# 数据结构复习

## 绪论

正确性 可使用性 可读性 健壮性 效率

-数据与数据结构：由某一数据对象及该对象中所有数据成员之间的关系组成。

-抽象数据类(ADTs: Abstract Data Types)型及面向对象概念

-数据结构的抽象层次:线性聚类(直接存取、顺序存取)、非线性聚类(层次聚集、群聚集)。

-算法的性能标准：正确性、可使用性、可读性、效率、健壮性

-空间、时间复杂度度量: 算法原地工作的含义是空间复杂度为O(1)。



## 数组

### 2.1一维数组

它是连续存储的线性聚集，别名向量，但是可以不连续存放。

一维数组的特点: 除第一个元素外，其他每一个元素有一个且仅有一个直接前驱。除最后一个元素外，其他每一个元素有一个且仅有一个直接后继。

可以根据它推广到二、三甚至多维数组。

### 2.2顺序表

顺序表的定义和特点: n个表项的有限序列，是一种顺序存取，逐项访问的存储方式。顺序表上线性表的数组存储表示，且相邻的元素在物理位置上也相邻。顺序表元素编号从1开始。

顺序表的类定义: 包含数组指针\*data，最大允许长度MaxSize，当前最后元素下标last。

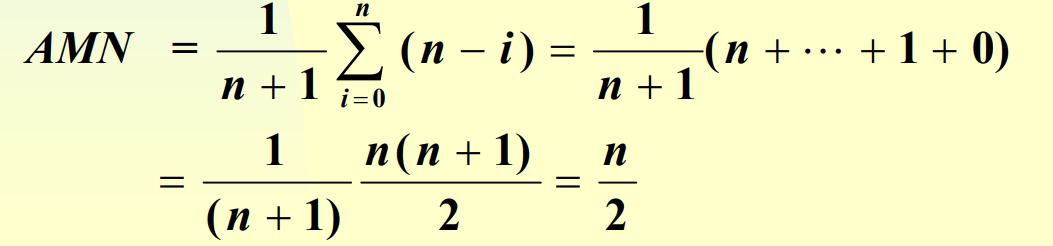
顺序表的查找、插入和删除算法

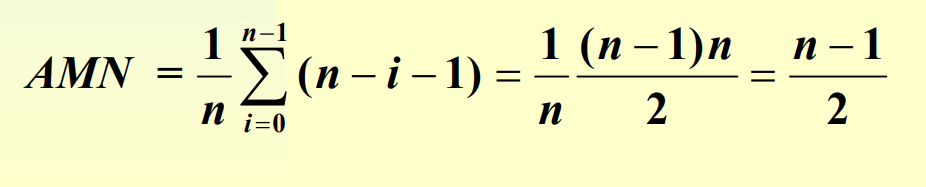
顺序表的查找:从表中从前向后顺序查找x。若结束查找后查找指针i>当前最后元素last，则查找失败。

顺序表的插入:在表中第i个位置插入新元素x。先判断i的合法性，以及last与MaxSize-1是否相等。如果合法，先将i-last的元素逐一后挪。最后将x插入到i位置。i大于等于1且小于等于n+1。

顺序表的删除: 在表中搜索x。若x存在，当前最后元素下标last--，并将i-last的元素逐一前挪，删除完成。若x不存在，删除失败。

在顺序表中插入及删除时计算平均移动元素个数(AMN):





使用顺序表的事例: 集合的“交”运算

多项式的抽象数据类型构造：存储时，可以用静态数组表示，也可以用动态数组表示。他们适用于指数连续排列的多项式，但对于指数不全的多项式不经济。因此可以用一个结构体/类来定义其中的每一项，包含系数和指数两个成员。将每一项的内容放在数组中连续存储。

两个多项式相加: （前提：两个多项式有序存储）需要一个额外的数组另存结果多项式。扫描两个相加多项式，若当前被检测项指数相等，系数相加，系数未变为0则将结果添加到结果多项式；若当前被检测项指数不相等，将指数小者加到结果多项式。如果有一个多项式已检测完，将另一个多项式的剩余部分复制到结果多项式。

### 2.3稀疏矩阵

稀疏矩阵的三元组表表示: 定义一个类，其中私有元素包含非零元素所在行号、列号以及值。

稀疏矩阵的转置算法: （前提：转置三元组表，三元组表有序存储）矩阵列数为Cols，共扫描Cols次。在第k次扫描所有列号为k的项，交换行列号，顺次存储在转置矩阵三元组表中。**该算法的时间代价为O(Cols\*Terms)**。

快速转置算法：建立辅助数组rowSize和rowStart，分别存储转置后各行非0元素个数和各行元素在转置三元组表中开始存放位置。扫描矩阵三元组表，根据某项的列号确定其转置后的行号，再根据行号和rowStart将该项存入三元组表中。**该算法的时间代价为O(max(Terms,Cols))**。

### 2.4 字符串

串是一种特殊的线性表，它的数据元素是一个字符。

串的模式匹配是在串中寻找子串(第一个位置)在串中的位置。

## 链表

### 3.1单链表

链式存储的条件下，节点内的存储单元一定连续，节点间的存储单元不一定连续。

单链表的结构和类定义: 单链表中每个元素由节点ListNode构成，包含element和指向下一项的指针\*link，链表中还包含头尾指针。由链表节点(ListNode)类、链表(List)类和链表游标(Iterator)类表达一个单链表概念。

单链表中的插入与删除

单链表的插入: 在链表第i个结点处插入新元素x。分为在第一个结点前插入，在链表中间插入和在链表末尾插入三种情况。首先将指针p从第一个元素开始遍历，找到第i-1个结点。若查找失败，返回失败。创造新结点newnode并分配内存空间。若表头指针为空或者i=0，那么说明为情况1，将first指针中的内容接在newnode->link中，如果first指针是空的，说明链表为空，last=newnode，然后first=newnode；若为其他情况，那么说明为情况2/3，将newnode的下一个元素即为原本第i-1个结点的下一个元素，即p中的下一个元素。如果p中的下一个元素为空，那么last=newnode，p->link=newnode。

单链表的删除: 在链表中删除第i个结点。分删除表头元素和删除表中或表尾元素两种情况。

带表头结点的单链表: 表头结点位于表的最前端，本身不带数据。设置表头结点的目的是统一空表与非空表的操作，简化链表操作的实现。表头结点和表头指针有区别。

用模板定义的单链表类: 在单链表的类模板定义中增加了表头结点。

链表的游标类(Iterator): 它主要用于单链表的搜索，包含一个数据成员current记录对单链表最近处理到哪一个节点，是List类和ListNode类的友元。它提供一些测试和搜索操作，比如检查链表中当前指针是否非空，返回链表表头指针等等。

静态链表: 利用数组定义，运算过程中存储空间大小不变。取第i个元素只能从表头结点开始沿链顺序查找，查找时间和i有关。如果静态链表的存储数组是用指针定义且动态分配的，那么它可以扩充。

### 3.2循环链表

循环链表的类定义: 循环链表是单链表的变形，它的最后一个节点的link指针不为NULL，而是指向了表的前端。它的特点是只要知道表中某一结点的地址，就可以搜寻表中所有结点。

用循环链表解约瑟夫问题；

### 3.3多项式及其相加

多项式的类定义: 包含coef、exp、\*link指针三个元素。

多项式的加法

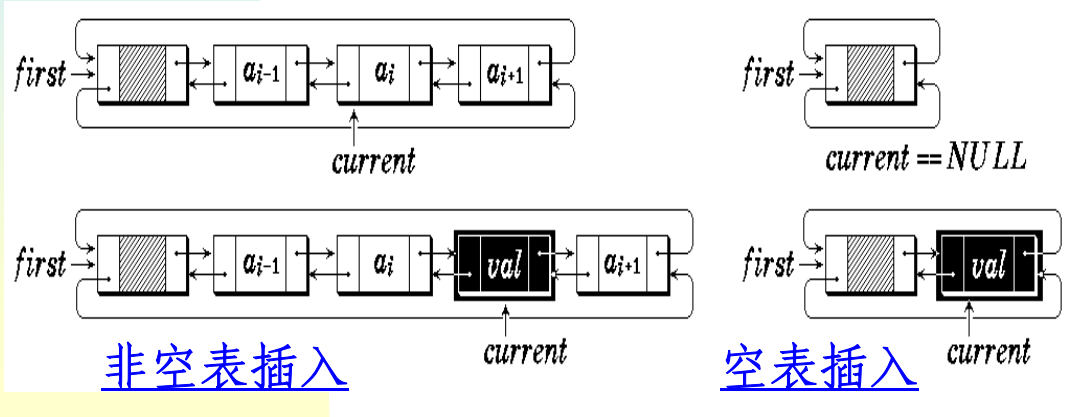
### 3.4双向链表

双向链表在前驱和后继方向都能遍历的线性链表，包含\*lLink、data、\*rLink三个元素。双向链表经常采用带表头节点的循环链表形式。

双向链表的操作算法

双向链表的搜索算法: 将搜索指针\*p初始化为first->rLink，巡链搜索含target的结点。若p等于first，且还未搜索到target，则搜索失败。

双向循环链表的插入算法: 分为空表插入和非空表插入。



双向链表的删除算法: 操作结点的左节点的右节点等于操作结点的右节点，操作结点的右节点的左节点等于操作结点的左节点。操作完毕后删去操作结点，并根据操作结点的左结点是否为头结点操作current。

### 3.5稀疏矩阵的链接表示

稀疏矩阵的链接表示采用正交链表，行链表与列链表十字交叉，且两个链表都是带表头结点的循环链表，用表头结点表征是第几行第几列。

## 栈和队列

### 4.1栈

栈是只允许在一端插入和删除的顺序表，包含栈顶top和栈底bottom，具有后进先出的特点。

栈的抽象数据类型: 包含进栈、出栈、取栈顶元素、判断栈空、判断栈满等功能。栈满时入栈会上溢，栈空时出栈会发生下溢。

栈的数组存储表示: 顺序栈，包含栈顶数组指针\*top、指针定义的栈数组\*elements和栈最大容量MaxSize。

多栈处理、栈浮动技术: n个栈共享一个数组空间V[m]，设立栈顶指针数组t[n+1]和栈底指针数组b[n+1]，初始分配空间s=[m/n]，其中有栈满后扩大该栈。

栈的链接存储表示: 链式栈无栈满问题，空间可扩充，且插入和删除仅在栈顶处执行，即在链头执行，它适合进行多栈操作。

栈的应用: 用后缀表达式求值；中缀表达式向后缀表达式转换

### 4.2队列

队列是只允许在一端删除，在另一端插入的顺序表。允许删除的一端叫对头front，允许插入的一端叫队尾rear，它的特点是先进先出。

队列的抽象数据类型: 包含入队、出队、取队头元素、置空队伍、判空、判满操作。

队列的数组存储表示: 用数组实现循环队列，使存储队列的数组可以当做首尾相接的表处理。**front=(front+1)% MaxSize，rear同上。初始时front=rear=0，当front=rear时队空，当(rear+1)% MaxSize == front时队满。**

队列的链接存储表示: 无头结点的单链表。在有多个队列时，通常使用链式队列，因为它的内存分配是动态性的，比较灵活。

双端队列: 允许在两端进行插入和删除(可能会有输入输出限制)。

队列的应用：打印杨辉三角形

### 4.3优先级队列

优先级队列的定义: 不同于队列的先进先出，它每次从队列中取出的是具有最高优先权的元素。数字越小优先权越高。

优先级队列的链接存储表示

优先级队列的应用举例

## 递归与广义表

### 5.1递归

三种用到递归方法的情况: 定义是递归的(斐波那契数列)、数据结构是递归的(单链表)、问题的解法是递归的(汉诺塔问题)。

迷宫问题: 从入口开始，向左前右三个方向以此递归，如果找到出口则输出路径。

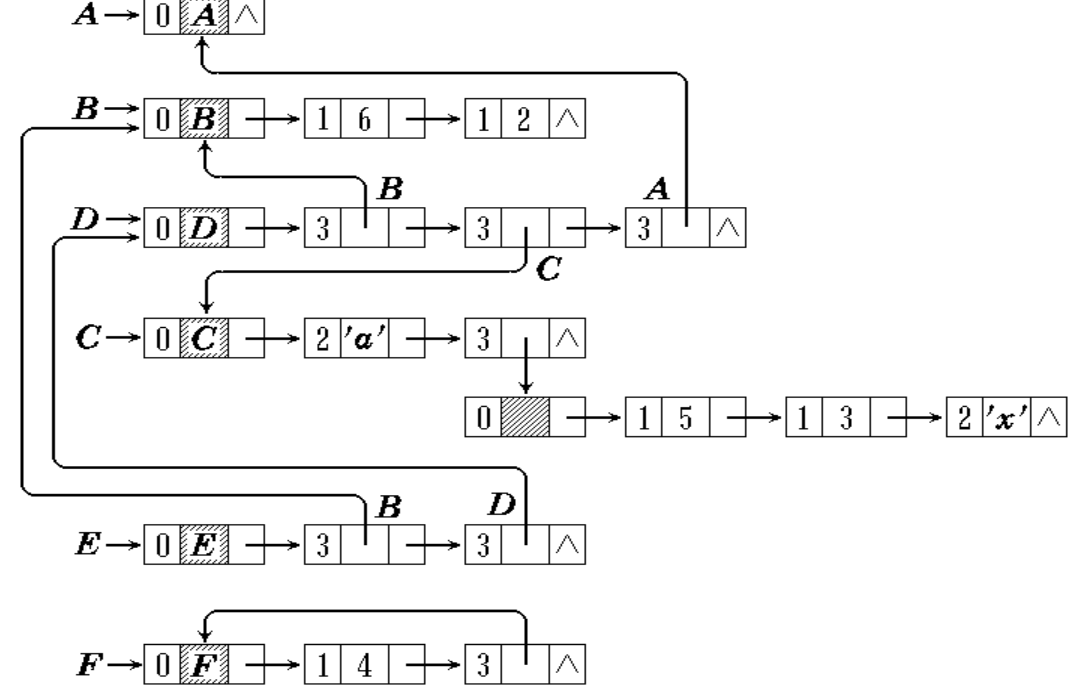
递归过程与递归工作栈：递归过程实现的机制及递归工作栈的引用。每一次递归调用的时候，都需要为过程中使用的参数、局部变量等另外分配存储空间。也可以用递归调用树来表示。

### 5.2广义表

广义表是n个表元素组成的有限序列

广义表的定义：广义表长度为n，n=0时广义表为空；广义表的深度为子表的深度+1；表的第一个表元素称为广义表的表头、**其他表元素组成的表**称为广义表的表尾。

广义表的表示及操作：广义表结点包含标志域utype、值域value和尾指针域tlink三个元素。当utype=0时为表头结点，1为整型原子结点，2为字符型原子节点，3为子表结点。



广义表的递归有向表头递归和向表尾递归两个方向。

## 树与森林

### 6.1树与森林的概念

树是由n个结点组成的有限集合。它有一个特定的根节点，该节点只有直接后继没有直接前驱；除根之外的其他结点划分为m个互不相交的有限集合，每个集合又是一棵树。



结点的深度是从树的根结点到该结点所经历的边的数量。

结点的高度是从该结点到叶子结点所经历的边的最大数量。

### 6.2二叉树

二叉树性质: 若二叉树的层次从0开始，则在二叉树的第i层最多有2i个结点；高度为k的二叉树最多有2k+1-1个结点；二叉树的叶结点个数等于度为2的非叶节点个数加1。二叉树中有完全二叉树、满二叉树。

完全二叉树的性质: **具有n个结点的完全二叉树的高度为[log2(n+1)]-1**；如果将一颗有n个结点的完全二叉树按顺序标号，然后存放在一个一维数组中，则i节点的父节点和子女均可以用编号算出。

二叉树的数组表示：完全二叉树比较适用此方法，一般二叉树会浪费大量存储空间，尤其是单支树。

二叉树的链表表示：有二叉链表(指向左右子女)，三叉链表(指向左右子女和父节点)。

二叉树的遍历: 包含前序、中序、后序、层次遍历。用代码实现时，有非递归遍历算法和递归遍历算法。非递归遍历方法利用栈的特性实现（需要去看一下ppt37-42）先序遍历时栈内元素入栈后立马出栈输出；中序遍历时当左子树为空或左子树遍历完成时当前元素出栈；后序遍历时当左右子树均为空或左右子树均遍历完毕时出栈。

中序遍历的非递归算法：利用栈的特性，先定义一个指针\*p指向根节点，然后将p=p->leftChild，并将每一个p依次存入栈中。当p的左孩子为空时，检查栈中是否有元素，如果有，那么栈顶元素出栈，p变为出栈元素的右孩子。重复第一步，直到p指向空且栈中无元素为止。

二叉树层次遍历算法：在层次遍历算法中用到了队列。首先将根节点加入队列。然后遍历队列，队头元素出队，出队元素的左右孩子依次入队，直到队列为空为止。

二叉树遍历的游标类: 后序、层次游标类（ppt55-57）

线索化二叉树：增加了leftThread和rightThread的二叉树，其中Thread为0时，表示对应结点为子女指针；Thread为1时，表示对应节点为线索指针。掌握前中后继线索二叉树中前驱和后继的查找方法。

前序线索化二叉树：

中序线索化二叉树：如果右节点为后继指针，且非等于根节点，则后继节点为右节点。如果右节点为孩子指针，那么后继节点为当前结点右子树的中序遍历下第一个节点，如果右子树的根节点有左孩子，则继续向下查找直到没有左孩子为止，最终的后继节点为没有左孩子的子节点。

后序线索化二叉树：首先找到结点的父节点，再到其父节点的右子树中找后序下的第一个节点。这种方法比较复杂，查找效率较差。

### 6.3堆

堆可以用完全二叉树来表示，分为最小堆和最大堆。

堆的建立：形成堆时用**向下调整算法**。从最后一个非叶结点开始调整，每次调整后都要向下继续调整，直到最后从根节点开始向下调整为止。

堆的插入：堆插入时用到的**向上调整算法**。在堆的尾部加入新节点，然后向上进行关键码调整。

堆的删除：每次删除位于堆顶(0号位置)的最小元素，然后把堆的最后一个元素挪到0号位置，再从0号位置向下调整为小根堆。

### 6.4树与森林

树的存储方式：广义表表示、双亲数组表示、左子女右兄弟表示

树与二叉树的转换：类似于左子女有兄弟表示法。

树的遍历：树的**先根遍历**等同于二叉树的**先序遍历**，树的**后根遍历**是等同于二叉树的**中序遍历**。树的层次遍历采用的是广度优先遍历算法。

### 6.5霍夫曼树

霍夫曼树的构造方法

霍夫曼编码

带权路径长度的计算

用n个权重构造出来的霍夫曼树共有2n-1个结点，其中有n个叶节点和n-1个度为2的结点，没有度为1的结点

## 集合与搜索

### 7.1集合

集合是成员的一个群集，集合中的成员一般无序，且必须互不相同。

用位向量(bit Vector)实现集合：用位(0,1)向量来实现集合

用有序链表实现集合：为集合的每一个对象建立一个带表头结点的单链表，并建立一个**一维的指针数组seq[n]**作为各单链表的表头结点向量。在输出时设置**一个布尔数组out[n]**用于标记元素是否已经输出。在算法具体实现中**还使用了栈**，每次输出一个对象编号时，都会把这个编号进栈，记录还要检查输出的等价对象的单链表。

### 7.2并查集

等价类：等价类是一个对象的集合，在此集合中所有对象应满足等价关系。等价关系是集合上一个自反、对称、传递的关系。

并查集：它是一种建立等价类的解决方法。并查集中每个集合用一棵树表示。它的功能有Find(i)，搜索i所属字集合的名字x；Union(x,y)将x，y所在的集合合并，根为x所在的集合的根。

并查集的优化：Find和Union的操作性能不好，如果最终的有序链表形成了单支树，那么完成n次搜索需要的总时间将达到O(n2)。

1. Union加权操作: 为了避免产生退化的树，改进方法是先判断两个集合中元素的个数，结点个数少的根节点成为结点个数多的根节点的子节点，即将Union进行加权操作。
2. Union折叠操作: 如果j是从i到根的路径上的一个节点，且parent[j]!=root[j],则parent[j]=root[i]。

### 7.3搜索

搜索就是在数据集合中寻找满足某种条件的数据对象。

平均搜索长度：衡量一个搜索算法的时间效率的标准是在搜索过程中关键码的平均比较次数或平均读写磁盘次数，即平均搜索长度ASL(Average Search Length)。

顺序搜索算法：主要用于在线性结构中进行搜索，可以用迭代或递归。数组的0号元素作为监视哨，存放需要搜索的元素。

折半搜索算法：它是基于有序顺序表的搜索方式，它将n个对象存放在一个有序顺序表中，按关键码大小排序。定义low，mid，high三个指针辅助查找。

### 7.4二叉搜索树

二叉搜索树是从有序表构造出来的，它的特点是左子树上所有结点的关键码都小于根节点的关键码，右子树上所有结点的关键码都大于根节点的关键码。需要会计算ASLsucc。包含n个结点的二叉查找树的数量可用Catalan函数计算。

二叉搜索树的插入：插入前需要先判定树中是否有该元素。如果该元素已存在就不再插入。

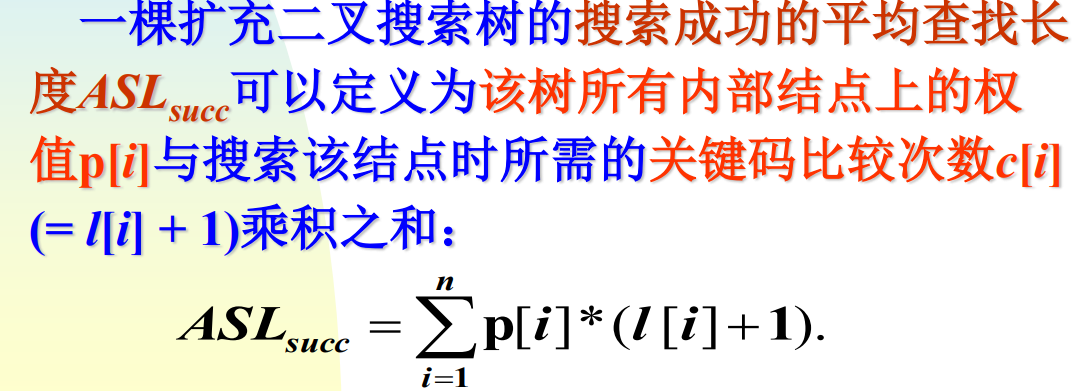
二叉搜索树的建立：建立起来的二叉搜索树和数据输入顺序有关，不同的二叉搜素树会影响二叉搜索树的搜索性能。

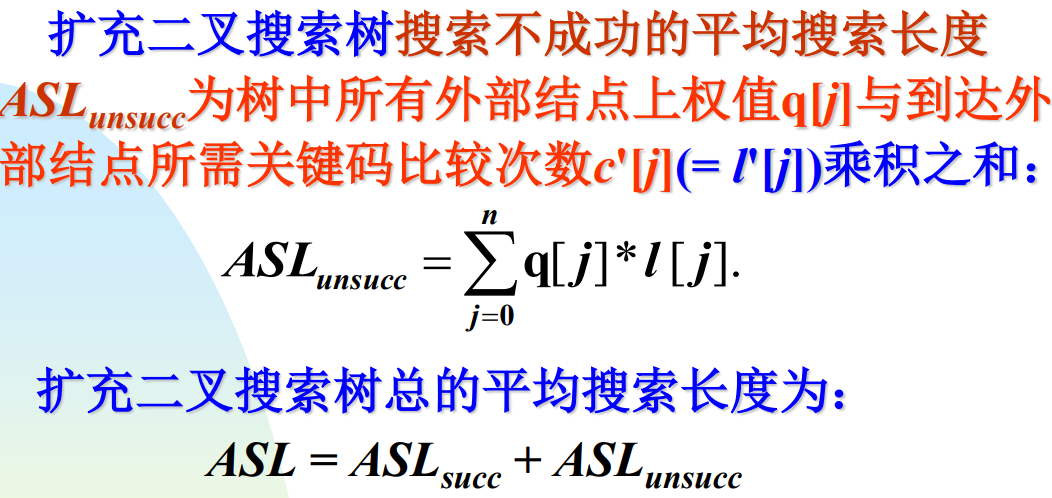
二叉搜索树的删除：分为四种情况。删除叶节点时，只需将父节点指向它的指针清空再释放它即可；被删节点缺右(左)子树时，可以拿它的左(右)子女节点顶替它的位置后再释放它；被删节点左右子树都存在时，可以在他的右子树中寻找中序下的第一个节点(右子树中关键码最小的)再用它的值填补到被删结点中。

二叉树的中序遍历是有序序列。

### 7.5 AVL树

一个n个元素的关键码集合有n个搜索概率，以及n+1个搜索不成功的间隔内搜索概率。可以根据它**计算ASLsucc和ASLunsucc**



定义

构建平衡二叉搜索树：使用W[i][j]来存放它的累计权值和(实例掌握)

AVL树：它是高度平衡的平衡二叉树，它的左右子树都是AVL树，且左右子树的高度差绝对值不超过1。每个结点附加一个数字balance，它是该节点右子树高度减去左子树的高度得到的高度差，AVL树中任意结点的平衡因子只能是-1,0,1。N个结点的AVL树的平均搜索长度为O(log2n)。

平衡化旋转: 分为单旋转(左旋和右旋),双旋转(左右旋和右左旋)

插入后进行平衡化旋转：例题掌握。

删除后进行平衡化旋转：例题掌握。类二叉搜索树的删除，删完后要进行平衡化旋转调整。

AVL树的高度:AVL树最少有多少结点与树的高度h有关。Nh是高度为h的AVL树的最少结点树，那么**N0=0.N1=1,Nh=Nh-1+Nh-2+1。**

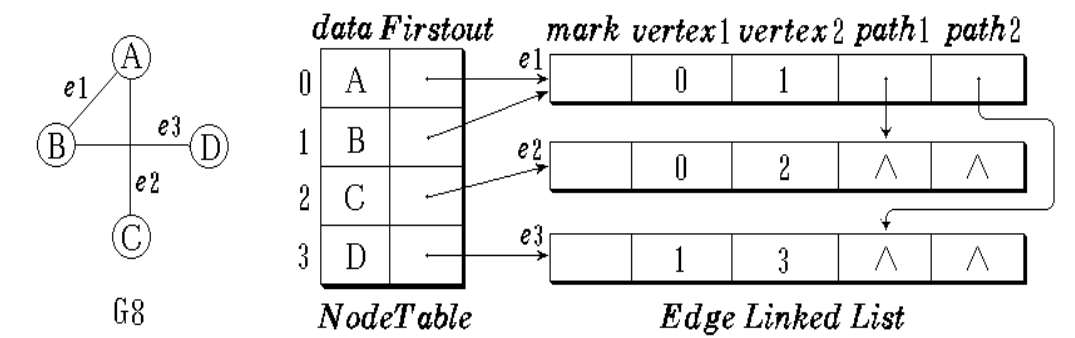
## 图

### 8.1图的存储表示

图的邻接矩阵元素个数与顶点有关，非零元素个数与边有关

图的邻接表既与顶点个数 n 有关，又与边数有关。一个结点链接链表长度为该节点的出度。

图的邻接多重表：以下为无向图的情况。首先是边结点：在邻接多重表中，每一条边只有一个边结点。边结点中结构包括是否处理过该边的标记mark，依附于该边的两顶点位置vertex1和vertex2，指向下一条依附于顶点vertex1的边的链接指针path1，指向下一条依附于顶点vertex2的边的链接指针path2；然后是顶点结点：存储顶点信息的结点表以顺序表方式组织，每一个顶点结点有数据成员data和指示第一条依附于该顶点的边的指针firstout。如果是有向图，则需要在顶点节点中加入一个firstin指针。



连通无向图最多有 n(n-1)/2 条边，最少有n-1条边。

连通有向图最多有 n(n-1) 条边，最少有 n 条边。n = 1无边。

### 8.2图的遍历与连通性

辅助数组的实现：为了避免重复访问，可以设置一个标志顶点是否被访问过的辅助数组visited[i]，它的初始状态为0，被访问后变为1.

深度优先搜索算法DFS：递归或用栈的方式实现。用邻接表表示图时，沿firstout->link链可以找到某个顶点v的所有邻接顶点w，由于总共有2e个边结点，每次扫描边的时间为O(e)，而且对所有顶点仅递归访问1次，所以遍历图的**时间复杂度为0(n+e)**。用邻接矩阵表示图时，遍历图中所有顶点所需的**时间为0(n2)**。

广度优先搜索算法BFS：用队列实现。

连通分量：无向图为非连通图时，从图中某一顶点出发进行的B可能遍历到图中的所有顶点，只能访问到该顶点所在的最大连通子图。

重连通分量与关节点的概念

求解关节点及构造重连通图的方法

### 8.3最小生成树

构造最小生成树的方法

Prim 算法：选点法。从连接网络N中的某一顶点出发，选择与它关联的具有最小权值的边，然后将其顶点加入到生成树的顶点集合U中。再用邻接矩阵作为图的存储方式。在代码实现时，包含辅助数组lowcost和nearvex

Kruskal 算法：选边法。利用最小堆和并查集来实现。

### 8.4最短路径

采用Dijkstra算法实现。用邻接矩阵存图，引入辅助数组dist，用于表示当前找到的从源点到终点的最短路径的长度。不要求代码。

### 8.5**活动网络**

活动网络的拓扑排序算法：它用来检测是否有有向环。每次在网络中选择一个没有直接前驱的顶点并输出，从图中删去该顶点并同时删去所有它发出的有向边。循环进行，直到无没有直接前驱的顶点结束循环。若跳出循环后图中还剩下未输出的顶点，则网络中必定存在有向环。

求解关键路径的方法：用有向边表示一个工程中的活动，边上的权值表示活动持续时间，顶点表示事件，完成整个工程所需的时间取决于从源点到汇点的最长路径长度，这条路径最长的路径就叫做关键路径。关键路径上的所有活动都是关键活动。

## 排序

### 9.1排序的相关概念

排序的时间开销：关键码比较次数KCN、数据移动次数RMN

稳定性：判定标准为关键码相同的两个对象排序后顺序不变。

内外排序：内排序是在排序期间数据对象全部存放在内存中的排序，外排序是指在排序期间全部对象个数太多，需要根据排序过程的要求不断在内外存之间移动的排序。

### 9.2插入排序

直接插入排序：当插入第i个对象时，前面的0-i-1的对象已经排好序，用关键码依次比较后找到插入位置，原来位置的对象向后瞬移。这是一种稳定的排序方式，关键码比较次数和对象移动次数与对象关键码的初始排列十分相关，有最好情况和最坏情况。

折半插入排序：结合了二分与直接插入排序的排序，平均性能高于直接插入排序，它所需要的执行时间与待排序对象的初始排列无关，仅依赖于对象的个数。在插入第i个对象时**需要经过[log2i]+1**次关键码比较。他也是一个稳定的排序方法。

链表插入排序：在每个对象的结点中增加一个链接指针数据成员link。它类似直接插入排序，只是存储排序对象的方式不同。

希尔排序：又称为最小增量排序。它取一个整数gap<n作为间隔，将全部对象分为gap个子序列，所有距离为gap的对象放在同一个子序列中，在每一个子序列中分别实行直接插入排序，然后缩小gap直到最终gap为1。这样一开始gap的值比较大而子序列中的对象比较少，排序速度比较快。随着排序的进行，gap的值逐渐变小而子序列中对象个数逐渐变多，因为直接插入排序在关键码序列已基本有序时速度较快，所以此时的排序速度依然很快。

### 8.3交换排序

交换排序的基本思想是两两比较排序对象的关键码，如果发生逆序就交换。

起泡排序：需要一个附加对象实现对象值的对换，它是一个稳定的排序方法。

快速排序的算法：快速排序就是任取待排序对象中的某个对象作为基准，按照该对象的关键码大小将整个对象序列划分为左右两个子序列，要求左侧子序列中所有对象的关键码都小于等于基准对象关键码，右侧子序列中所有对象的关键码都大于等于基准对象关键码。快速排序是递归的，需要有一个栈存放每层递归调用时的指针和参数，额外存储开销为o(log2n)。

快速排序的递归算法和用栈实现的非递归算法

当待排序关键码序列已经基本有序时，快速排序显著变慢。

快速排序的优化：若能更合理的选择基准对象，使得每次划分所得到的两个子序列中的对象个数尽可能的接近，取每个待排序对象序列的第一个对象、最后一个对象和位置接近正中的三个对象，取其关键码居中者作为基准对象。

### 8.4选择排序

第i趟排序时在后面n-i个待排序对象中选出关键码最小的对象，作为有序对象序列的第i个对象。

直接选择排序：在i到n-1中选择具有最小关键码的对象，若它不是第一个对象则与第一个对象对调。它是一种不稳定的排序方法，对象的移动次数与对象序列的初始排序有关。

锦标赛排序：首先取n个对象的关键码，得到n/2个优胜者，然后再对n/2个对象进行关键码的两两比较，直到选出一个关键码最小的对象为止。胜者树用来存这个比赛结果，胜者树的根表示最后选出的具有最小关键码的对象。输出序列时，拿出胜者树顶的对象，并在胜者树中删去该对象的所有结点，生成新的胜者树，继续重复该过程直到胜者树空。排序方法减少了很多排序时间，但是用了较多附加存储空间。它是稳定的排序方式。当在n个数据（n很大）中选出最小的 5~8 个数据时，锦标赛排序最快。锦标赛排序算法将待排序数据个数 n 补足到 2的 k 次幂2k-1<n<2k。

堆排序：先建立初始最大堆，每次向下调整。最大堆第一个对象0与n对调，再进行堆向下调整，重新建立最大堆。重复该过程，得到有序序列。堆排序是不稳定的排序方式。在堆排序中将待排序的数据组织成完全二叉树的顺序存储。

### 8.5归并排序

用事例表明二路归并排序的过程

二路归并排序的非递归算法

该算法的性能分析

归并排序可以递归执行

归并排序需要较多的附加存储。

归并排序对待排序关键码的初始排列不敏感，故排序速度较稳定。

### 8.6基数排序

多关键码，需要用到栈

### 8.7外排序

多路平衡归并排序的过程

外排序的时间分析

利用败者树进行多路平衡归并

利用置换选择方法生成不等长的初始归并段

最佳归并树的构造及WPL计算

