数据结构预习指南

最后一个NPE

# C:\Users\chenj\Documents\Tencent Files\715711877\FileRecv\MobileFile\IMG_20181209_233337.jpg第一章

数据的基本结构

**数据结构的概念：**

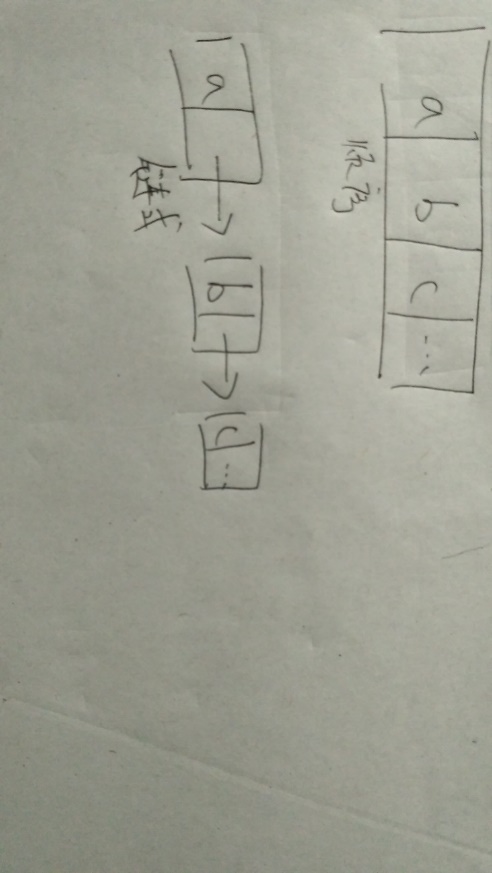
相互之间存在一种或多种特定关系的数据元素的集合。

**术语：**

数据：能输入到计算机中并被计算机程序处理的符号总称。

数据元素：数据的基本单位，一个数据元素由若干个数据项组成。

数据对象：性质相同的数据元素的集合。

 基本结构：

数据的存储结构

集合，线性结构，树形结构，图状结构（网状结构）

**数据的存储结构：**

又称物理结构。指的是数据结构在计算机中的表示。

有两种：顺序存储结构和链式存储结构。

**抽象数据类型ADT(abstract data type)**：指一个数学模型以及定义在该模型上的一组操作。D是数据对象，S是D上的关系集，P是对D的基本操作集。

**算法**：

特性：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，效率与低存储量要求

时间复杂度：指的是随问题规模n的增大，算法执行时间的增长率。

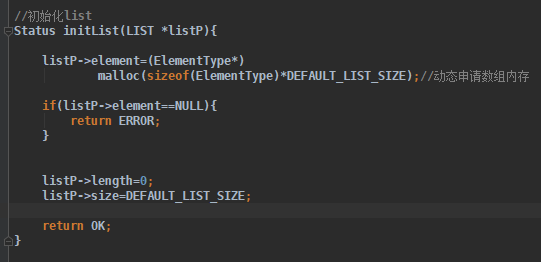
空间复杂度：算法所需存储空间的量度。

## **第二章**：

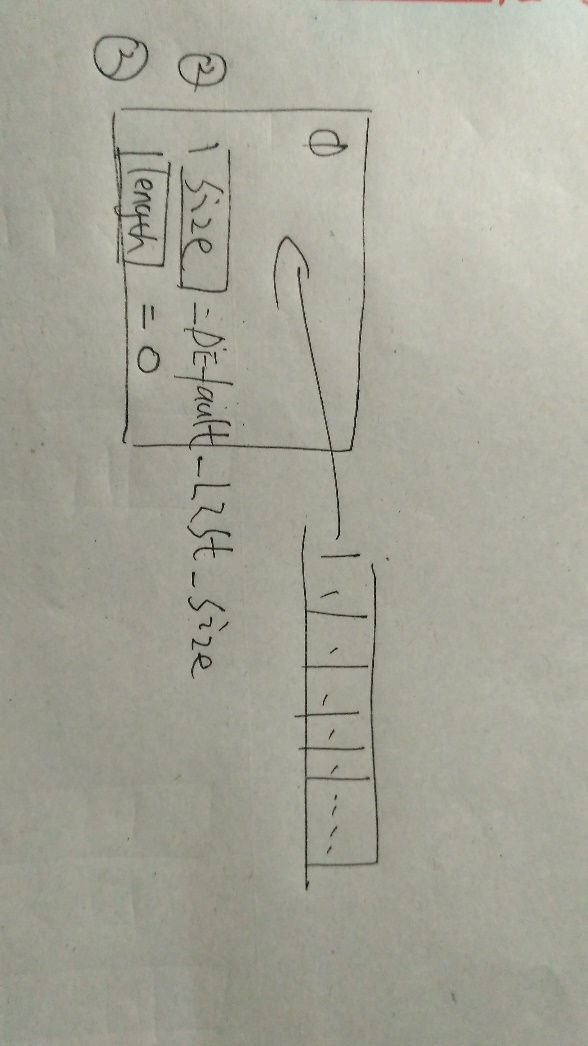
**线性表概念：**是n个数据元素的有限序列。可以看做数组。

**线性表的操作：**

初始化:



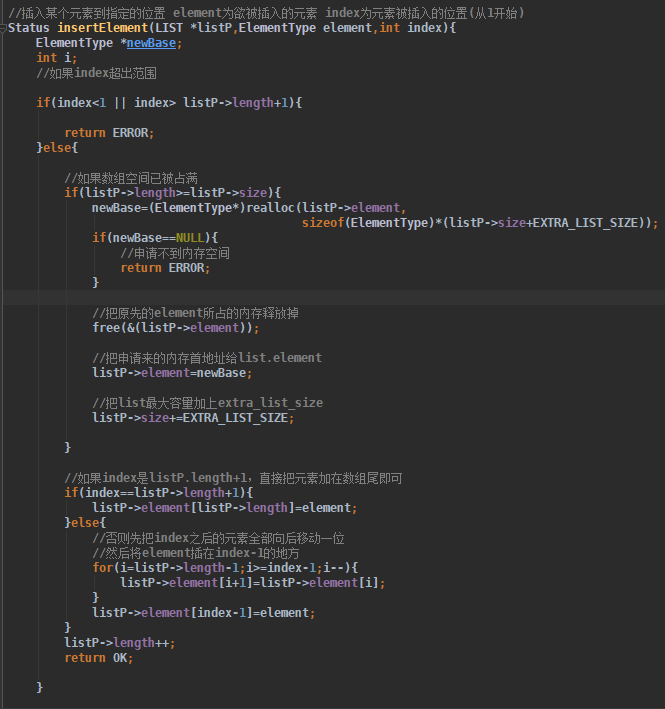
顺序表初始化代码

 说明：传入一个LIST类型的结构体指针，申请一块内存给该结构体指针的一个叫element的成员， 如果申请后的那块内存赋值给element为NULL，代表申请内存失败，返回失败，否则置该结构体的length成员为0，size成员为宏定义的默认表长度。

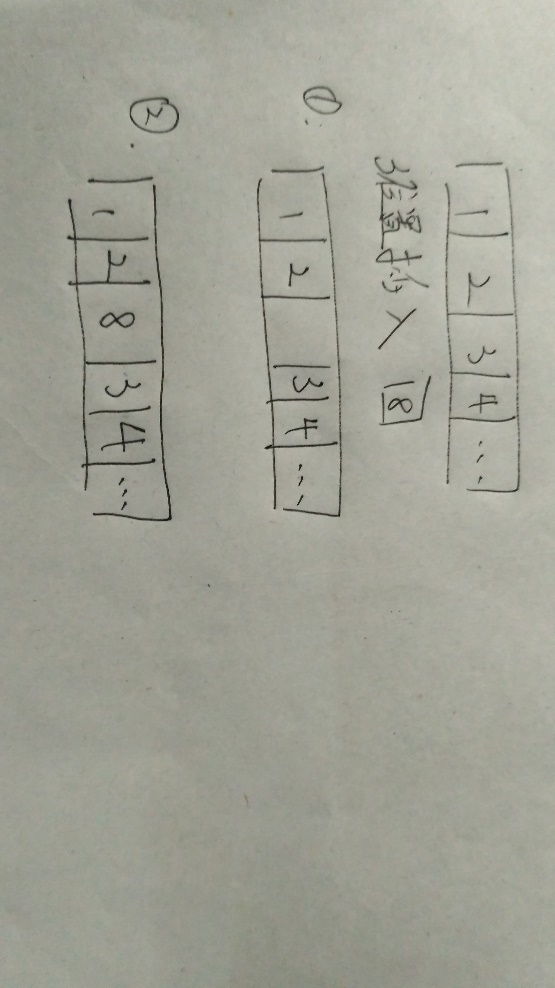
初始化图示

插入元素：

说明：首先判断传入的index参数是否超出范围（范围∈（1，length）），如果超过，返回失败，否则进入下一步：判断数组是否满了，如果满了，就申请一块新的内存空间重新赋值给element成员，并且把线性表的size加上增量。接下来，判断如果欲插入的位置在表尾，则直接插入，否则就把index后的元素全部后移，然后把元素插在index-1的位置。



顺序表插入元素代码



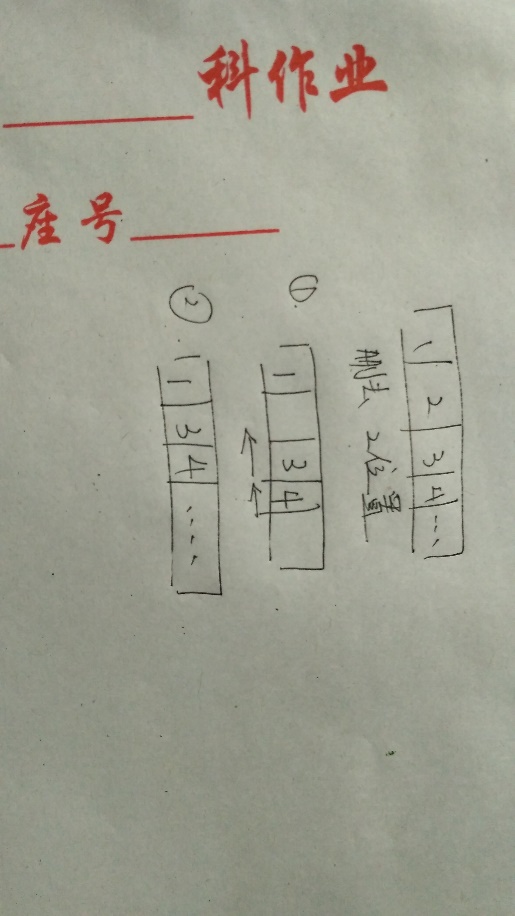
插入元素图示

删除元素：



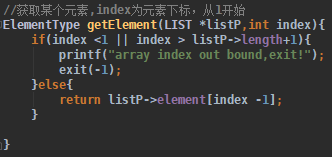
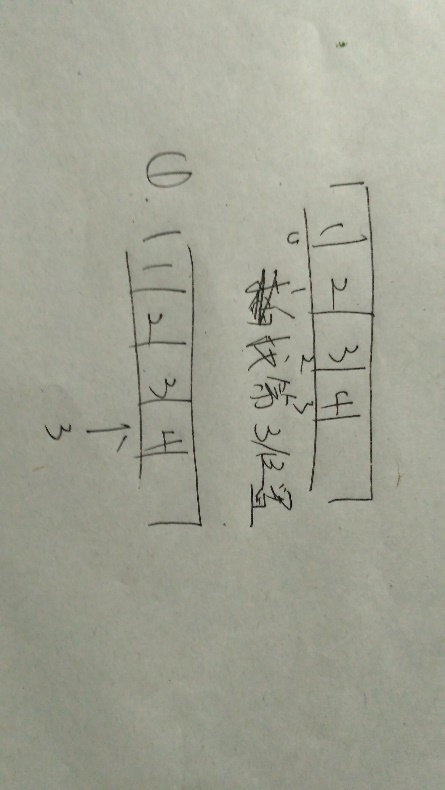
顺序表删除元素代码

说明：第一步先判断index是否超出范围。第二步判断插入的位置是否在表尾，如果是，什么也不做，否则将index之后的元素左移一位。最后将list的length成员-1；



删除元素图示

查询元素：



顺序表查询元素

说明：第一步判断index是否超出范围，否则像访问数组那样返回元素。

 线性表结构：

查询图示



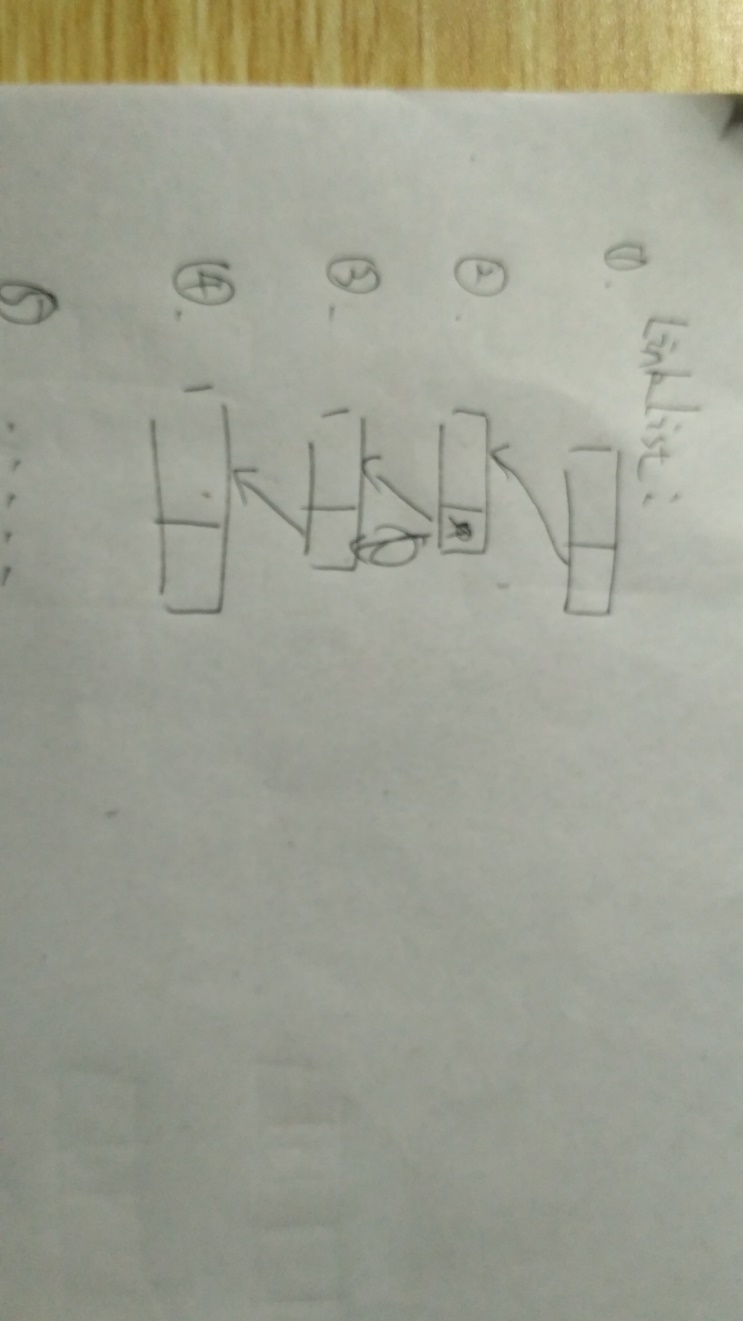
线性表结构图示

线性表结构

**链表的操作及其表示：**

初始化：

说明：循环length次，每次循环生成一个节点，如果这个节点指针为NULL，返回失败，否则将该节点的next成员置为NULL，并把这个节点赋值给链表指针的next成员。

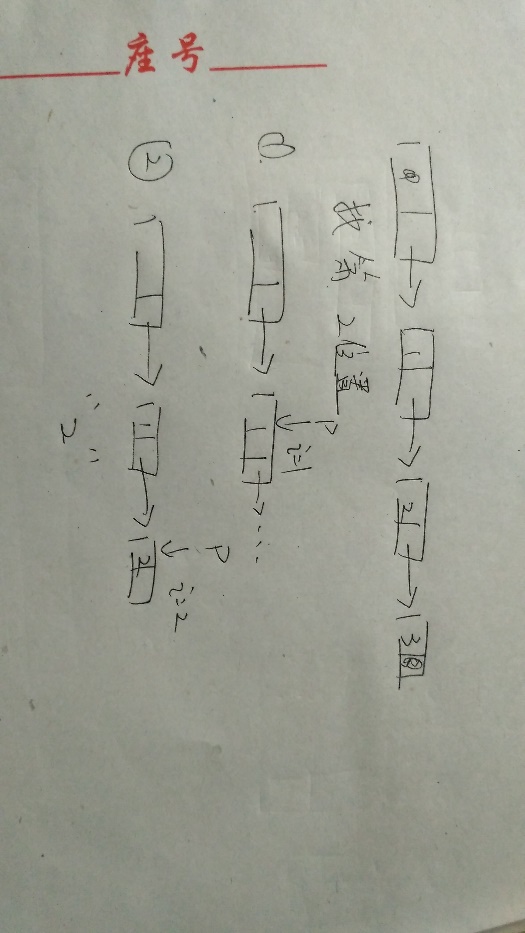
链表初始化代码

链表初始化图示

查询元素：

链表查询元素

说明：定义一个节点指针，将链表的next成员赋值给该指针，进入一个死循环，，每次循环：，当计数器等于index时，返回元素，否则计数器+1，节点指针指向节点指针的next成员。



链表查询元素图示



链表插入元素代码

插入元素：

说明：像查找元素那样，找到index所对应的那个节点指针，申请一个新的节点，放入欲被插入的元素，然后将新节点指针的next指向节点指针的next，节点指针的next置为新节点指针。

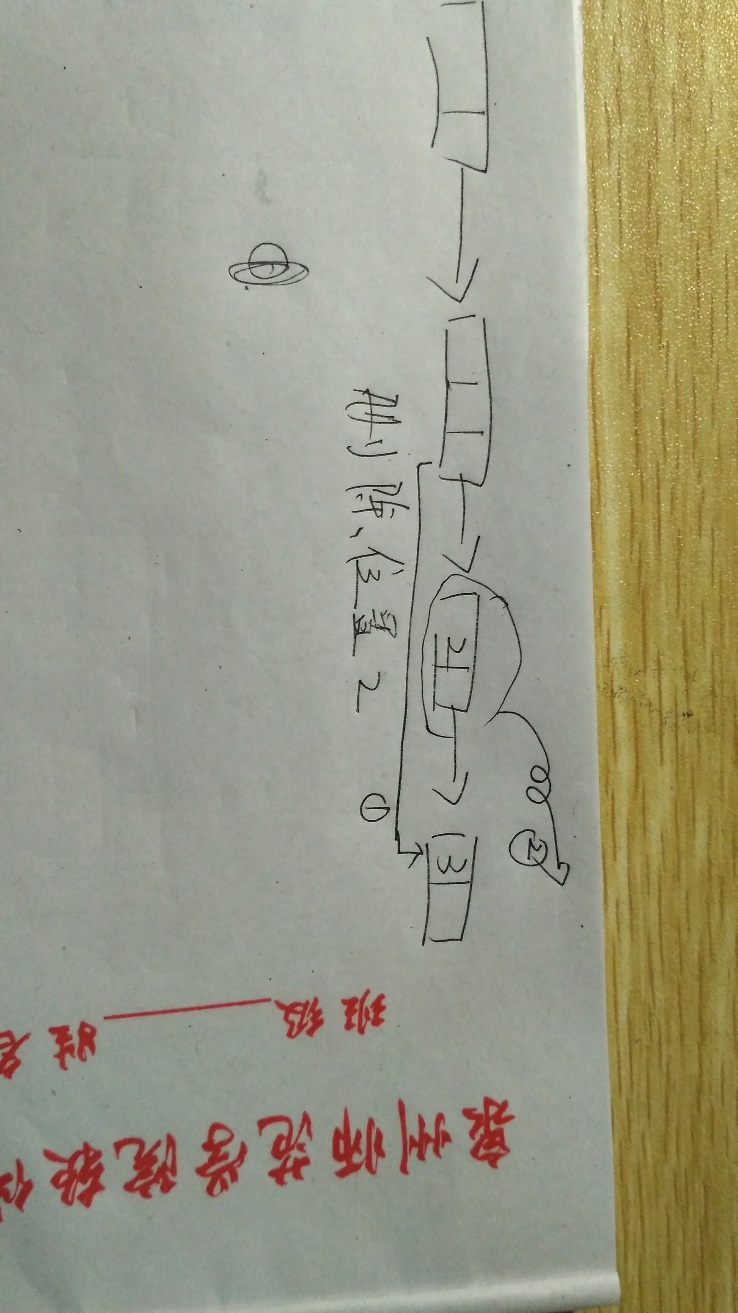


链表插入元素图示

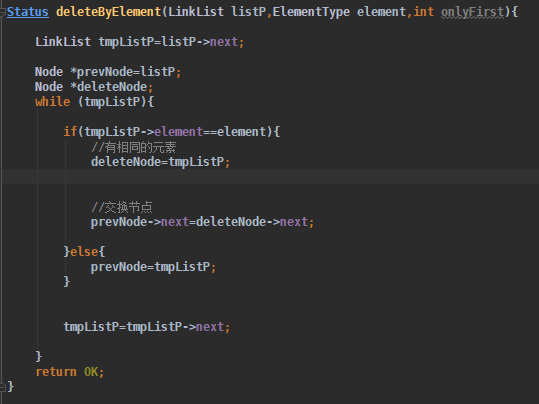
根据下标删除元素：

根据下标删除元素代码

说明：定义一个前驱节点，将链表的头结点赋值给前驱节点，进入一个死循环，每循环将链表当前的头结点赋值给前驱节点，然后把链表指向链表的next，当计数器i递增为index时，将链表的头结点赋值给欲被删除的节点，然后把前驱节点的next指向欲被删除节点的next，然后取出欲被删除节点的元素，最后释放欲被删除节点的空间。



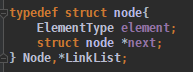
根据下标删除元素图示

根据元素删除元素：

根据元素删除元素代码

说明：同上，只是把判断条件从计数器的值改成判断元素。

线性链表的结构：



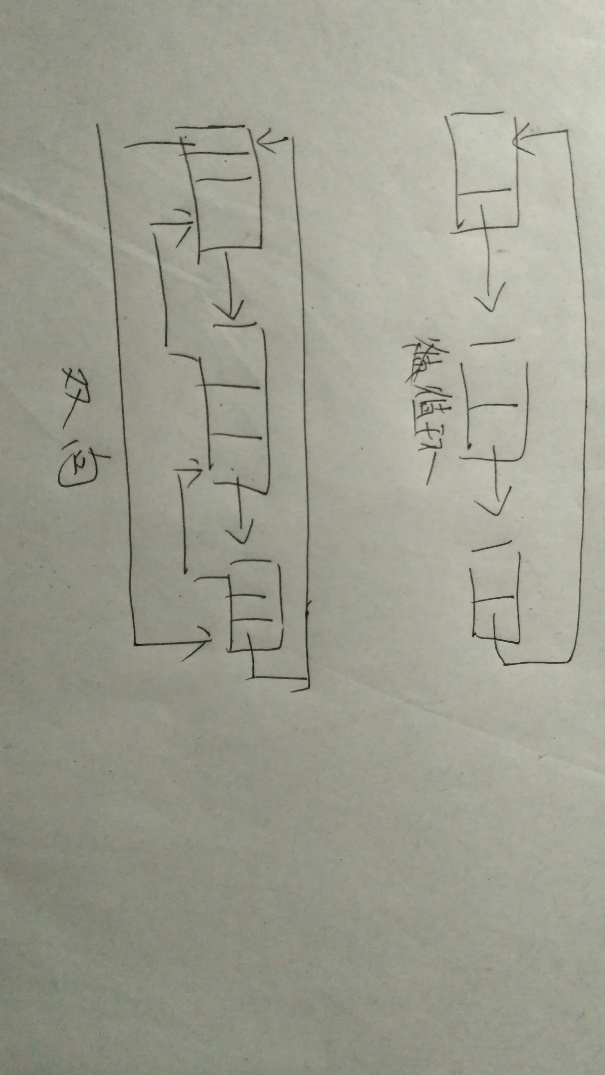
链表的结构



链表的结构图示

线性链表又称单链表，原因：每个节点只包含一个指针域。

**循环链表与双向链表：**



顺序表与线性表在某些操作方面的时间复杂度：

由**于顺序表**底层采用的数组，所以在**访问上**是任意（或者叫随机）访问，时间复杂度为O(1)。但是在**插入和删除**上，需要移动修改位置后的所有元素，所以时间复杂度为O(n)

**线性表**则相反，由于采用的是节点连节点结构，所以**删除和修改**很快，只要修改被修改节点的指针域与上一节点的指针域即可，时间复杂度为O(1)，在任意访问上，需要一个一个节点访问，所以时间复杂度为O（n）。

**这一条只是个人经验，结论可能是错的！！！**

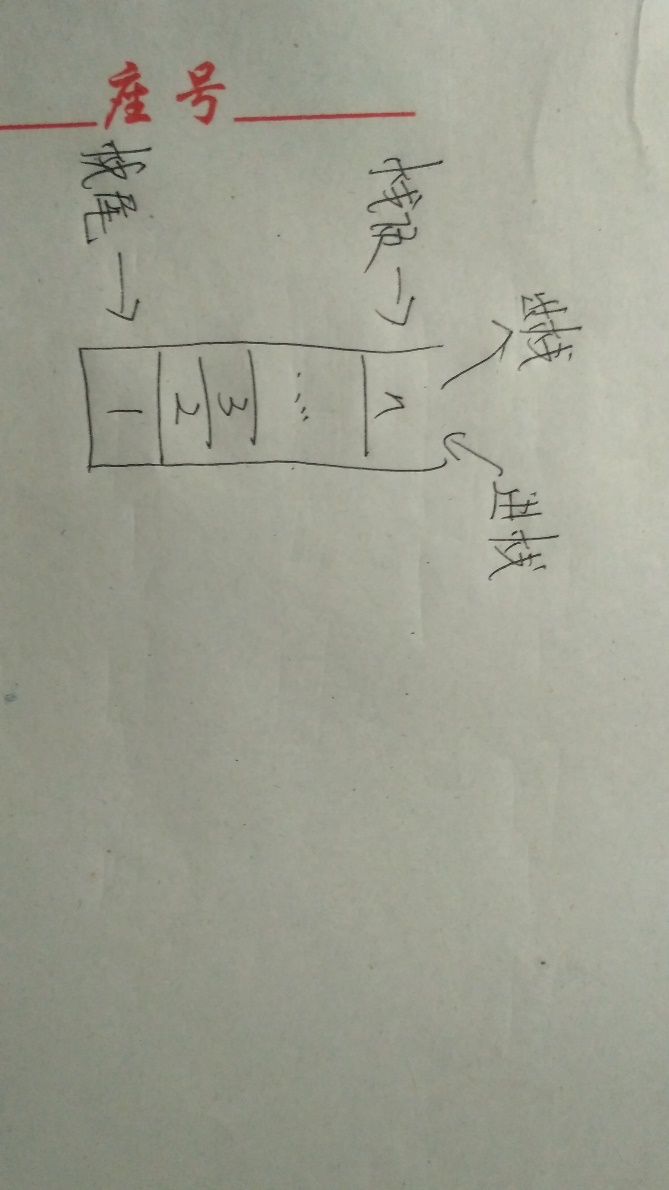
线性表的应用：

一元多项式的表示及相加。

# 第三章

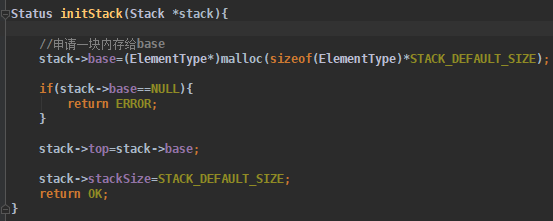
**栈的概念：**

是限定仅在表尾进行插入或删除操作的线性表。所以栈又称为后进先出（LIFO）的线性表，简称LIFO结构。



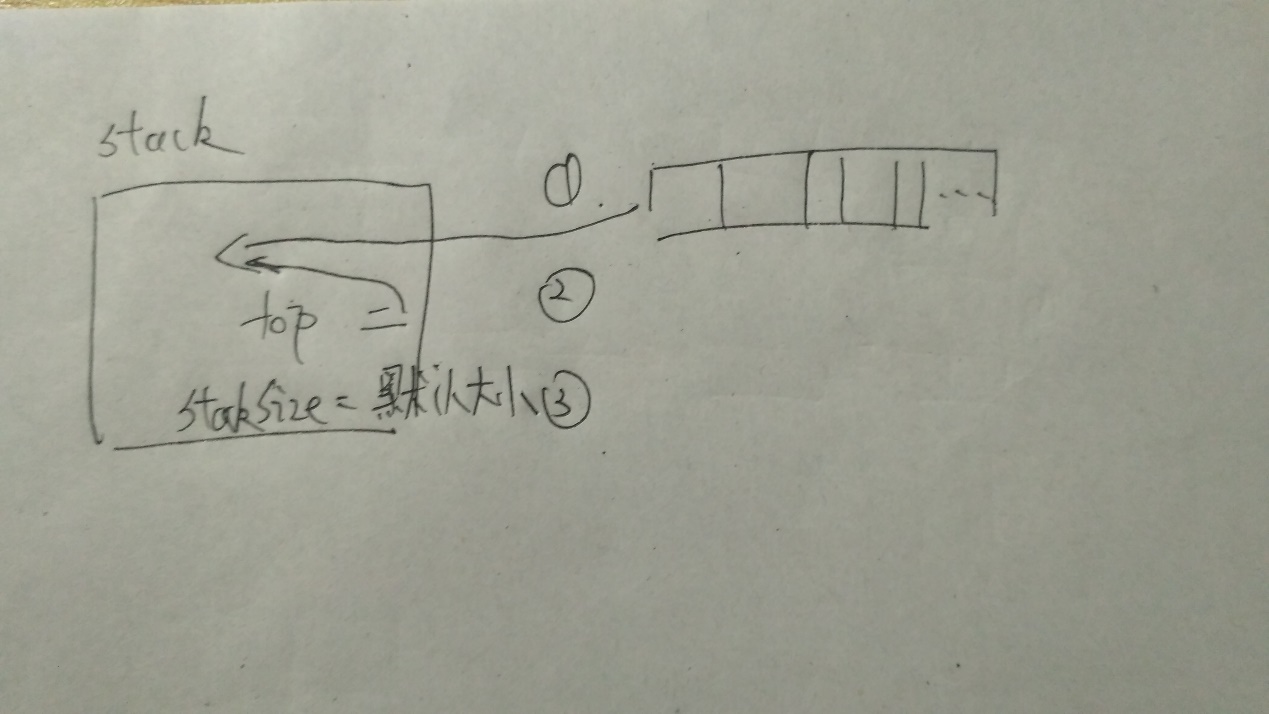
栈的结构

**顺序栈的表示及操作**：

 初始化：

顺序栈初始化代码

说明：首先申请一块新的内存赋值给stack结构体的base成员，并让该结构体的top指针指向base，并将stackSize成员置为栈的默认尺寸。

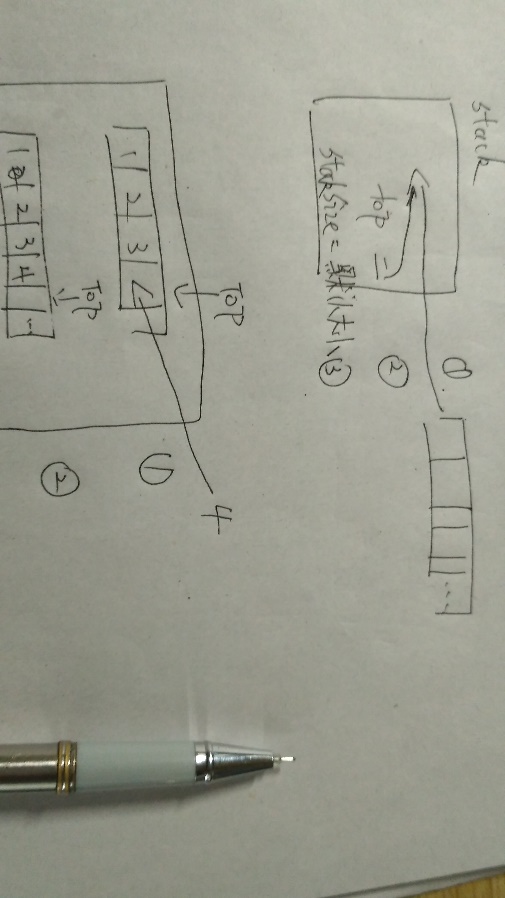


顺序栈初始化图示

入栈：

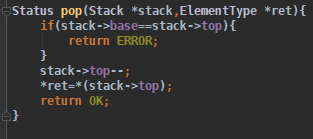


顺序栈入栈代码

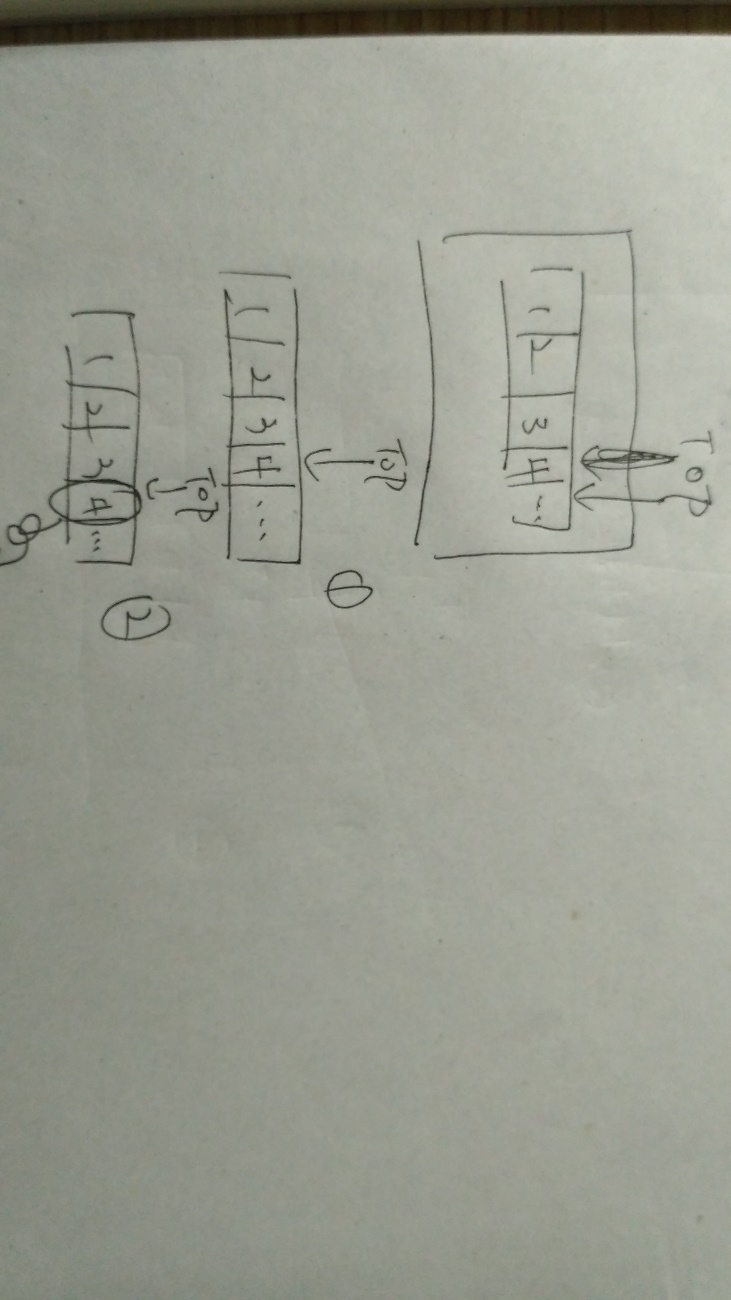


顺序栈入栈图示

说明：第一步先判断栈是否满了（也就是top与base两个指针相减的结果是否大于等于栈尺寸），如果是，像初始化栈那样，申请一块新的内存空间给base。接下来将欲被入栈的元素放入top指针指向的那块内存区域，最后top后移一位。



出栈：

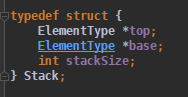


顺序栈出栈图示

说明：

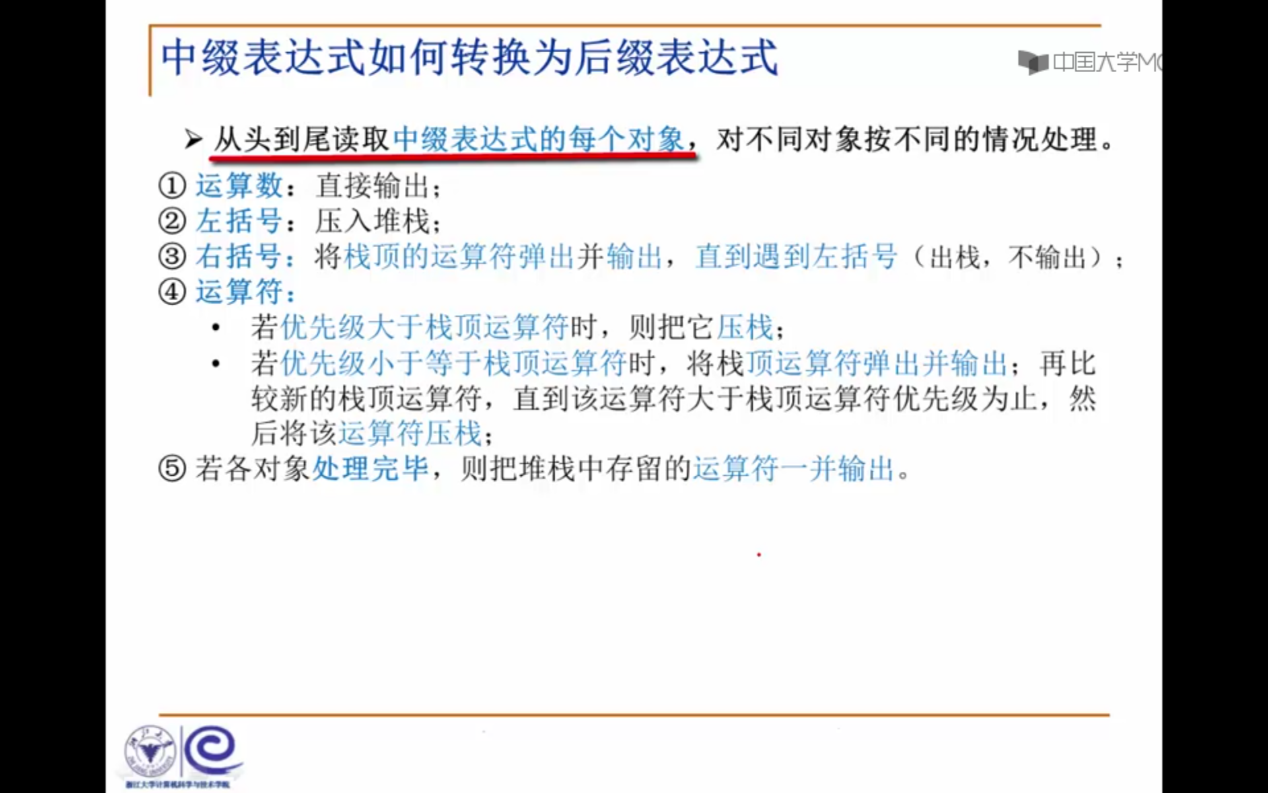
第一步先判断base是否等于top（空栈） ，否则top指针往前移一位，取出元素。

顺序栈出栈代码

 栈的结构：

栈的应用：

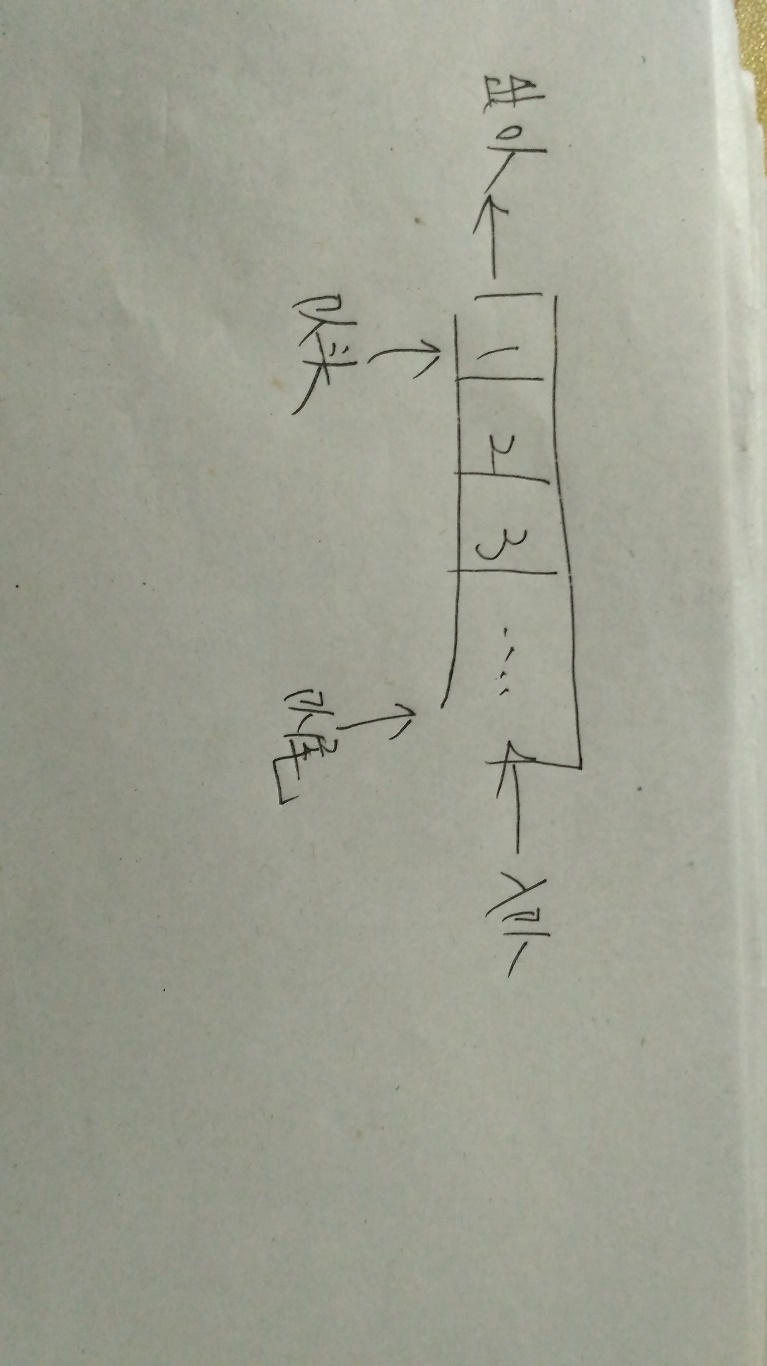
进制转换，表达式求值，括号匹配，行编辑，迷宫求解。



中缀表达式转后缀表达式

**队列**：

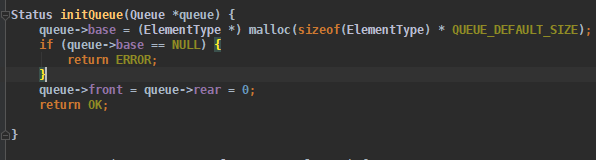
概念：先进先出（FIFO）的线性表，只允许在表的一段进行插入，另一端删除。



队列的表示

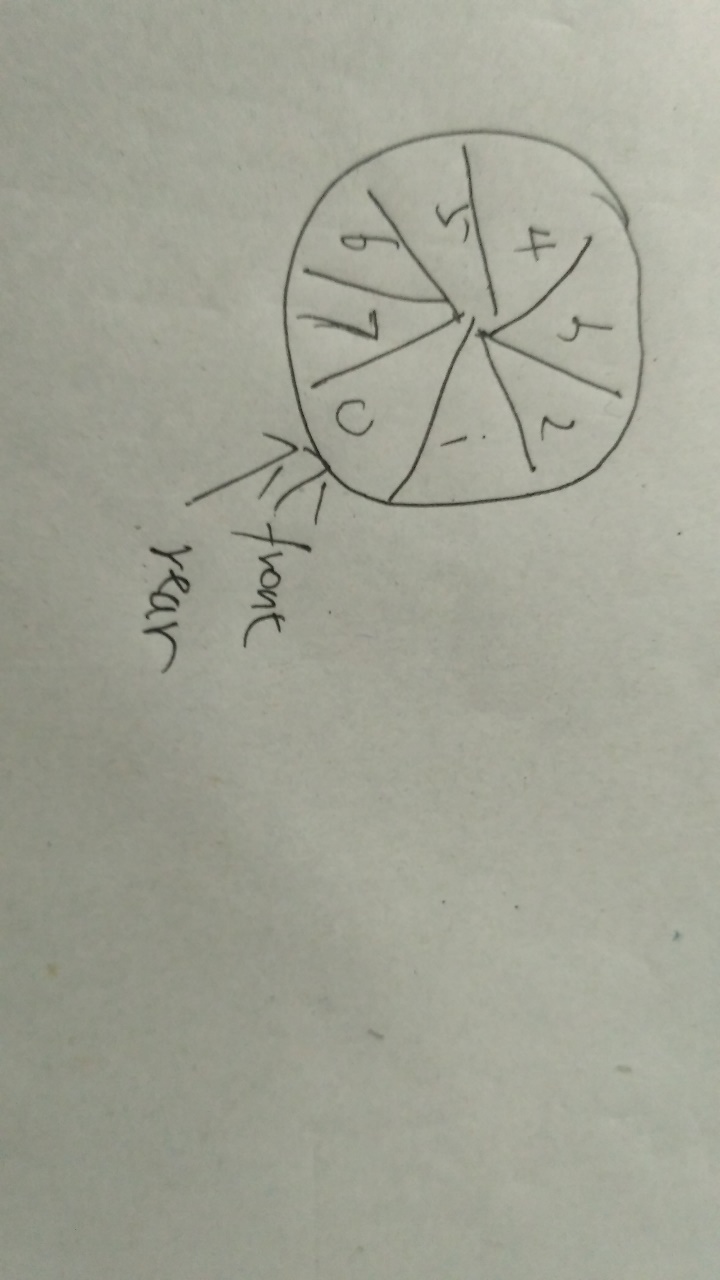
**循环队列**：

初始化：



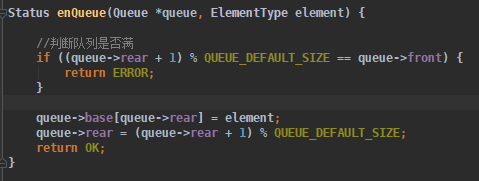
循环队列初始化代码

说明：跟线性表一样，申请一块空间给base，然后把front和rear都赋值为0；

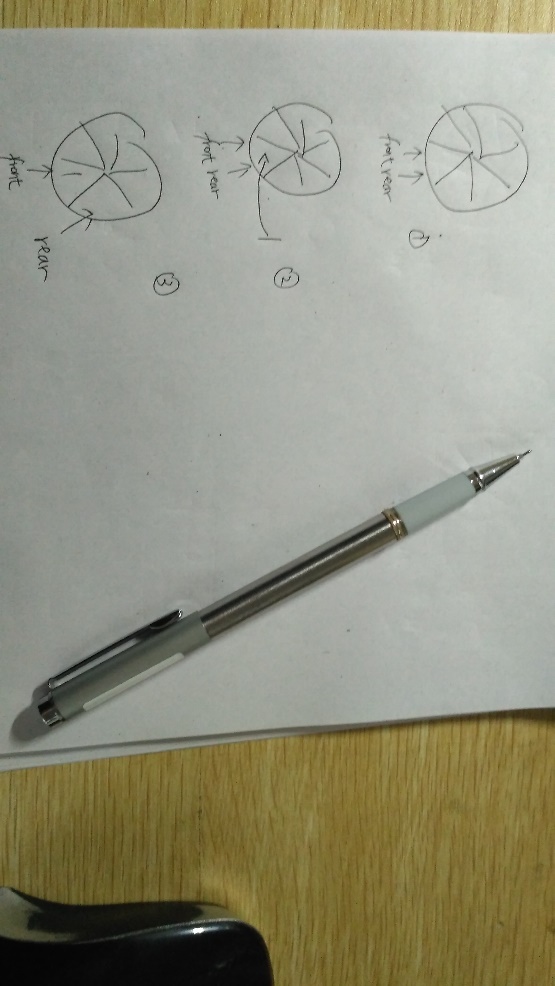


循环队列初始化

入队：

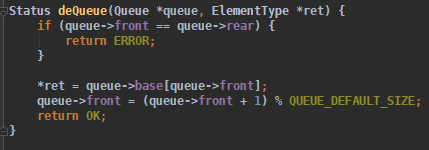


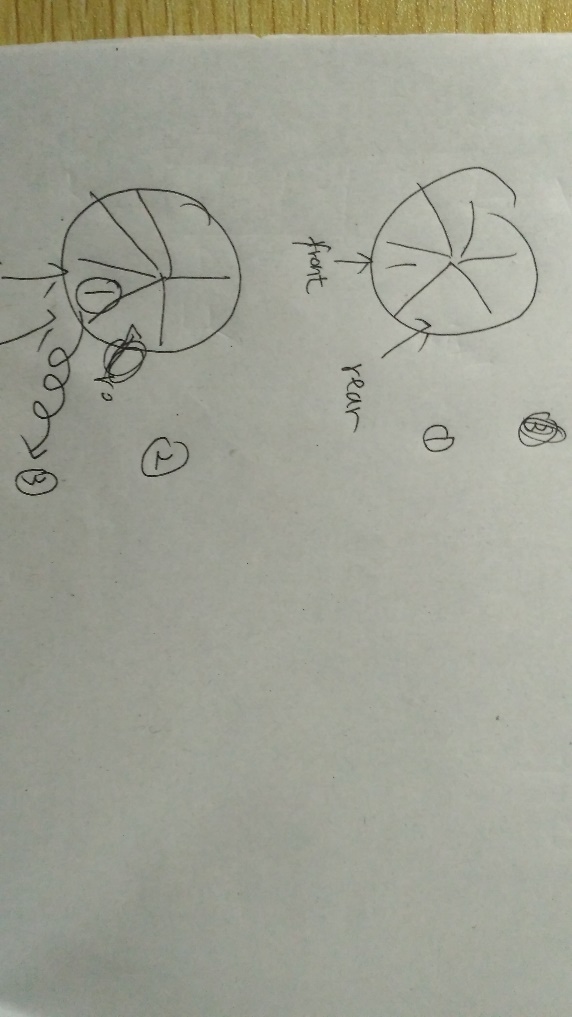
循环队列入队代码

 说明：判断队列是否满的标志：队尾所处的位置+1模上队列尺寸是否等于队尾所处的位置。每次插入元素都将元素放在rear下标处的位置，并且每次队尾的下标等于队尾所处的位置+1模上队列尺寸。

循环队列入队图示

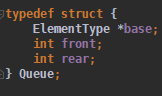
出队：

 说明：第一步判断front与rear是否相等，否则取出front下标处的元素，然后front=当前front加1模上队列尺寸。



循环队列出队代码

循环队列出队图示

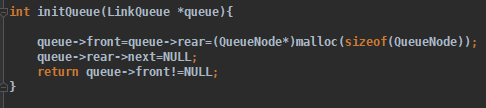


循环队列的结构：

**链队的表示及实现：**

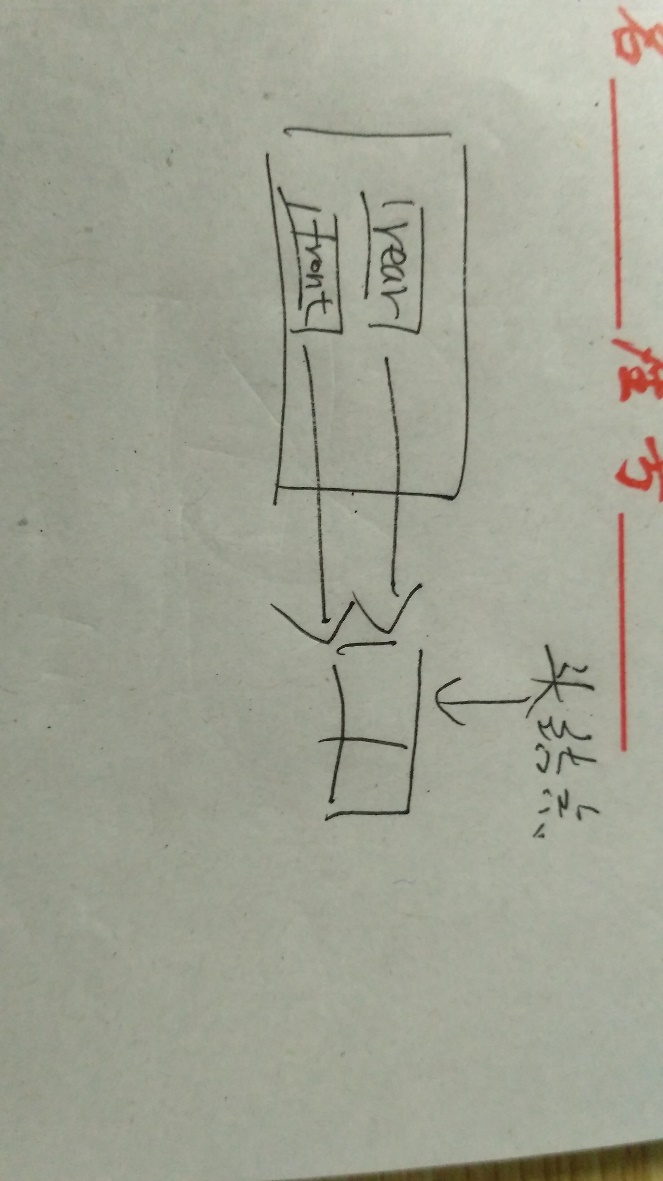
就是一个链表，只是操作只能在表头及表尾。

初始化：

 说明：申请一个节点，赋值给front，和rear，

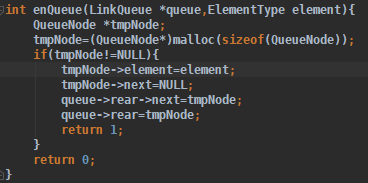
链队的初始化代码

然后将rear节点指向的next置为NULL。

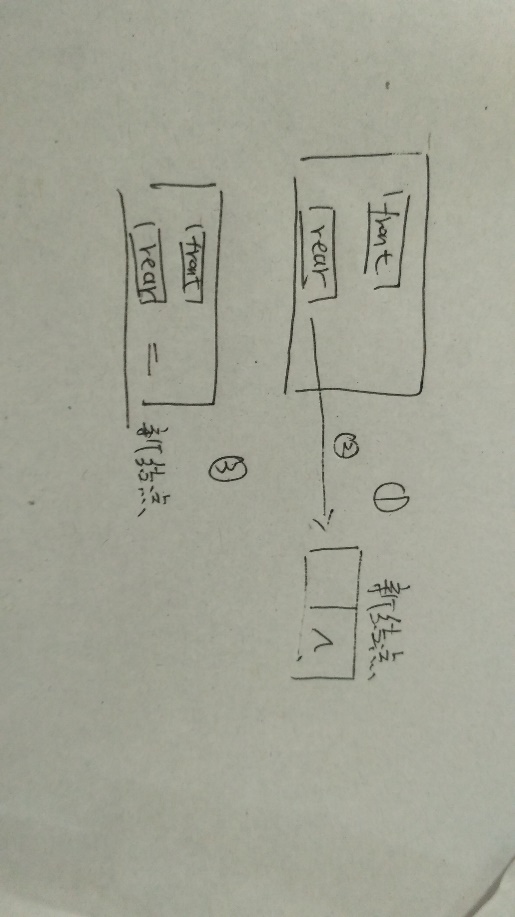


链队初始化图示

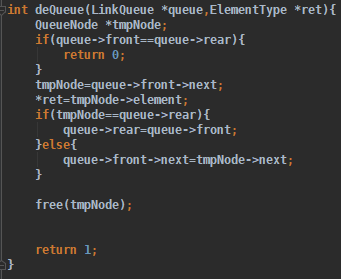
入队：

 说明：首先申请一个节点，将欲入队的元素放入这个节点，然后，将rear所指节点的next成员赋值为新申请的这个节点，最后，将链队的rear指针指向新申请的这个节点。

链队入队代码

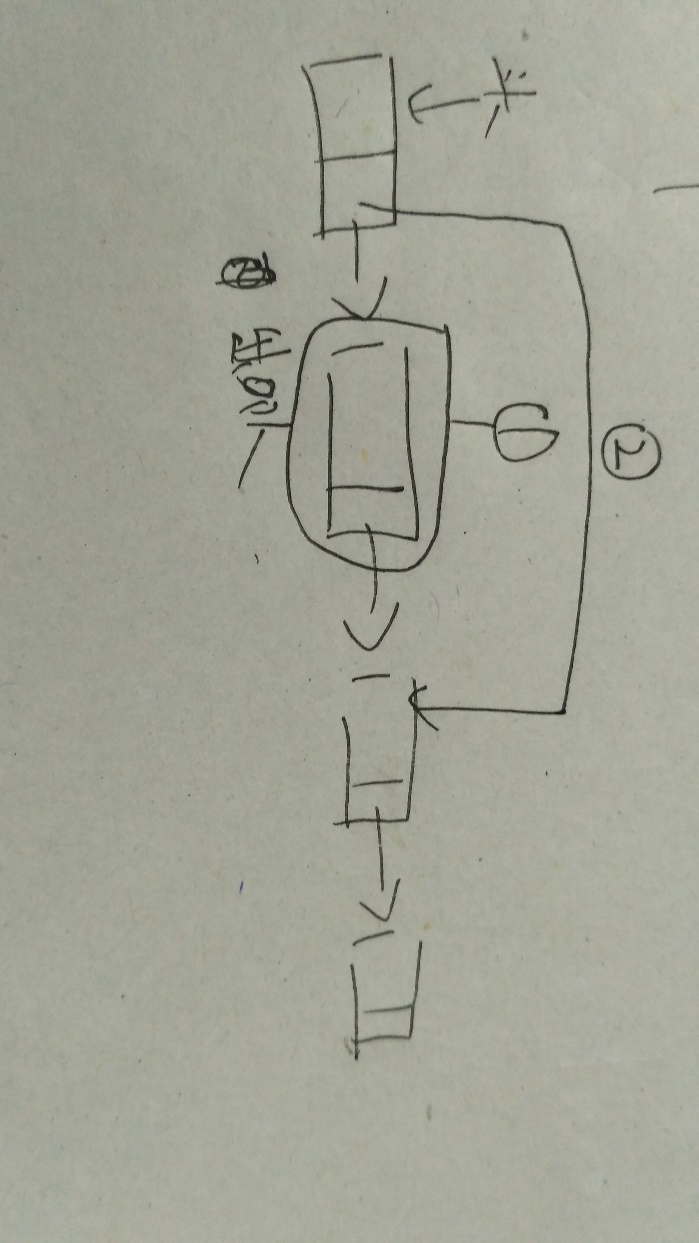


链队入队图示

 出队：

链队出队代码

说明：首先判断是否是空队列（rear==front）。接下来，取出队列的第二个节点（第一个节点为头结点，一般为空）。取出第二个节点后判断该节点是不是最后一个节点，如果是，将队列的rear赋值为front，否则，将front的next赋值为第二个节点的next，最后释放第二个节点的空间。



链队出队图示

队列的应用：

离散事件模拟。

# 第六章

## 树：

定义与基本术语：

0 ）n个节点的有限集。

在任意一颗非空树：

1 ）有且只有一个根节点。

2）n>1时，其余节点可分为m个互不相交的有限集，每个集合又是一棵树，称之为子树。

**基本术语**：

结点：

结点的度：拥有的子树数。

叶子结点（终端结点）：度为0的结点。

分支结点（非终端结点、内部结点）：度不为0的结点。

孩子：结点的子树的根。

双亲：子树的根上面的结点。

兄弟：同一个双亲的孩子。

祖先：从根到该结点所经分支上的所有结点。

子孙：以某结点为根的子树中的任一结点。

层次：从根开始定义，根为第一层，根的孩子为第二层。

堂兄弟：处在同一层次的结点。

深度（高度）：树中结点的最大层次。

有序树：各子树从左至右是有次序的。

无序树：

森林：m颗互不相交的树的集合。

## 二叉树：

**定义**：

是一种树型结构，特定是每个结点最多只有两颗子树，且子树有左右之分，次序不能颠倒。

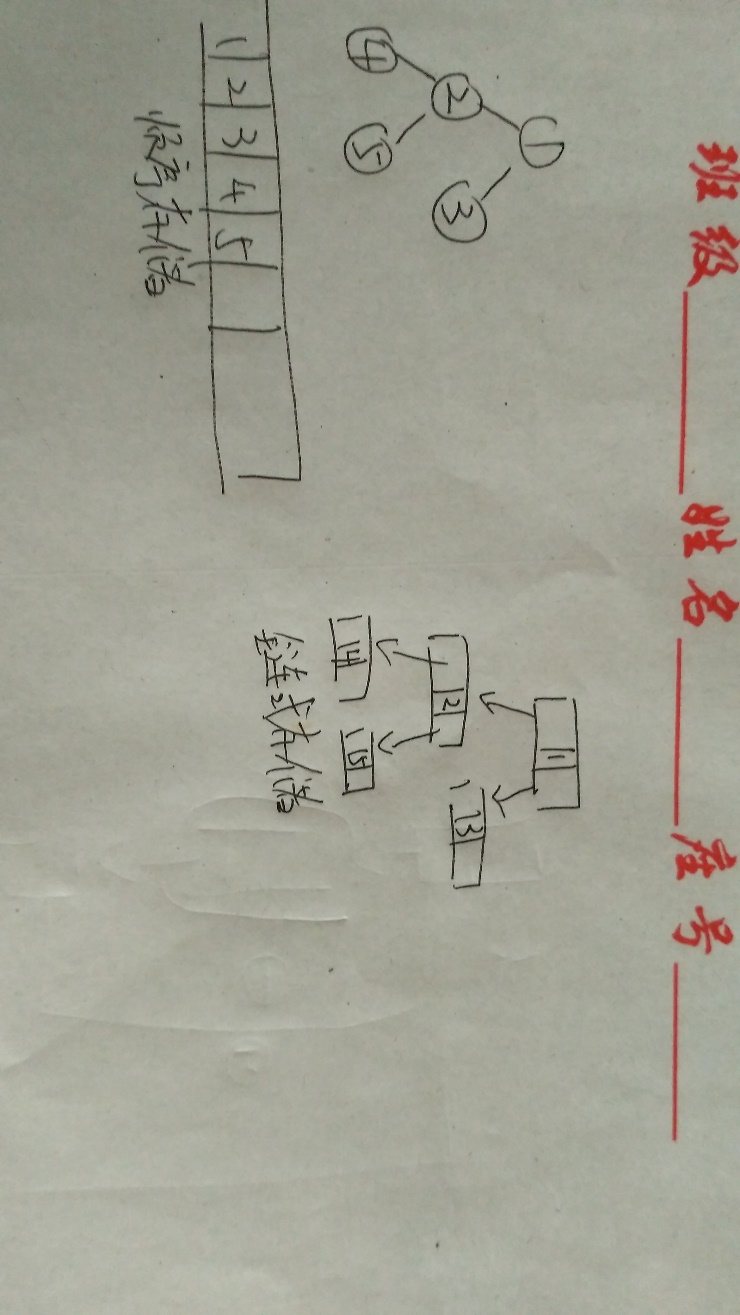
**性质**：

1. 在二叉树的第i层最多有2i-1个结点。
2. 深度为k的二叉树最多有2k-1个结点。
3. 叶子结点个数 = 度为2的结点数+1。
4. 具有n个结点的完全二叉树深度为log2n+1

完全二叉树：

满二叉树：深度为k且有2k-1个结点的二叉树称为满二叉树。

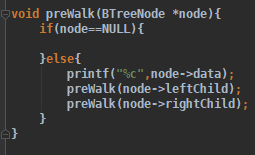
每一个结点都与深度为k的满二叉树的结点编号一一对应，称之为完全二叉树。

1. ****完全二叉树：左孩子=双亲\*2，右孩子=（双亲\*2）+1

**二叉树的存储结构**：

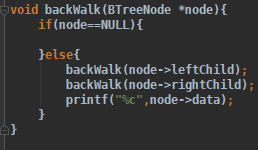
二叉树的存储结构

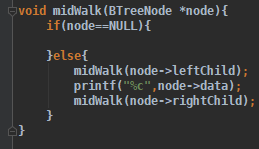
**遍历二叉树**：

前序遍历：

说明：首先判断传入的node是否为空，空的话就直接返回，否则就输出node的内容，然后递归调用当前函数，分别传入左孩子以及右孩子。

中序，后序一样，只是输出node.data的位置不一样。

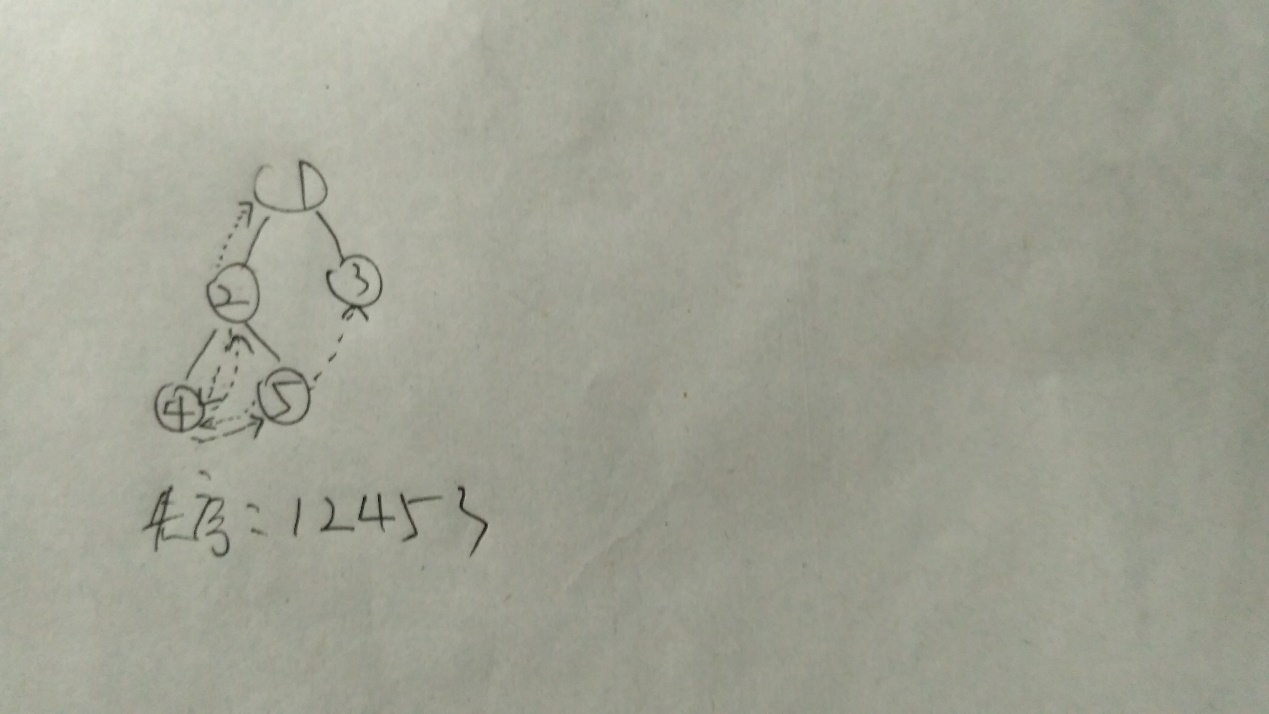




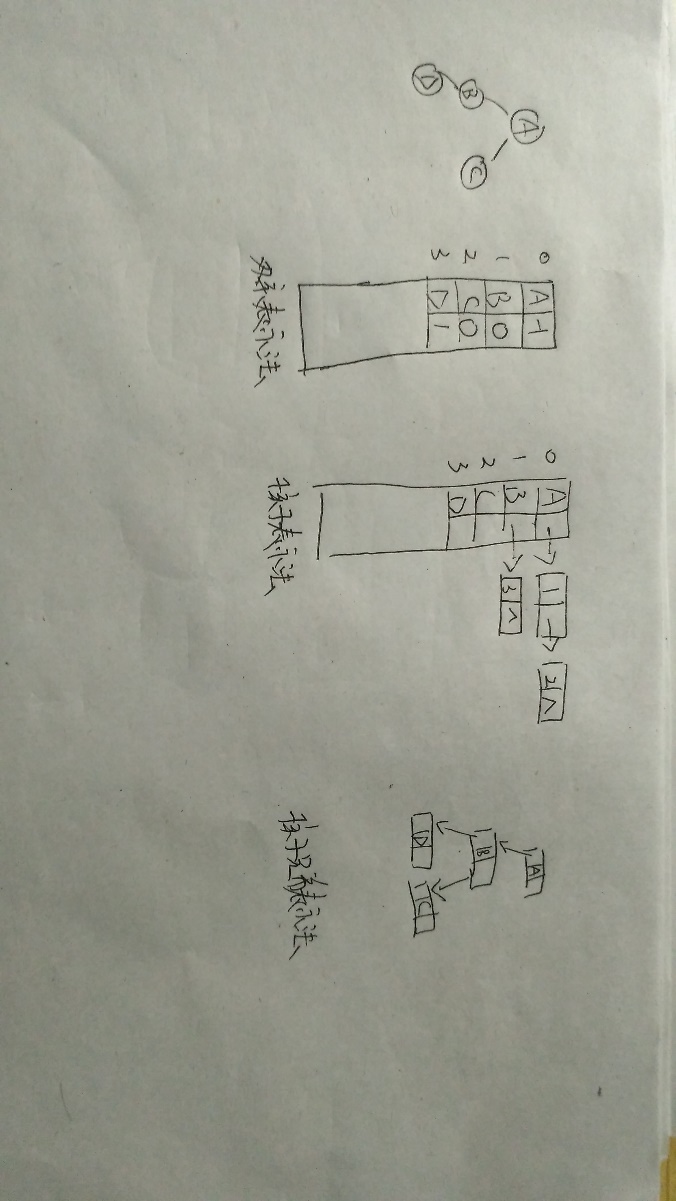
**线索二叉树**：

给二叉树加上线索：

1. 写出二叉树的某种遍历序列。
2. 在图上连接每个结点的先驱及后继。

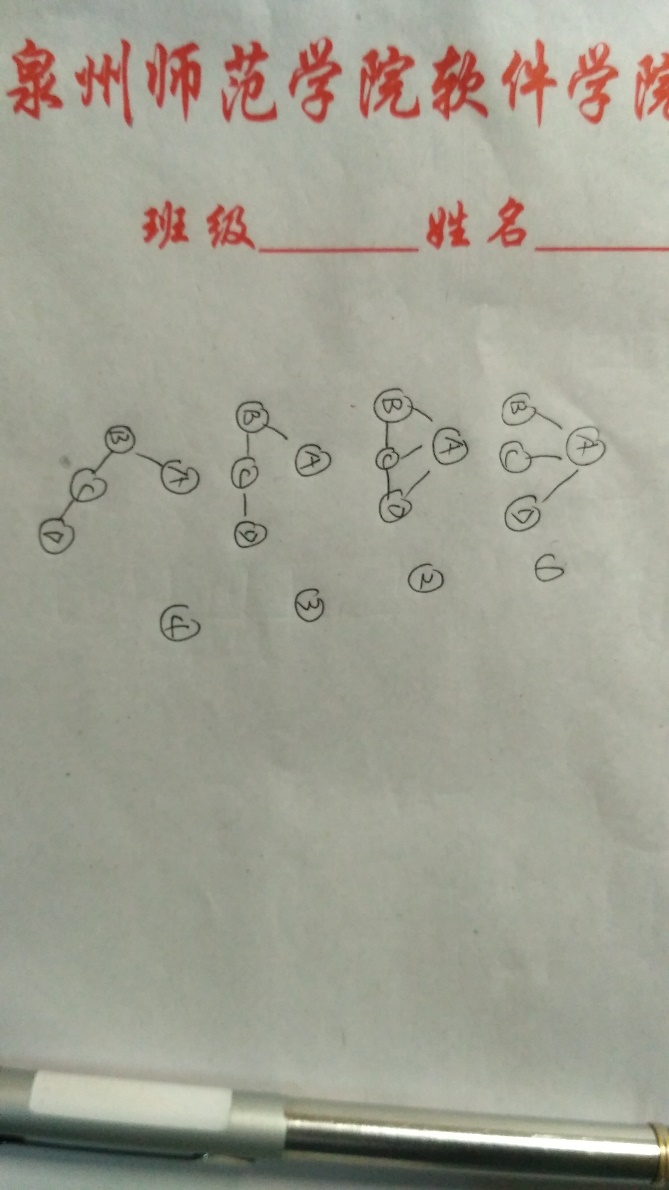


**树的存储结构**：



**树转二叉树**：

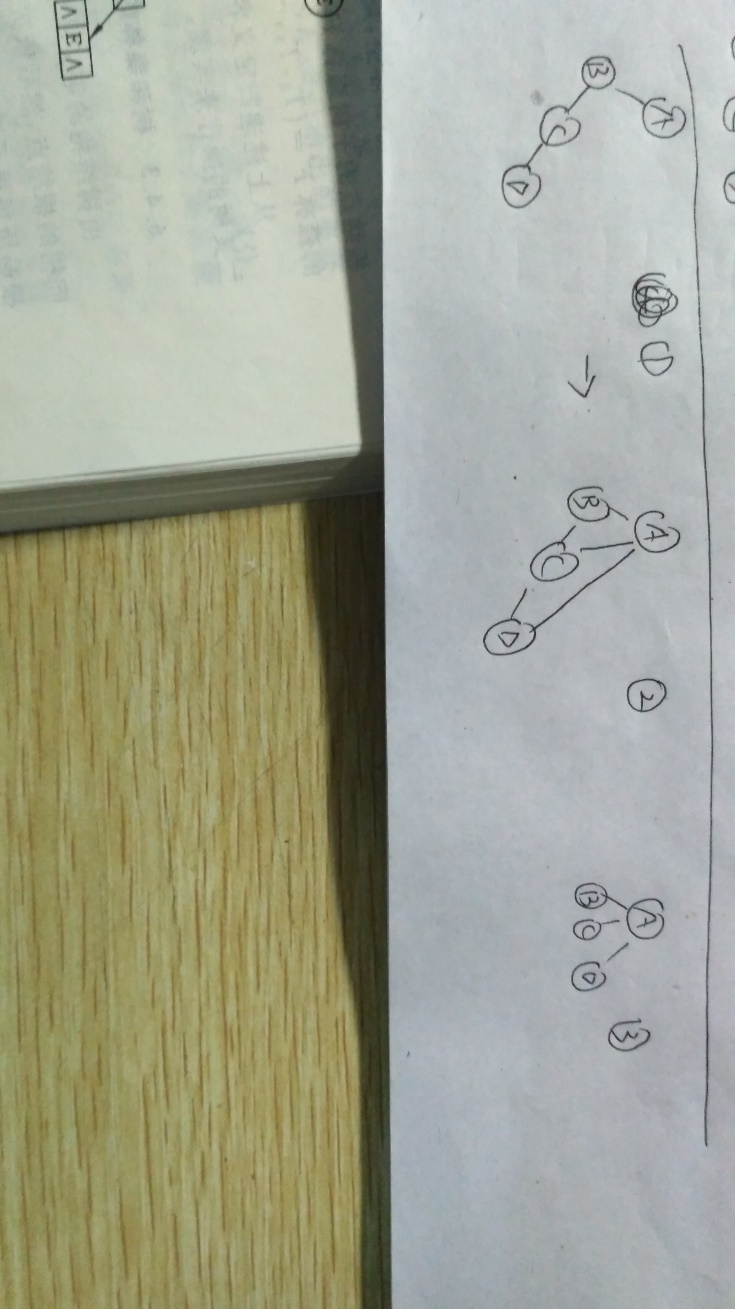
1. 先把兄弟结点连起来。
2. 去掉除第一个孩子外的所有线
3. 右边的结点成为左边结点的右子树。



树转二叉树

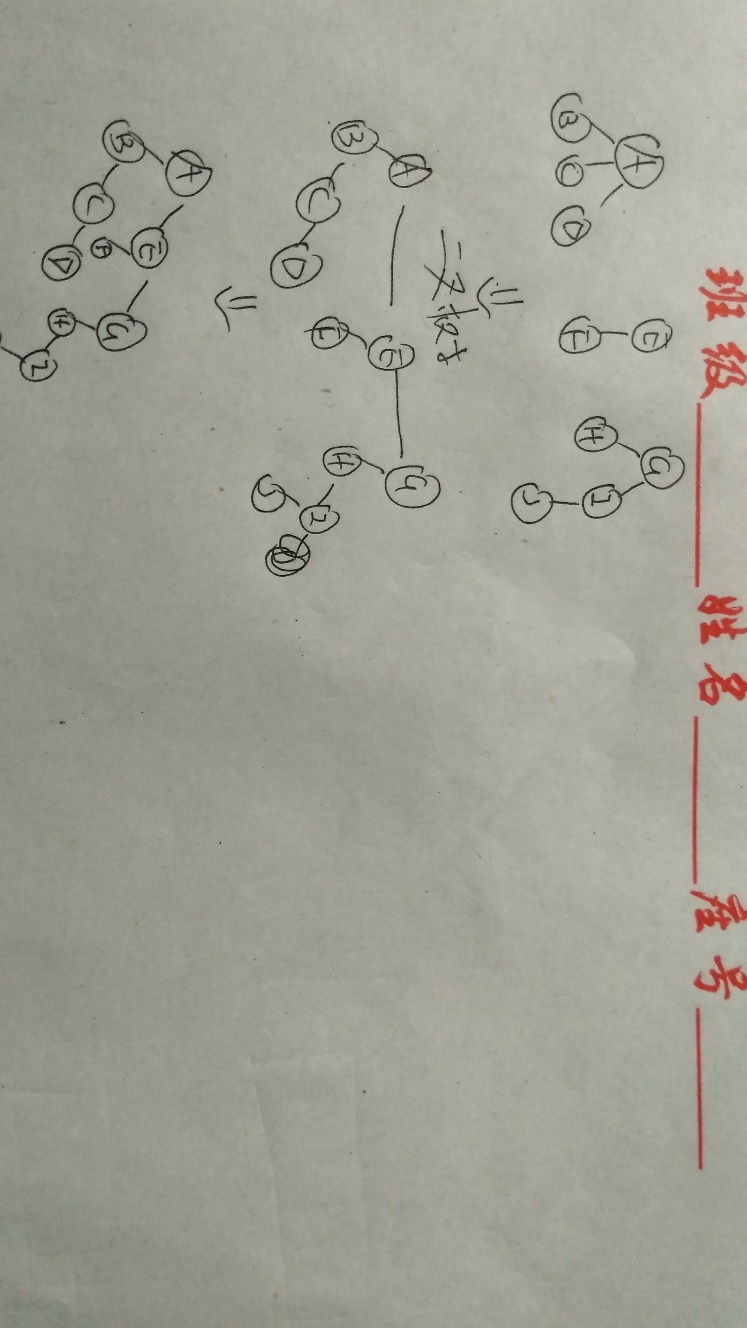
**二叉树转树**：

1. 将每个作为右子树的结点与离他最近的非父结点连线
2. 去掉每个结点右子树的连接。



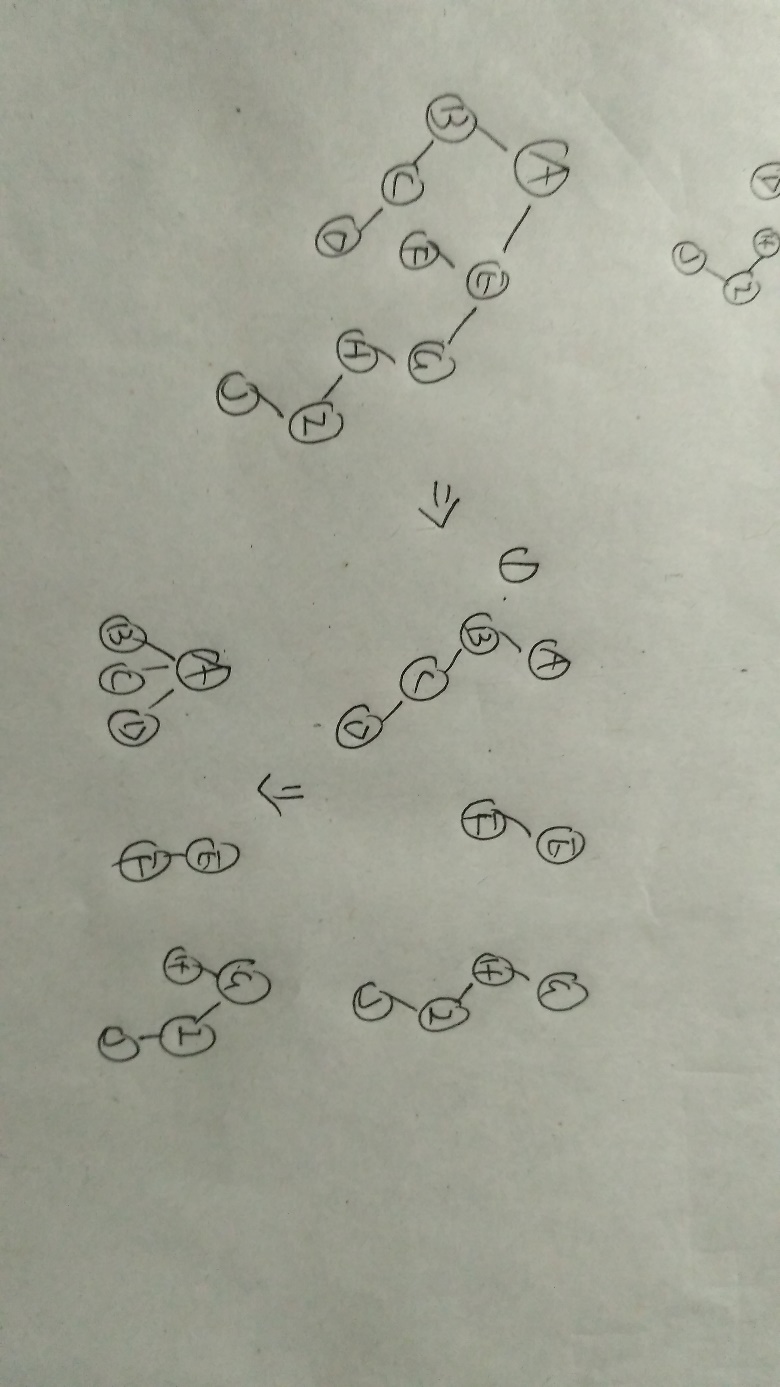
**森林转二叉树**：

1. 将每颗树转成二叉树。
2. 将每个二叉树的根节点相连。



**二叉树转森林**：

1. 如果一棵树有右孩子，那就把右孩子与根节点去掉，如果右孩子还有右孩子，一次类推，直到得到n颗二叉树。
2. 然后把这n颗二叉树转为树即可。



**森林的遍历**：

先根：将森林转成二叉树后进行前序遍历。

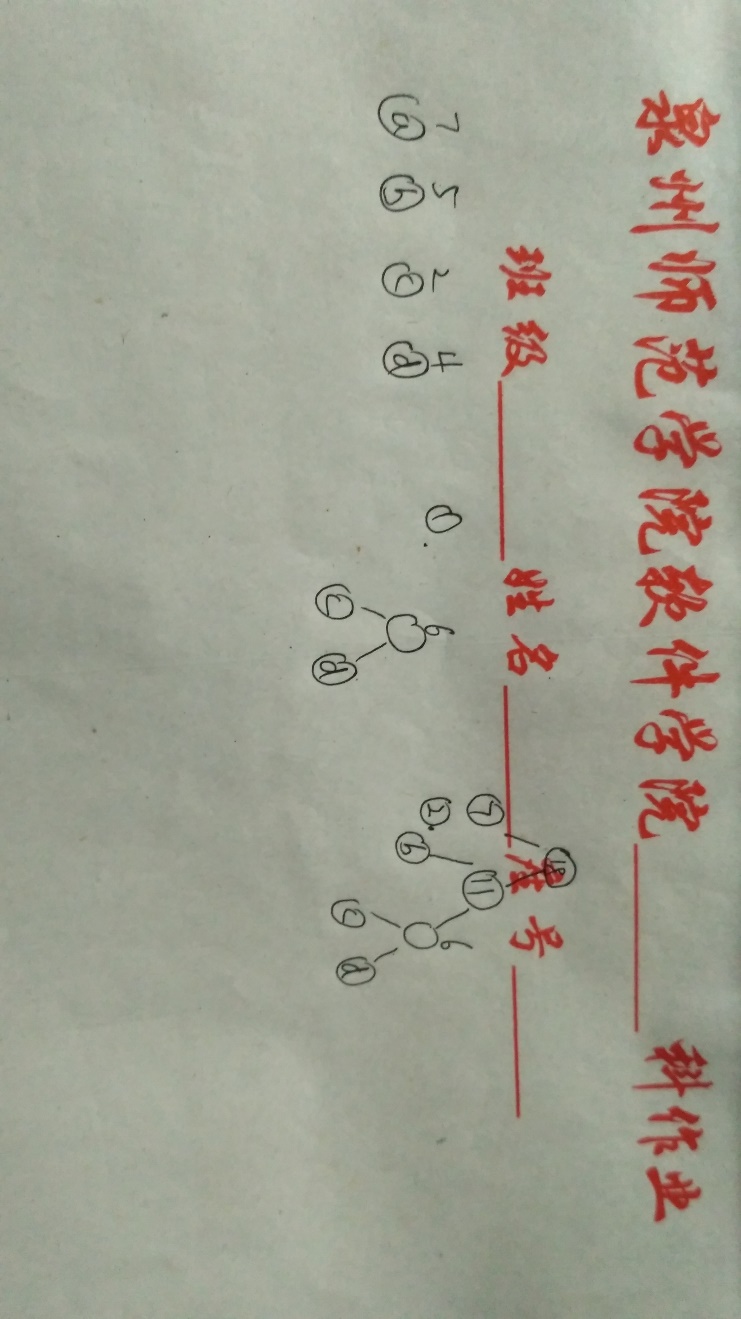
后根：将森林转成二叉树后进行中序遍历。

**哈夫曼树：**

又称最优树，是一类带权路径长度最短的树。

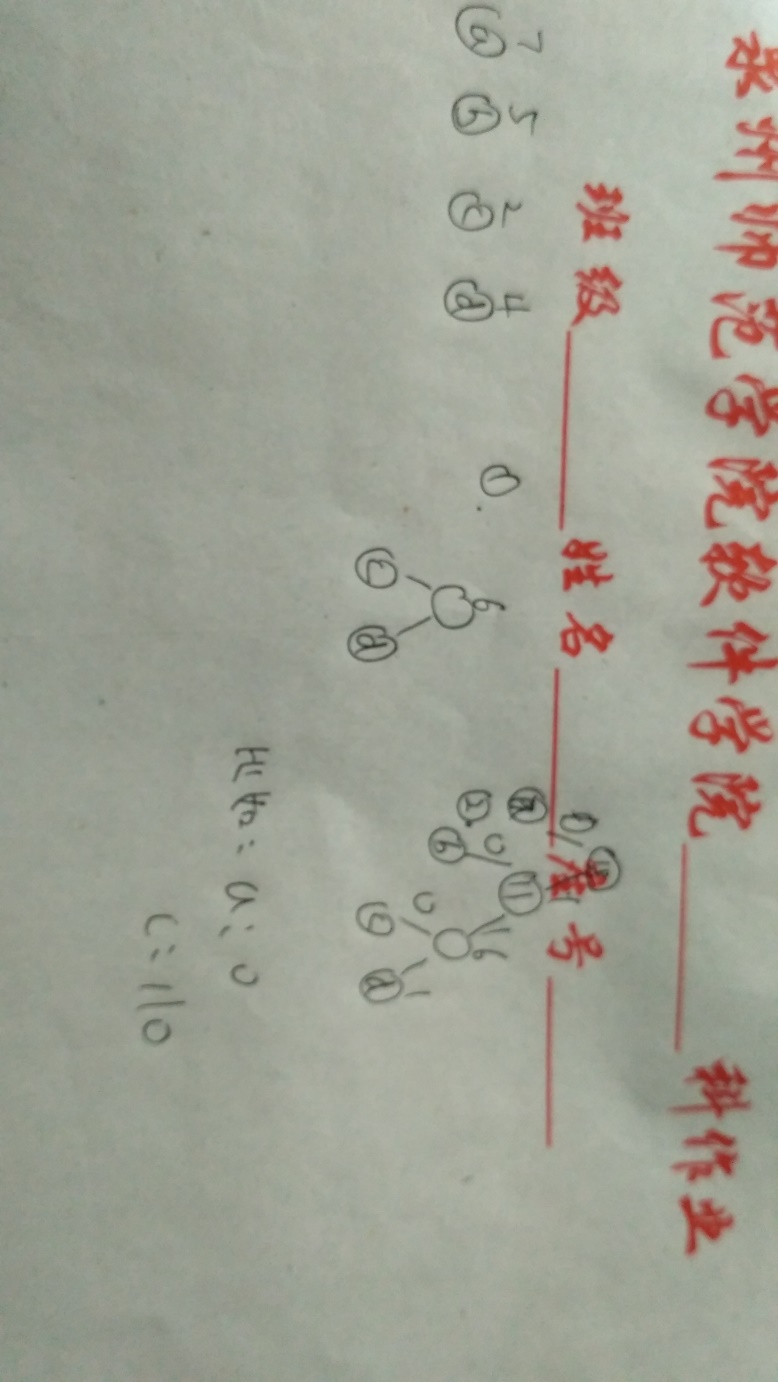
哈夫曼树的构造：

1. 将各个节点按照权值进行排列。
2. 找出两个最小的结点，构造一课树。
3. 将构造的树放入列表
4. 重复2，直到构造成一棵树。



哈夫曼编码：

根据给出的字符出现频率构造一课哈夫曼树，然后左子树为0，右子树为1.根据字符走过的路径输出相应代码。



WPL的计算：

所有（叶子结点的权值\*到达该结点的路径长度）

# 第七章

**图**：

结点的之间的关系是任意的。

术语：

顶点：图中的数据元素。

有向图：边是有向的。

边：两个元素之间的线。

无向图：边无向。

完全图：设顶点为n，则拥有1/2n（n-1）条边的无向图称为完全图。

有向完全图：n（n-1）条弧的有向图。

稀疏图：边数<nlogn

稠密图：上反。

权：与边或者弧相关的数。

子图。

邻接点。

入度：连入的边数。

出度：发出的边数。

连通图：各个顶点都是连通的。

强连通图：任意顶点之间都存在路径。

N个顶点的无向图最少需要n-1条边才能构成连通图。

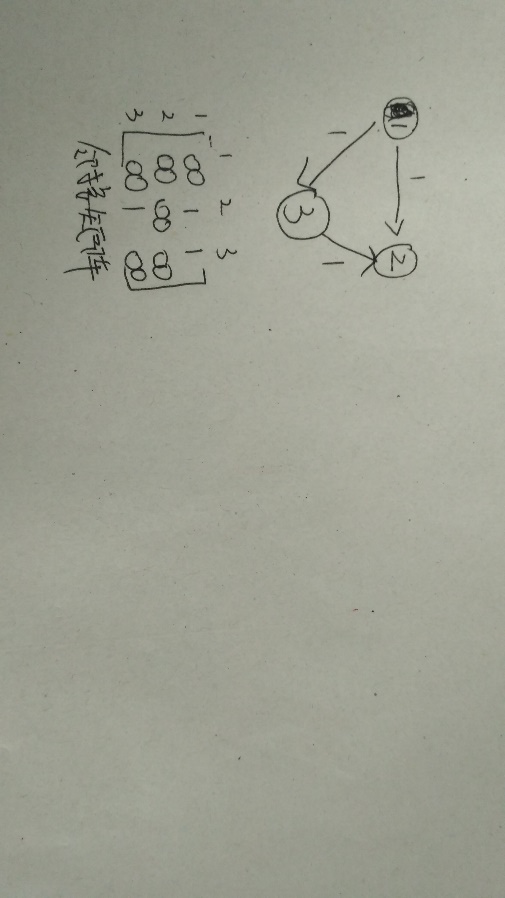
有向图 n 强连通图。

生成树：

极小连通子图。（没有环路）

生成森林。

**图的存储结构：**

二维数组表示：

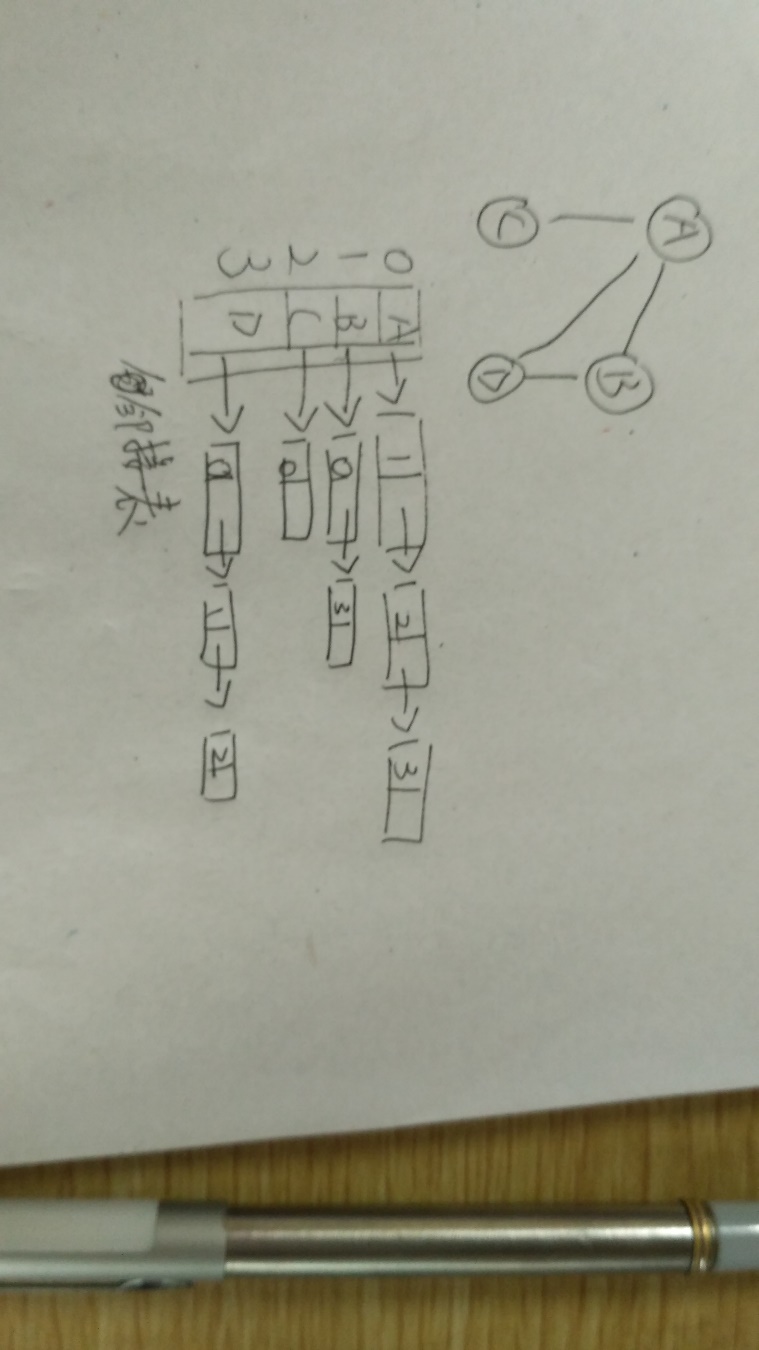
邻接矩阵：

行为出度，列为入度。

解释：把行看为i，把列看为j，如果i->j相连，那么数组[I,j]=1，否则为0（有权图为无穷，数字为权值）。

邻接表表示：

其中，数组中存放的是图的顶点，指针指的是该从该顶点出发能连通的顶点。



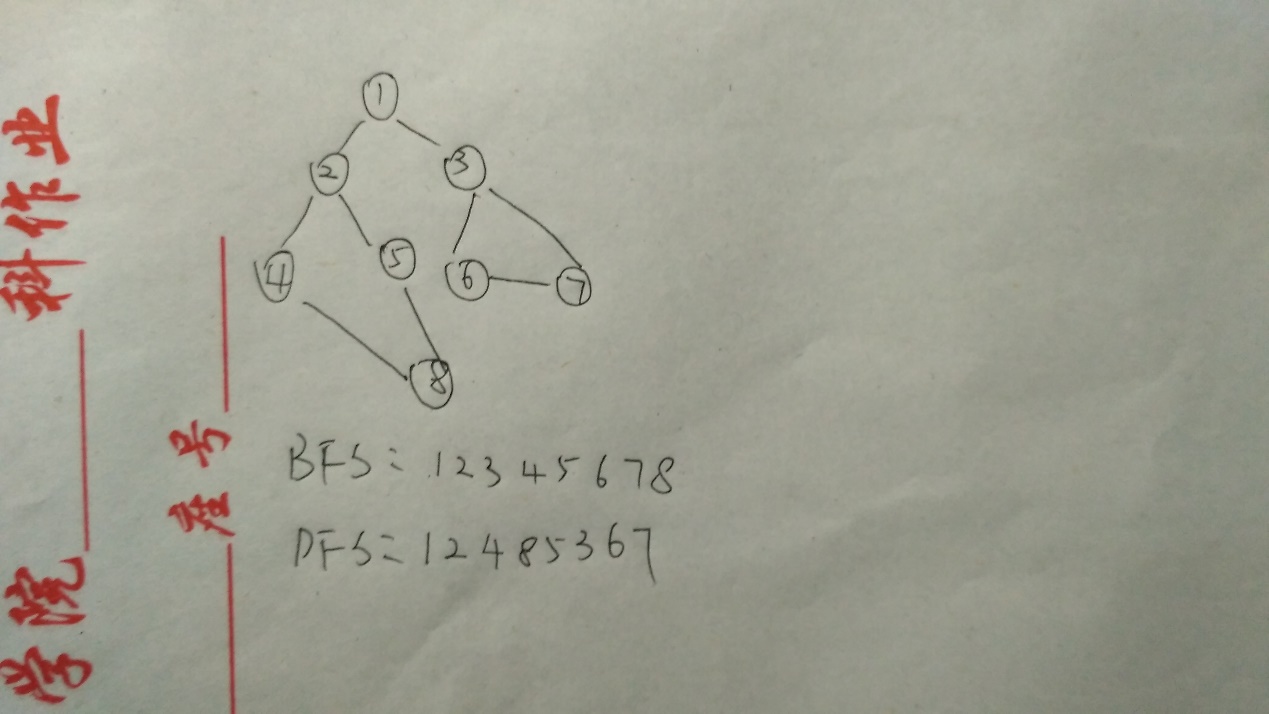
**图的遍历**：

深度优先搜索（DFS）：

选择一个起始点，访问它的一个邻接结点（该结点未被访问过），如果该结点的所有邻接结点都被访问过了，回退到上一步操作，直到访问的结点数等于图中所有结点数。

广度优先搜索（BFS）：

选择一个起始点：访问他的所有邻接结点，然后继续访问所有邻接结点的邻接结点，直到所有结点都访问一遍。



图的两种遍历

**迪杰斯特拉算法**：

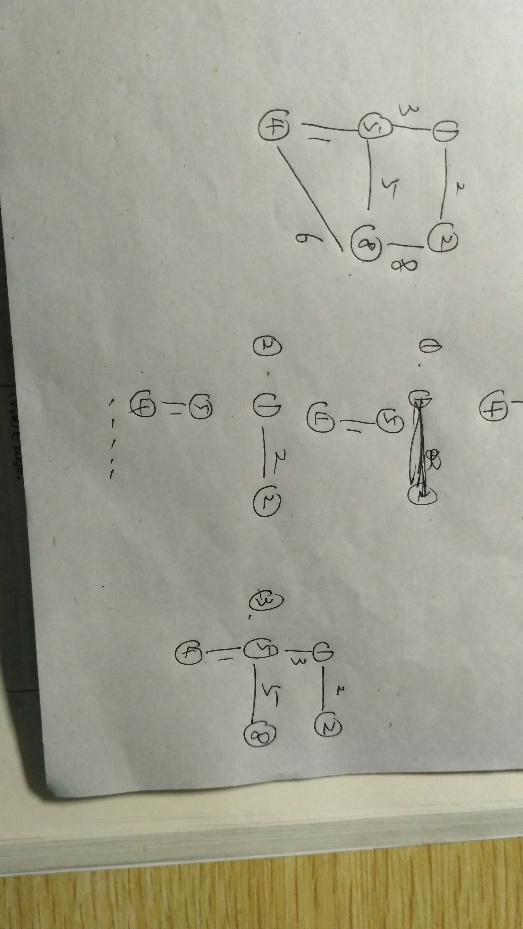
1. 记录当前能到达的结点的最小值，然后将这些结点记录下来备用（无法到达就记录为无穷）。
2. 从备用结点中，看能不能使当前所记录的结点的权值更小，如果有，替换为更小的值。
3. 以此类推，得到从起点到各个节点的最小值。

**最小生成树**：

Prim算法

Prim算法：

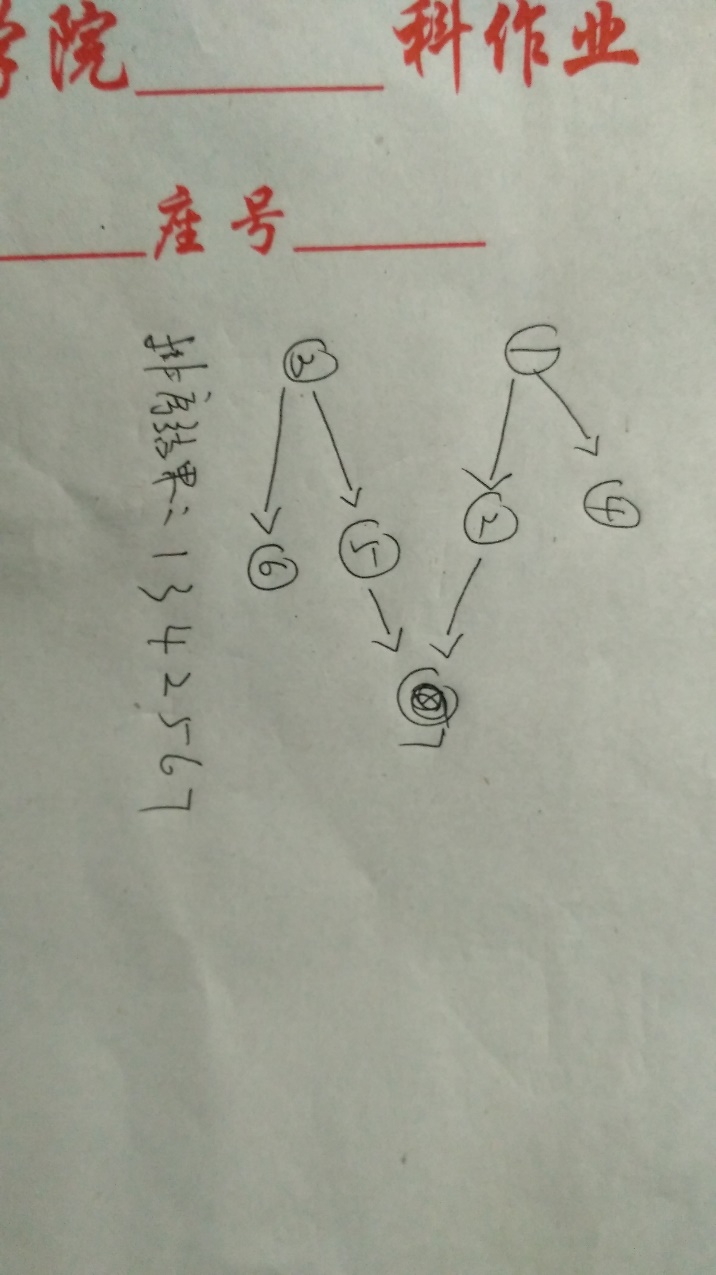
首先选取一个节点，然后选择代价最小且与当前已选择节点相连的边，并且选择的边不能导致当前的图形成环路，重复这一操作，直至构成一个n-1条边的连通图。



克鲁斯卡尔算法

克鲁斯卡尔：

选择代价最小的一条边，将两个节点连起来，并且选择的边不能导致环路，重复上一步，直至构造完成。

**拓扑排序**：

拓扑排序

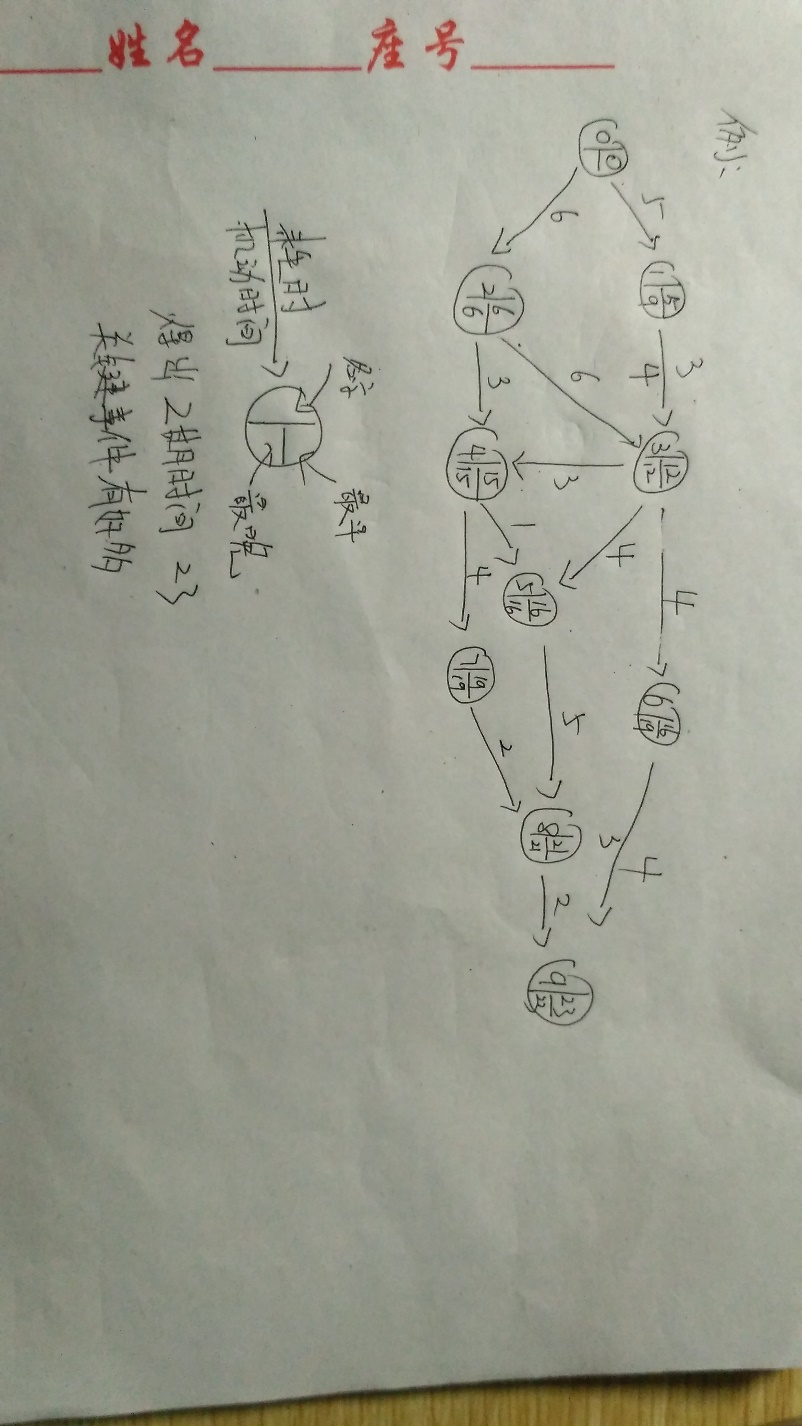
过程：输出没有前驱节点的节点，删掉输出节点的所有出度，重复上一步，直至完成。

**关键路径**：

一般求工期完成时间、关键事件。

过程：

1. 求出每个结点的最早发生时间（到达该结点的最晚时间）
2. 求出每个节点的最晚发生时间（从最后一个节点到达该结点的最早时间
3. 求出每条边的机动时间（后一节点的最晚发生时间-边的时间-前一节点最早发生时间）。



# 第九章

ASL（平均查找长度）:每个元素查找次数之和除以元素个数。

**二分查找（折半查找）**：

1. low=0，high=数据项长度，mid=（low+high）/2 取整数部分。
2. 如果欲查找的元素等于mid，则查找结束。
3. 如果欲查找的元素小于mid，那么high=mid-1，反之low=mid+1。
4. 如果low>high，查找结束，否则回到操作1。

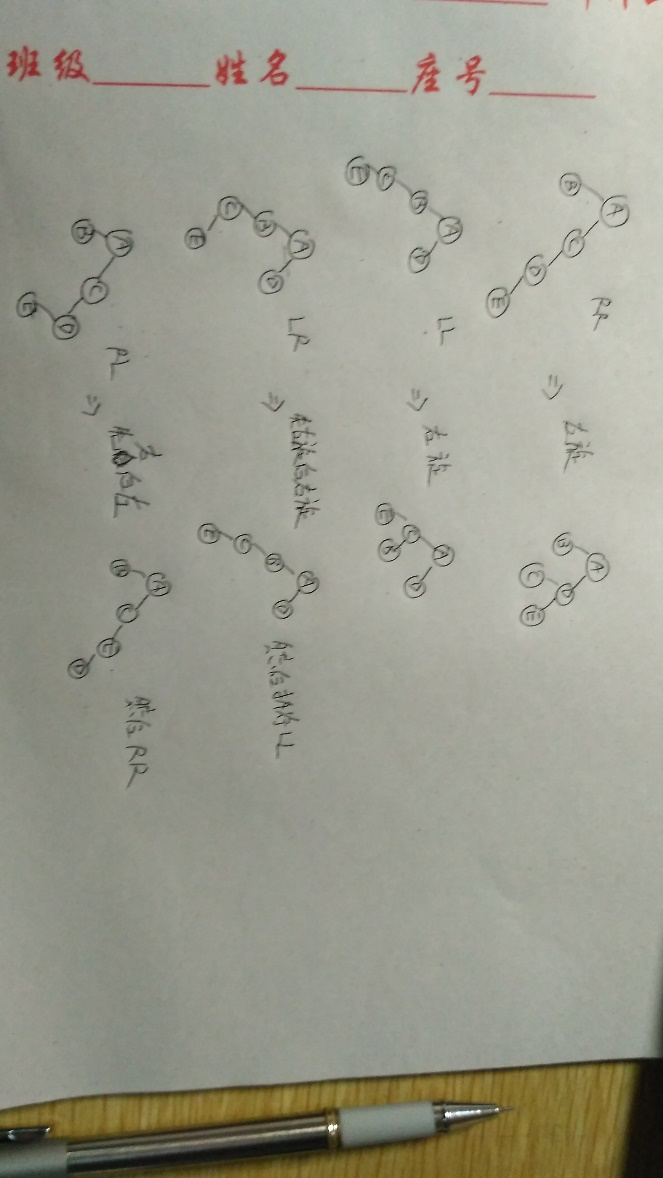
**二叉排序树的构建**：

1. 选择mid所指的元素作为根节点。
2. 令high=mid-1，找出下一个mid所指的元素，让该元素作为根节点的左孩子。
3. 以此类推，得到左子树，右子树也是一样。

**平衡二叉树**：

是空树，或者左右子树都是平衡二叉树，且左右子树高度之差小于1.

四种失衡处理：



B树：

生成：m阶B树每个结点的关键字数n (m/2)-1 <= n <= m-1

1. 依次读入关键字，加入到叶子结点上。当叶子结点关键字数超限，则进行分裂。
2. 分裂：中关键字上提到其父节点，其余关键字分裂成其左右结点。

插入：如上。

删除：两种情况：

1. 删除非叶子结点，删除之后寻找其左子树叶子结点上最大的关键字替换或者是右子树上最小的关键字。
2. 删除叶子结点，如果删除之后仍然符合B树的结构，则直接删除，否则向其兄弟借，其次向其父亲借，如果无法借，则三者合并成一结点，若超限执行上提。

**哈希表**：

通过一个哈希函数（输入一个关键字，这个函数会计算这个关键字的存放地址）。

构造方法：

1. 直接定址。
2. 数字分析。
3. 平方取中。
4. 折叠法。
5. 除留余数法（取模）。

把关键字对给定的哈希表长度进行取余，得出的余数就是该关键字在表中的存放位置。

如果冲突：也就是计算出来的位置已经有元素了，那就采用：

1. 线性探测：

每次加1，如果是空位置，那就存放到该位置，否则继续递增。

2．平方探测：

基本操作如上，只是每次的加1变成-1平方 1 平方…..