第一章 绪论

1.1 时间复杂度的求法

（一）循环主体中的变量参与循环条件的判断

a) 找出基本操作

b) 设基本操作执行次数为T(n)，根据初始条件和基本操作语句确定变量与次数的关系式

c) 带回循环条件，求出T(n)，确定O(n)

（二）循环主体中的变量与循环条件无关

（1） 递归程序

a) 确定递推关系（注意这里确定的是基本操作次数的递推关系）

b) 推出递推关系与执行次数的表达式

c) 令低级递推关系中的次数为常数（0或1），整理式子

d) 推导出T(n)

（2） 非递归程序

等比、等差数列求和

1.2 实例

王道单科P8，综合第二题

1） 归类：循环主体中的变量参与循环条件的判断

基本操作：i++(注意不是k参与循环条件的判断)

初始条件i=1,执行一次i加一，值为2;执行第二次，i再加一，值为3；…；执行T(n)次，i的值为T(n)+1;

带回循环变量：i<n-1,终止条件为i=n-1;既T(n)+1=n-1

推出T(n)=n-2,所以T(n)=O(n)

2） 归类：循环主体中的变量参与循环条件的判断

基本操作：y=y+1

初始条件y=0;执行一次y加一,执行T(n)次后值y=T(n);

带回循环变量：(T(n)+1)*(T(n)+1)>n

推出：T(n)=n½-1 所以 T(n)=0(n½)

3） 归类：循环主体中的变量与循环条件无关,非递归程序

T(n)= ∑∑∑1=0(n^3)

4） 归类：循环主体中的变量与循环条件无关,非递归程序

T(n)=M*N=0(M*N)

综合题第一题

归类：循环主体中的变量与循环条件无关,递归程序

递推关系已给，题中没给的要自己推出来

T(n)=2T(n/2)+n————①（注意：这里的 n,2 等都是执行次数，不是变量的值）

T(n/2)=2T(n/2*2)+n/2 ————②

把②带回①得到 T(n)=2*2*T(n/2*2)+2*n

令 T(n/2*2)中令 n/2*2=1，解出 n=2*2,2=Log2 n

T(n)=n*T(1)+n*Log2 n = n+n*Log2 n = 0(n*Log2 n)

P7 第 4 题

归类：循环主体中的变量参与循环条件的判断

基本操作：x=x*2

初始条件：x=2,执行一次后 x=2*2,执行两次后 x=2*2*2,…,执行 T(n)次后， x=2^T(n)

带回循环变量:2^T(n)=n/2

T(n)= Log2(n) - 1 = 0(Log2(n))

第 5 题

归类：循环主体中的变量与循环条件无关,递归程序

基本操作：n*fact(n-1)

递推关系：T(n)=1+T(n-1)（这里 1 为上面的基本操作执行了一次）

T(n-1)=1+T(n-2)，代入上式得到：T(n)=1+1+T(n-2)

令 T(n-2)中 n-2=0,则 2=n

原式整理为：T(n)=n+T(0)（这里表示的是次数的变化，即我每次减一，前面就加一，减到 n 时，前面也加到 n）

T(n)=n =O(n)

第二章 线性表、栈、队列

2.1 各种链表特殊操作的时间复杂度

DS 复杂度 操作	删除最后元素	删除第一个元素	在最后插入元素	在最前插入元素
单链表	O(n)	O(1)	O(n)	O(1)
循环单链表（头指针）	O(n)	O(1)	O(n)	O(1)
循环单链表（尾指针）	O(n)	O(1)	O(1)	O(1)
双链表（头指针）	O(n)	O(1)	O(n)	O(1)
双链表（尾指针）	O(1)	O(n)	O(1)	O(n)
双链表（头、尾指针）	O(1)	O(1)	O(1)	O(1)
循环双链表	O(1)	O(1)	O(1)	O(1)

注意：若题中选项出现多个时间复杂度合适选项，选择修改指针最少的。

2.2 链表的指针修改原则——不断链原则

先定义一下指针（个人定义）

主链接性指针——通过已知指针（头指针或尾指针）和该指针可以链接操作所有参与元素，即该指针一断，元素失去控制（断链）；

非主链接性指针——断开后不影响元素链接操作；

（1） 对只有主链接性指针的链表操作步骤

a) 建立新的主链接性指针

b) 修改旧主链接性指针

（2） 既有主链接性指针又有非主连接性指针

a) 修改非主链接性指针

b) 建立新的主链接性指针

c) 修改旧主链接性指针

2.3 链表算法设计

常用方法：头插法；尾插法；双指针；多指针

（1） 删除

删除一个链表元素时，能同步找到它的直接前驱是最高效的，而如何实现同步直接影响算法的复杂程度

（2） 建表

头插法；尾插法；双指针，各有各的特点，自己总结吧

（3） 查找

这几次考的都是通过双指针的距离查找元素位置

（4） 排序

对无序链表排序，在空间复杂度为O(1)的条件下，时间复杂度最佳为O(n^2)

若算法设计是对当前排序的元素操作，则总体复杂度<=O(n^2)

若算法设计需要用到排序后的结果，则总体复杂度>=O(n^2)

2.4 顺序表算法设计思想

(1)双指针

a) 元素之间的距离，或利用该距离找元素

b) 元素值之间的比较

c) 删除某一个或某一范围的值

(2)置换

a) 顺序表中部分元素之间的位置互换

b) 改变部分元素次序

c) 移动部分元素次序

(3)折半

a) (对有序表)所求元素或所涉及操作与中间元素有关

b) 查找

2.5 静态链表

静态链表是借助来描述线性表的链式存储结构，静态链表表明使用数组实现线性表的操作不一定要移动元素，也可以“修改指针”。

2.6 栈

（一）顺序栈

（1） 注意栈顶指针 top 的位置，一般初始 S.top=-1；栈满 S.top=MaxSize-1;栈长 S.top+1

（2） 进栈判满，出栈判空;进栈时指针先加 1，数据再进栈；出栈时数据先出栈，指针再减 1

（二）共享栈

两个栈的判空：top0=-1;top1=MaxSize

判满：top1-top0=1

最大优点：只有整个存储空间被占满时才上溢,对存储效率无影响

（三）链栈——操作受限的链表（插入最前元素，删除最前元素）——之前有贴总结过，自己找吧

（四）出栈的合法性（*）：

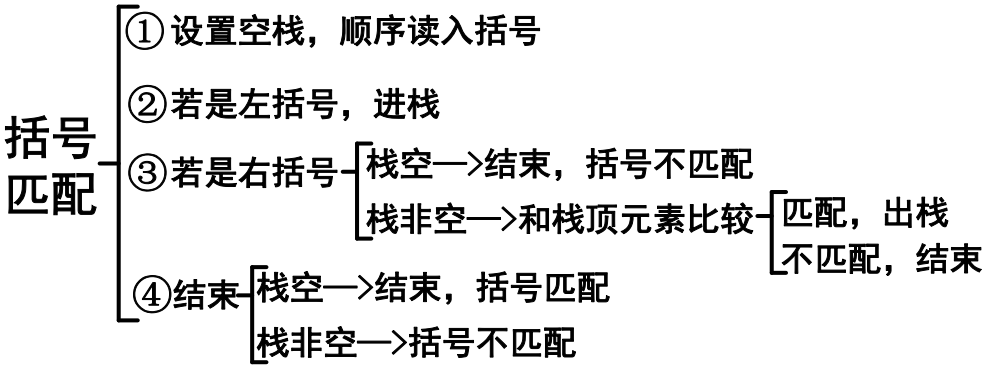
1) $\frac{1}{2n-1}C_{2n}^{n}$

2) 若进栈顺序为 1,2,3,4,…,i,…,n，出栈时若 i 出栈，则满足下面两个条件的：1.在 i 之前进栈的元素；2.在 i 之后出栈。则一定以进栈的逆序排列在 i 之后（或紧凑或分散）

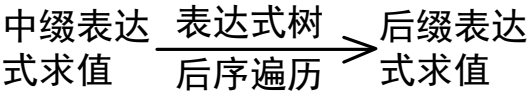
3) 出栈元素“个数”限制，若 i 出栈，i 之前元素在 i 之后出栈，则 i 之后的元素个数一定大于等于 i-1

（五）应用

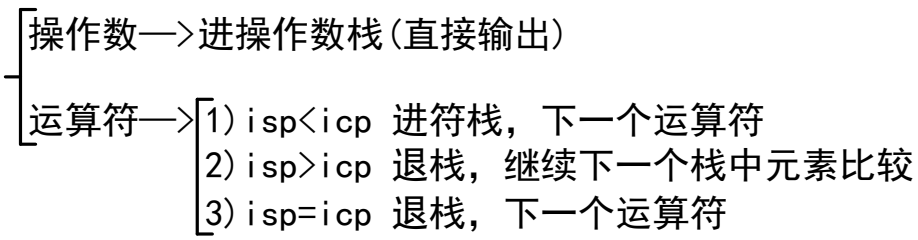
1) 括号匹配——最基本的应用了栈的特点



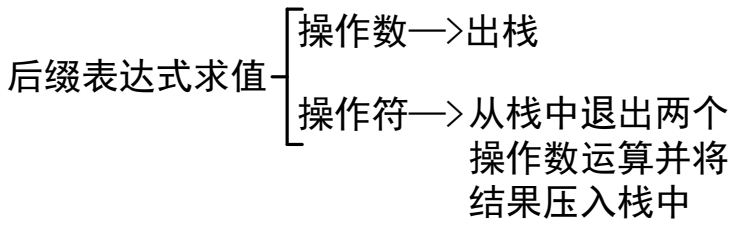
2) 表达式求值——重点在中缀表达式转化为后缀表达式——仔细看看 P85 11 题



(从内层运算到外层运算进行构造)



操作符	#	(	*/	+-	)
isp	0	1	5	3	6
icp	0	6	4	2	1



3) 递归——要明白高级语言中函数的调用就是通过栈实现的——递归的利用使程序的效率变低

2.7 队列

(一) 顺序队列

(1) 队首指针和队尾指针规定指在什么位置，根据题或要求看仔细了

(2) 判空和判满（依赖于第一条）

(3) 进队和出队操作（依赖于第一条），王道单科书 P72 中的操作都是 1.判满或判空 2.操作数据 3.修改指针，但这不是默认的，你看 P76 第 8 题，就是 2011 真题，它是先修改指针，再操作数据的

(二) 循环队列

(1) 逻辑——顺时针的环

(2) 实现——除法取余运算

(3) 基本操作

a) Q.front=Q.rear=0

b) Q.front=(Q.front+1)%MaxSize

c) Q.rear=(Q.rear+1)%MaxSize

d) 队列长度=(Q.rear-Q.front+MaxSize)%MaxSize

e) 判空：Q.front=Q.rear

f) 判满：

①牺牲一个单位：Q.front=(Q.rear+1)%MaxSize

②添加标志位（Q.size 或 tag，其实原理一样）

(三) 链式队列

（1） 操作受限的链表——删除最前元素，插入最后元素（不同链表的时间复杂度看前面的贴，题中问哪种链表适合做队列或栈的其实就是问复杂度）

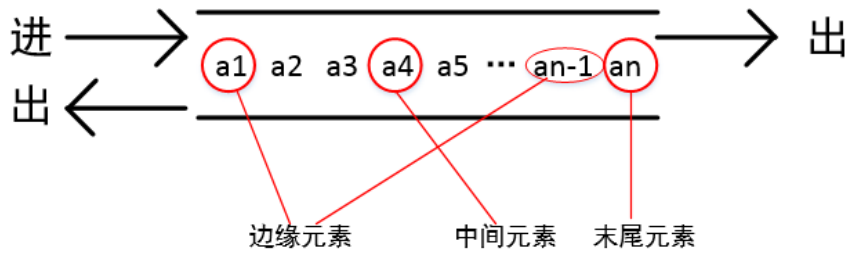
（2） 操作时指针的修改原则，做队列的题时，要掌握几种队列的模型，能随手画出来，还有最重要的：题中要求队首指针和队尾指针指在哪（是指向元素还是元素上一个位置或者元素下一个位置）

(四) 双端队列

(1) 不受限制的双端队列

n 个元素进队，出队序列的方式共 n！种，即 n 个数全排列

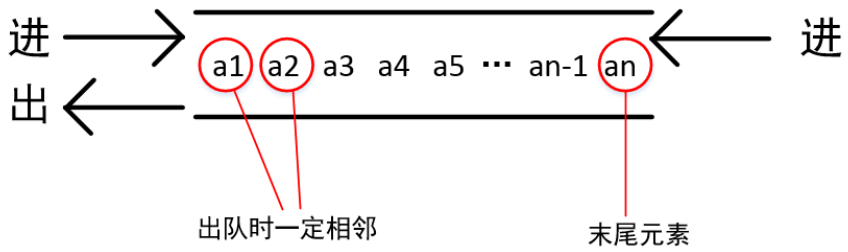
(2) 输入受限的双端队列



a） 增加的出队方式为 $n!-\frac{1}{2n-1}C_{2n}^{n}$ 的子集

b） *若末尾元素（这里指进队序列的最后一个元素）最先出队--→进队序列固定--→在出队序列中 an（即末尾元素）的下一个元素只能是边缘元素，不可能是中间元素（注意这里的边缘和中间是相对而言的）。

(3) 输出受限的双端队列



a） 增加的出队方式为 $n!-\frac{1}{2n-1}C_{2n}^{n}$ 的子集

b) *若末尾元素（这里指进队序列的最后一个元素）最先出队--→全部元素进队后再出队--→入队时最前面的两个元素在出队时一定相邻

如果上面的你明白了，那么再看单科书 P74-75 就简单了

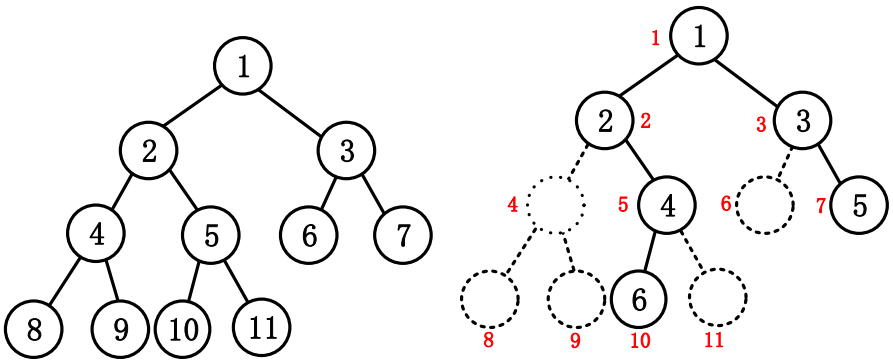
书中以 4 个元素为例，输入受限的队列为什么得不到 4 2 3 1 和 4 2 1 3 ，因为若 4 最先出队，2 为中间元素不可能在 4 的后面；输出受限的队列为什么得不到 4 2 3 1 和 4 1 3 2 呢，因为若 4 最先出队，1 和 2 必相邻。

再看 P77 17 题，元素依次入队后在出队，则 a 和 b 一定在出队序列中相邻，你就会快速准确的选出 C，因为 a 和 b 不相邻，而且一定正确，比你一个一个试快得多。

第三章 树和二叉树

3.1 二叉树的存储

(一) 顺序存储



完全二叉树　　二叉树

1	2	3	4	5	6	7	8	9	10	11
1	2	3	0	4	0	5	0	0	6	0

重要性质：

1) 数组下标从 1 开始
2) i>1 时，i/2 是双亲
3) 2i<=N，i 的左孩子为 2i，否则无左孩子
4) 2i+1<=N，i 的右孩子为 2i+1，否则无右孩子

5) 结点 i 所在层次为 $\lfloor \log_2 i \rfloor + 1$

（二） 链式存储

二叉链表，n 个结点，含 n+1 个空链域。

3.2 树和二叉树的计算

最核心的公式：(1)n=B+1

(2)n=n0+n1+n2+…+nk

(3)B=0*n0+1*n1+2*n2+…+k*nk

（一）树与二叉树高度与结点数的相互计算

	树（度为 m）		二叉树
第 i 层结点数	m^{i-1}		2^{i-1}
高度为 h 求结点数 n	至多	$\frac{m^h-1}{m-1}$	2^h-1
	至少	h+(m-1)	h
结点数为 n 求高度 h	最小	$\lceil \log_m(n(m-1)+1) \rceil$	$\lceil \log_2(n+1) \rceil$
	最大	n-(m-1)	n

部分树和二叉树公式其实是通用的把 m 换成 2 就可以，除了结点的至少数和高度的最大数，这是由于二叉树的特殊性质决定的，因为二叉树不是简单的度为 2 的树。

（二）n 个结点的完全二叉树

	奇偶	度为 0 的结点-叶节点	度为 1 的结点	度为 2 的结点
n 个结点的完全二叉树	n 为奇数	$\frac{n+1}{2}$	0	$\frac{n-1}{2}$
	n 为偶数	$\frac{n}{2}$	1	$\frac{n}{2}-1$

注意： 叶结点数量最多

完全二叉树的叶结点只能出现在最后两层中，所以题中出现第 k 层有 Nk 个叶结点时，此时树的高度为 k 或 k+1

1） 为 k 时，此时树的结点数为：k-1 层的满二叉树结点数+Nk

2） 为 k+1 时，此时树的结点数为：(k 层结点数-叶结点 Nk) * 2 + k 层结点总数

3.3 通过遍历序列构造二叉树

（1）中序序列可以严格的区分左右子树

其他序列可以提供树的根信息

所以只有其他序列和中序序列的组合可以唯一确定树形

（2） 先序和后序

若先序序列为：a1,a2,a3,…,ak-2,ak-1,ak

后序序列为：e1,e2,e3,…,ek-2,ek-1,ek

a) a1=ek 为该树的根；

b) 若 a2=ek-1 则该结点为根下的唯一孩子结点但不知左右，以该结点为根继续向下分析；

c) 若 a2!=ek-1,则 a2 为左子树，ak-1 为右子树，并且可以划分出左右子树的元素范围，再分别以这两个为根向下分析；

可以看出：若二叉树没有度为 1 的结点则通过后序和先序也可以唯一的确定树形

特别：先序：a1,a2,a3,…,an

后序：an,an-1,…,a2,a1

则：层次遍历和先序相同，这样的二叉树共有 2^(n-1)，且都是单支树。

该树的结点的度为 1 或为 0，且 a1 为根，an 为叶子，并且在中序遍历中，a1 和 a2 或分布两侧或相邻在一边。

（3） 字母分析法

先序遍历：N(NLR)(NLR)　　中序遍历：(LNR)N(LNR)　　后序遍历：(LRN)(LRN)N

a)用此方法分析结点遍历顺序中的先后关系—>结点的辈分关系

b)若要 NLR=LNR （即先序遍历和中序遍历相同）则 L 为空，LRN=LNR（后序遍历和中序遍历相同）则 R 为空，若 NLR=LRN 则 LR 都为空，即只有根。

3.4 线索二叉树

(一)　线索二叉树查找线性关系前驱或后继结点

	查找前驱结点	查找后序结点
先序线索二叉树 （N L R）	无左孩子：直接查找 有左孩子：需要通过双亲结点	无右孩子：直接查找 有右孩子：有左孩子：左孩子结点 无左孩子：右孩子结点
中序线索二叉树 （L N R）	无左孩子：直接查找 有左孩子：左子树的最右下结点	无右孩子：直接查找 有右孩子：右子树的最左下的结点
后序线索二叉树 （L R N）	无左孩子：直接查找 有左孩子 有右孩子:右孩子结点 无右孩子:左孩子结点	无右孩子：直接查找 又右孩子：需要通过双亲结点

说明：

1) 需要通过双亲结点，是指仅通过二叉链表是无法完成查找的，需借助栈或三叉链表来完成查找
2) 先序线索二叉树查找先序后继结点时，左孩子优先右孩子；后序线索二叉树查找后序前驱结点时，右孩子优先左孩子

(二)　树和二叉树的转换

树	对应二叉树
叶结点	无左孩子的结点
非叶结点+根 （N 总-N 叶+1 或 N 非终端结点+1）	无右孩子的结点
无右兄弟的叶子结点	度为 0 的结点（叶） (=无左孩子+无右孩子）
有孩子，无右兄弟 无孩子，有右兄弟	度为 1 的结点
既有孩子，又有右兄弟	度为 2 的结点

说明：

1) 在树中，每一个非叶子结点，既度>=1 的结点都有一个无右兄弟的孩子结点，该结点转化为二叉树中的无右子树结点
2) 二叉树中的叶子结点一定来自树中的叶子结点
3) 孩子兄弟表示法和二叉链表法，同一种存储方式的不同解释将一棵树转换为二叉树

(三)　树，森林，二叉树

树	森林	二叉树
先根遍历	先序遍历	先序遍历
后根遍历	中序遍历	中序遍历

如果给出树的先根遍历与后根遍历可以唯一的构造出树所对应的二叉树，进而为一确定树。

3.5 平衡二叉树（AVL）

(一)　二叉树的操作

插入：

1) 二叉排序树是一种动态集合，该树是在查找过程中生成的
2) 插入的结点一定是叶结点

删除：删除后—>重新链接—>确保二叉树性质不会丢失

1) 若是叶结点，直接删除
2) 结点只有一棵左或右子树，替代
3) 结点有左右子树，有两种方式

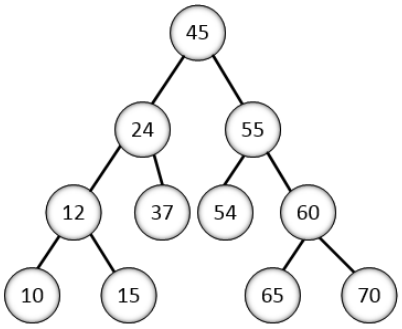
a) 孩子替代法（孩子的整棵树替代）：左孩子替代，被删结点的右子树连接到被删结点左子树的最右边；右孩子替代法，被删结点的左子树连接到被删结点右子树的最左边；
b) 线性关系替代（转化为删除叶结点或只有一棵子树的结点）：左子树的最右结点（叶结点或无右子树结点）；右子树的最左结点（叶结点或无左子树结点）

（二） 相关计算

深度为 h 的平衡二叉树	最少结点数	描述：所有非叶子结点平衡因子均为 1 或-1	n0=0,n1=1,n2=2 nh=nh-2 + nh-1 + 1
	最多结点数	描述：所有结点平衡因子均为 0	2^h-1

3.6 排序二叉树

判断二叉排序树查找路径是否合法



（1） 按照给定的路径画出二叉排序树，若该树无分支则路径正确，否则错误
（2） 快速判断：任意元素后面相邻的两个元素都大于或都小于该元素
（3） 快速构造二叉排序树，在给定插入元素序列中，一个结点的左孩子是其后第一个小于该结点值的结点，一个结点的右孩子是其后第一个大于该结点值的结点

例：查询序列为：45 55 60 70

45 后 55 60 都大于 45,55 后 60 70 都大于 55，所以该路径合法

单科书 156 页第 6 题：A：95 22 91 24 94 71

91 后 24 小于 91,94 大于 91，该序列不合法，其他的都合法

3.7 哈夫曼编码树

(1) 带权路径长度最小的二叉树称为哈夫曼树，最优二叉树

(2) N 个结点构造的最优二叉树，共有 2N-1 个结点，无度为 1 的结点

(3) 如果没有一个编码是另一个编码的前缀，则称这样的编码为前缀编码

(4) 0 和 1 表示左子树和右子树不是确定的所以哈夫曼树不唯一

第四章 查找

(一) 查找的 ASL

<table>
<tr><th>查找类型</th><th>存储结构</th><th>表是否有序</th><th colspan="2">ASL</th></tr>
<tr><td rowspan="4">顺序查找</td><td rowspan="4">顺序存储

链式存储</td><td rowspan="4">有序表

无序表</td><td rowspan="2">无序表</td><td>$ASL_{成功}=\frac{n+1}{2}$</td></tr>
<tr><td>$ASL_{失败}=n+1$</td></tr>
<tr><td rowspan="2">有序表</td><td>$ASL_{成功}=\frac{n+1}{2}$</td></tr>
<tr><td>$ASL_{失败}=\frac{n}{2}+\frac{n}{n+1}$</td></tr>
<tr><td>折半查找</td><td>顺序存储</td><td>有序表</td><td colspan="2">$ASL_{成功}=\frac{n+1}{n}\log_2(n+1)-1$</td></tr>
<tr><td colspan="2">$ASL_{失败}=\log_2(n+1)-1$</td></tr>
<tr><td>分块查找</td><td>顺序存储

链式存储</td><td>索引表有序

查找表有序
↓
无序</td><td colspan="2">都采用顺序查找：
$ASL_{成功}=\frac{b+1}{2}+\frac{S+1}{2}$

索引表采用折半查找：</td></tr>
</table>

			$ASL_{成功}=\log_2(b+1)+\frac{S+1}{2}$ 若查找表有序，且采用折半查找： $ASL_{成功}=\log_2(b+1)+\log_2(S+1)$

注意：

(1) 折半查找计算 ASL 时要画出判定树，而且要画出失败结点，这样通过判定树就可以计算 ASL 了

(2) 分块查找有两个最优问题：1.s=根号 n 时平均查找长度达到最小值；2.查找表有序时，索引表和查找表均采用折半查找，实现查找最优

(二)B 树

B 树一共要解决三个问题：1：定义问题 2：操作问题 3：计算问题

一、 定义问题

关于 B 树的定义一共是 5 条，你可以直接记住，也不是很难理解，但我是这样记的：按照结点

根结点	（若不是终端结点）至少 2 棵子树，最多 m 棵子树	每一结点内： 1. 关键字升序排列 2. 子树数=关键字数+1	总体： 叶结点数=关键字总数+1
非根非叶结点	至少$\lceil \frac{m}{2} \rceil$棵，至多 m 棵		
叶结点	出现在同一层的虚拟结点		

二、 操作问题

（一）插入操作

定义：关键字数=m-1 为临界状态

关键字数<m-1 为安全状态

操作：1.插入在最底层的某个非叶结点内

2.（1）插入数据前该结点为安全状态，直接插入新数据

（2）插入数据前该结点为临界状态，插入后分裂，分裂处结点左右指针分别指向左右关键字，下指指针不变

注：若分裂过程传到根结点，树的高度+1

（二）删除操作

安全状态：关键字数 > $\lceil \frac{m}{2} \rceil$ -1
临界状态：关键字数 = $\lceil \frac{m}{2} \rceil$ -1
倒数第二层：终端层

（1） 删除非终端层结点

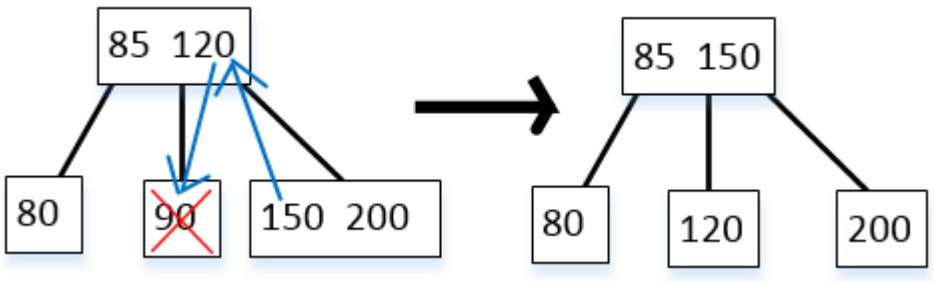
Case1:：该结点的子树节点中有安全状态，则选择前驱或后继结点替代，转化为删除终端层结点；

Case2：子树皆为临界状态，合并子树结点

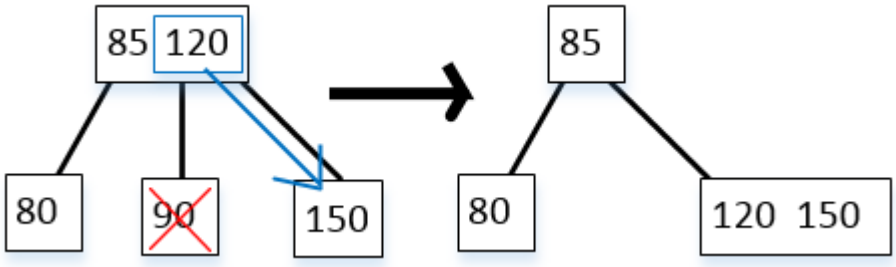
（2） 删除终端层结点

Case1：该结点在删除前为安全状态，则直接连右指针直接删除；

Case2：该结点为临界状态，左右兄弟有安全状态，则轮换替代



Case3：该结点为临界状态，左右兄弟皆为临界状态，则借父归并



注意：在做复杂的操作时要一步一步的通过基本操作来完成

三、 计算问题

注意：你要看清题，n 是代表结点还是关键字数

（1） n 个关键字

n个关键字 ⟶ 树的高度（不包括非叶结点） ⟶ 结点数（非叶结点）

$\log_m(n+1) \leqslant h \leqslant \log_{\lceil \frac{m}{2} \rceil}(\frac{n+1}{2})+1$

$\frac{m^h-1}{m-1} \leqslant$结点数$\leqslant \frac{2(\lceil \frac{m}{2} \rceil^{h-1}-1)}{\lceil \frac{m}{2} \rceil-1}+1$

（2） n 个结点

n个结点（非叶结点）⟶ 关键字数

$(m-1)(\lceil \frac{m}{2} \rceil-1)+1 \leqslant$关键字数$\leqslant n(m-1)$

(三)散列表

散列函数：除留余数法（%，mod）

H(key)=key%p

p 是不大于 m 最接近 m 的质数

α = n/m n 是元素个数，m 是表长

（1） 开放定址法求 ASL

$$ASL_{成功}=\frac{1}{n}\sum 各元素成功次数$$

$$ASL_{失败}=\frac{1}{p}\sum 各余数失败次数$$

做题时最好画出这样的表格

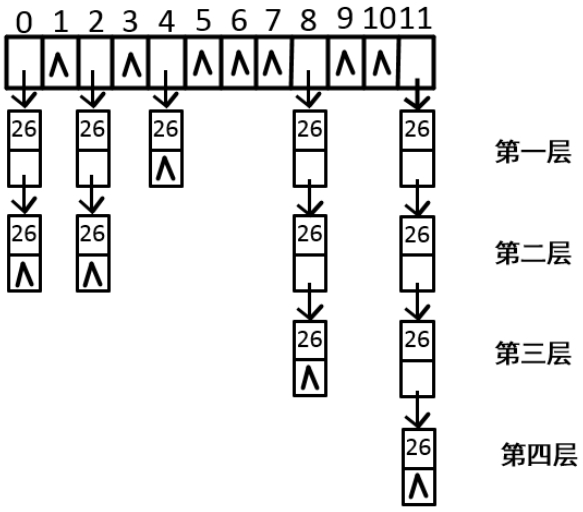
地址	0	1	2	3	4	……	n
关键字							
成功次数							
失败次数							

（2） 拉链法

$$ASL_{成功}=\frac{1}{n}\sum 层\times 每层结点数$$

$$ASL_{失败}=\frac{1}{p}\sum 链表结点个数$$

例如下图



ASL 成功=（1*5+2*4+3*2+4*1）/12=23/12

ASL 失败=(2+2+1+3+4)/12 =12/12

(四)B+树

“+”

（1） 每个结点加了一个关键字与子树数相等

（2） 叶结点加了关键字，包括了全部的关键字

（3） 叶结点加了子树指针，指向记录

（4） 叶结点加了水平指针

B+树要是直接考概念，那就是送分了，但也可能变形式考

我想了几道题大家思考一下，给大家一个方向

Q1：在 B 树中关键字的出现次数，B+树中关键字的出现次数？

Q2：在 B 树和 B+树中查找某个关键字各有几条路径？

Q3：B 树与 B+树的叶结点各有几个指针，有什么不同？

第五章 图

(一) 定义，重点

1） 定义：非空点集 +依赖于点的可空边集。

2） 重点：无向图：生成树；连通性；边，点，度计算

有向图：路径；距离

(二) 生成树

连通图——生成树非连通图——生成森林

生成树是一棵树，所以它拥有树的性质，这里用的最多的是：n 个顶点的图的生成树有 n-1 条边

根据上面的重要性质有以下两个结论：

(1)一个图有 n 个顶点，小于 n-1 条边，必为非连通图

(2)一个图有 n 个顶点，大于 n-1 条边，必有环

(三) 连通性

无向图保证连通性与保证非连通性问题

保证：在任何情况下都是

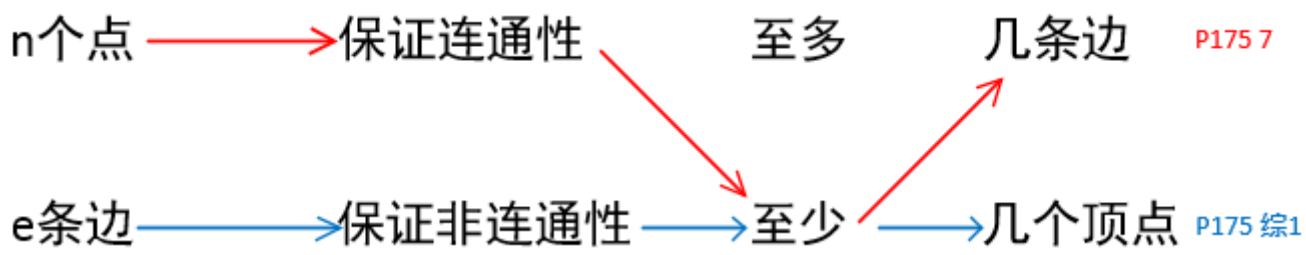
记住：一个公式：$e=\frac{n(n-1)}{2}$

两个模型：(1)生成树模型

(2)最简环模型 ：n 个顶点 n 条边组成的环

以下几个关键字的顺序组合都是一道题，自己写出来做一下这就掌握了

例如：红线的组合就是 P175 的第 7 题 蓝线的组合就是综合题 1，自己把其他的总结出来

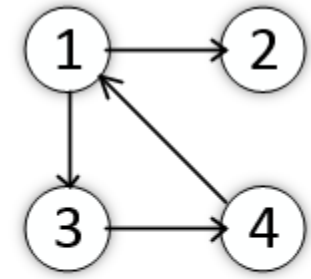


(四) 边，点，度计算

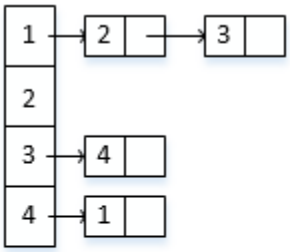
	n个顶点边的最大数	连通问题	度，边关系	度，点关系	边，点关系
无向图	$e=\frac{n(n-1)}{2}$	连通图(很重要) 连通分量极大连通子图 生成树：包含全部顶点的极小连通子图	$\sum_{i=1}^{n} TD(vi)=2e$	$0 \leqslant \overset{每个顶点的度}{} \leqslant n-1$	$n-1 \leqslant e \leqslant \frac{n(n-1)}{2}$ 生成树　　完全图
有向图	$e=n(n-1)$	强连通图 强连通分量	$\sum_{i=1}^{n} ID(vi)=\sum_{i=1}^{n} OD(vi)=e$	$0 \leqslant \overset{每个顶点的度}{} \leqslant 2(n-1)$ $0 \leqslant \overset{出度或入度}{} \leqslant (n-1)$	

（五） 图的存储

1） 邻接矩阵，邻接表



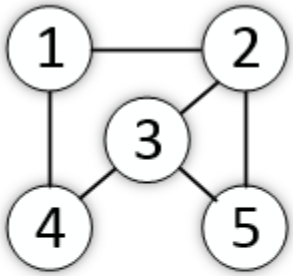
$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$



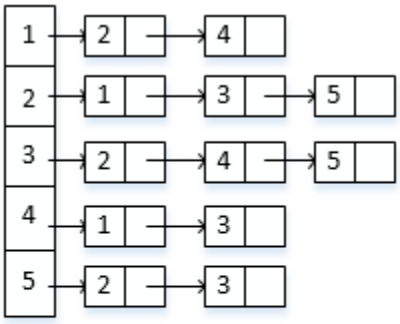
n 个顶点
e 条边

n^2 - e 个 0
e 个 1

边表结点 e个
最多n(n-1)



$$\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}$$



n 个顶点
e 条边

n^2 - 2e 个 0
2e 个 1

边表结点 2e个
最多n(n-1)

	邻接矩阵	邻接表
存储结构	顺序存储	链式存储
创建时间复杂度	O(n^2+ne)	O(n+e)
空间复杂度	O(n^2)	无向图:O(n+2e) 有向图:O(n+e)

2） 十字链表，临接多重表

十字链表：有向图的链式存储结构（邻接表和逆邻接表的融合）

a） 共 n 个顶点结点，e 个弧结点

b） 共 2n+2e 个指针域，其中有效指针为 2e，空指针为 2n

c) 在十字链表中即容易找到以 vi 为尾的弧，也容易找到以 vi 为头的弧，因而容易求得顶点的出度和入度

临接多重表：是无向图的链接存储结构

a) 邻接表在存储无向图时，每条边都要建立两个边表结点
b) 每条边仅用一个边表结点

(六) 图的遍历

	空间复杂度	时间复杂度
BFS 算法	O(n)	邻接表：O(n+e) 邻接矩阵：O(n^2)
DFS 算法	O(n)	邻接表：O(n+e) 邻接矩阵：O(n^2)

说明：

1) BFS 生成树（生成森林）DFS 生成树（生成森林）
   图的邻接矩阵不唯一>DFS 和 BFS 唯一
   图的邻接表不唯一>DFS 和 BFS 不唯一
2) 图的遍历和图的连通性

无向图	若连通：一次遍历就能访问图中所有顶点 非连通：一次遍历仅能访问该结点所在连通分量的所有顶点
有向图	强连通(或初始结点到其它结点有路径)：一次遍历即能访问所有结点 非强连通：一次遍历无法完成所有结点的访问

3) 对于无向图，BFS 生成树，起点到其它顶点的路径是图中对应的最短路径也即是所有生成树中树高最小的
4) 判断无向图 G 是一棵树的条件
   a) 无回路连通图
   b) 有 n-1 条边的连通图

(七) 图的应用

一、 MST

1. 存在相等权值的边—>MST 树形可能不唯一

   各边权值各不相等—>MST 树形唯一

   说明：所有权值均不相等，或者有相等的边，但是在构造最小生成树的过程中权值相等的边都被并入生成树的图，其最小生成树唯一

2. MST 性质：必存在一棵包含最小两栖边的最小生成树

   (1) Prim 算法（通过点集判定边）——选择与当前点集权值最小的点加入该点集

   时间复杂度 O(n^2)，因不依赖于边，所以适用于求边稠密的图的最小生成树

   (2) Kruskal 算法（通过边集判定点）——按权值的递增次序选择合适的边表来构造最小生成树

   时间复杂度 O(eloge)，因不依赖于点，适合于边稀疏而顶点较多的图

二、 最短路径

1. 带权路径长度最短的那条路径称作最短路径
2. 算法性质:两点之间的最短路径也包含路径上其他顶点间的最短路径

   (1) Dijkstra 单源最短路径问题

   时间复杂度 O(n^2)

   边上负带权值，Dijsktra 算法并不适合，原因：已求得最短路径的顶点集，归入 S 内的最短路径不再变更

   (2) Floyd 各顶点之间最短路径

   A(k)[i][j]是从顶点 Vi 到 Vj 中间顶点的序号不大于 k 的最短路径长度

   时间复杂度 O(n^3)

   Floyd 算法允许图中带有负权值的边，但不允许有包含带负权值的边组成回路

三、 拓扑排序(AOV 网)

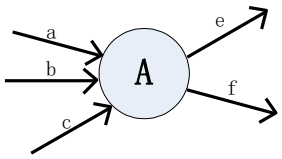
AOV 网是一种以顶点表示活动，以边表示活动的先后次序且没有回路的有向图

注意：

1) 拓扑排序序列可能不唯一，存在同等地位的入度为 0 的顶点
2) DFS 算法的顶点退栈序列是逆向的拓扑有序序列
3) 邻接矩阵是三角矩阵——>存在拓扑序列，反向不成立

四、 关键路径(AOE 网)

顶点表示事件，有向边表示活动，边上的权值表示完成活动的开销



a) A 发生后，d,e 才能开始

b) a,b,c 都结束时，A 才能发生

注：

1) 关键路径上的所有活动都是关键活动，可通过加快关键活动来缩短整个工程的工期，但也不能任意缩短关键活动，因为一旦缩短到一定的程度，关键活动可能变成非关键路径。

2) 网中关键路径不唯一，对于有几条关键路径的网，只有加快那些包括在所有关键路径上的关键活动才能达到缩短工期的目的。

补：数组的压缩存储

(一) 数组的存储结构

多维数组映射方式：按行优先，按列优先

$$LOC(a_{i,j})=LOC(a_{i,j})+[i*(h_2+1)+j]*L$$

从0开始已存i行　h2+1个元素　当前元素同行已存元素j个

h2 为列表上限

(二) 矩阵的压缩存储

对称矩阵

第一步：确定正序递增 OR 逆序递减 —> 确定形式

第二步：行 OR 列为主增量 —> 确定关系式

	行为主增量	列为主增量
正序递增	$K=\begin{cases}\frac{i(i-1)}{2}+j-1 & i>j\\ \frac{j(j-1)}{2}+i-1 & i<j\\ \left(\frac{n(n+1)}{2}\right. & i<j)\end{cases}$	i 与 j 互换
逆序递减	$K=\begin{cases}\frac{(i-1)(2n-i+2)}{2}+(j-i) & i<j\\ \frac{n(n+1)}{2} & i>j\end{cases}$	i 与 j 互换