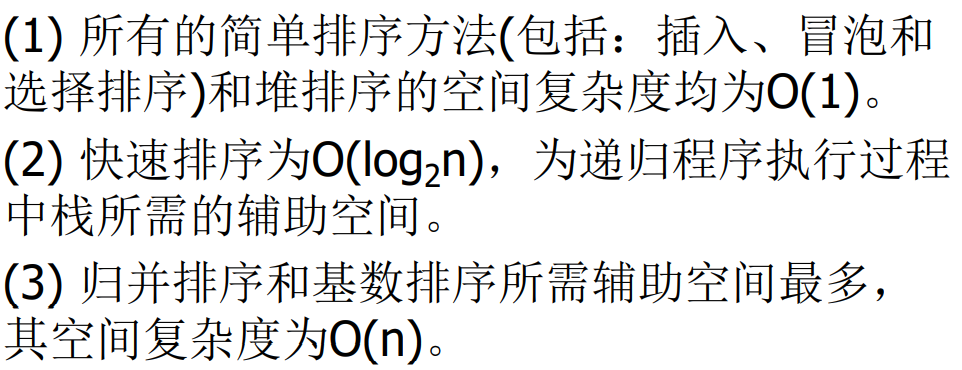
时间复杂度为O(nlog2n)的排序：快排，堆排，归并排序（n值较大，归并排序比堆排序更快）

时间复杂度为O(n^2)的排序中选排记录移动次数最少（先找到最小值放最前面）

时间复杂度为O(n)的排序只有基数排序（适合n值很打，位数较小的）

有序，冒泡和插入优化为O(n)，快排退化为O(n^2)

选排，堆排，归并排序的时间性能不随记录序列关键字的分布而改变



希尔，快排，直接选择和堆排是进行大步幅跳跃的不稳定排序



背直接插入，快排，堆排，二路归并代码

冒泡for(i=0;i<n-1;i++) for(j=0;j<n-1-i;j++)

用大顶堆可以实现序列的递增排序，用小顶堆可以实现序列的递减排序