# 三、925 数据结构

## 1、绪论。

### （1）掌握相关的基本概念，如数据结构、逻辑结构、存储结构、数据类型、 抽象数据类型等；

#### 数据结构

数据元素相互之间存在一种或多种特定关系的集合

#### 逻辑结构

数据对象中数据元素之间的相互关系。

包括：

集合结构：数据元素除都同属于一个集合外，没有任何关系

线性结构：数据元素之间是一对一的关系

树形结构：数据元素之间是一对多的层次关系

图形结构：数据元素之间是多对多的关系

#### 物理结构

数据的逻辑结构在计算机中的存储形式

#### 存储结构

包括

顺序存储结构：

链式存储结构：

#### 数据类型

一组性质相同的值的集合及定义在此集合上的一些操作的总称。

#### 抽象数据类型 ADB

一个数学模型及定义在该模型上的操作。

抽象数据模型体现了程序设计中的**问题分解、抽象和信息隐藏**的特性。

### （2）掌握算法设计的原则，掌握计算语句频度和估算算法时间复杂度和空间复杂度的方法；

#### 算法

算法是解决特定问题步骤的描述，在计算机中为指令的有限序列，每条指令执行一或多操作

#### 算法特性

输入、输出、有穷、确定、可行

#### 算法设计要求

正确性、可读性、健壮性、时间效率高和存储量低

#### 时间复杂度

在算法分析时，语句执行次数T(n) 是关于问题规模n的函数，

进而分析T(n)随n的变化并确定T(n)的数量级。

记作：T(n)=O(f(n))

时间复杂度大小

O(1) < O(logn) < O(n) <O(nlogn) < O(n^2) < O(n^3) < O(2^n) <O(n!) < O(n^n)

#### 空间复杂度

### （3）了解使用类 C 语言描述算法的方法。

## 2、线性表。

0个或多个数据元素的有限序列

### （1）掌握线性表的逻辑结构和存储结构；

存储结构：

顺序存储：

链式存储：

### （2）掌握线性表在顺序结构和链 式结构上实现基本操作的方法；

创建链表

在某个位置上插入元素

在某个元素后插入元素

删除某个位置上的元素

删除某个元素

清空链表

### **（3）理解线性表两种存储结构的不同特点及其适用场合， 会针对需求选用合适的存储结构解决实际问题；**

#### 单链表

采用链式存储结构，不要求空间连续，元素个数不限制，插入和删除O(1)，查找O(n)

#### 顺序表

采用一段连续的存储空间存储，要预先分配空间，大小确定，元素个数有限，

查找O(1)，插入和删除O(n)

### （4）了解一元多项式的表示方法和基本运算 的实现方法。

## 3、栈和队列。

### （1）了解栈和队列的特点；

栈：后进先出LIFO

队列：先进先出FIFO

### （2）掌握在两种存储结构上栈的基本操作的 实现；

### （3）掌握栈的各种应用，理解递归算法执行过程中栈状态的变化过程；

#### 后缀表示法

#### 前缀表示法

### （4）掌握循环 队列和链队列的基本运算；

### （5）会应用队列结构解决实际问题。

## 4、串。

### （1）掌握串的基本运算的定义，了解利用基本运算来实现串的其它运算的方法；

串：由0个或多个字符组成的有限序列，又名字符串。

一般机位s=”a1a2a3…an”;

串中元素只由一个字符串组成，相邻元素具有前驱或后继关系

### （2）了解在顺序存储结构和在堆存储结构以及块链存储结构上实现串的各种操作的方法；

### （3）理解 KMP 算法，掌握 NEXT 函数和改进 NEXT 函数的定义和计算。

#### next 数组推导

##### str =’abcdex’

j=1,next[1]=0;

j=2,j到j-1 只有a，next[2]=1;

j=3,j到j-1 为 ab ，next[3]=1;

……

j=6,j到j-1为 abcde,next[6]=1;

##### str=’ abcabx’

j=1,next[1]=0;

j=2,j到j-1 为ab,next[2]=1;

j=3,j到j-1 为 aba,next[3]=2;

j=4,j到j-1 为abac,next[4]=1;

j=5,j到j-1 为abcab,next[5]=3

j=6,next[6]=1;

##### str=’ababaaaba’

j=1,next[1]=0;

j=2,j到 j-1 为a,next[2]=1;

j=3,j to j-1 is ab, next[3]=1;

j=4,j to j-1 is aba,next[4]=2;

j=5,j to j-1 is abab,next[5]=3;

j=6,j to j-1 is ababa,next[6]=4;

j=7,j to j-1 is ababaa,next[7]=2;

j=8,t to j-1 is ababaaa,next[8]=2;

j=9,j to j-1 is ababaaab,next[9]=3;

#### nextval数组推导

##### str= ababaaaba’

a b a b a a a b a

0 1 1 2 3 4 2 2 3

next[]={ 0 1 1 2 3 4 2 2 3}

0 1 0 1 0 4 2 1 0

nextval[]={ 0 1 0 1 0 4 2 1 0}

j=1,nextval[1]=0;

j=2,next[2]=1,str[2]=b,与第一位是a不相等，->nextval[2]=1;

j=3,next[3]=1,str[3]=a,与第一位的a相等，->nextval[3]=0;

j=4,next[4]=2,str[4]=b,与第二位的b相等，->nextval[4]=1;

j=5,next[5]=3,str[5]=a,与第三位的a相等，->nextval[5]=0;

j=6,next[6]=4,str[6]=a,与第四位的b不相等，->nextval[6]=4;

j=7,next[7]=2,str[7]=a,与第二位的b不相等，->nextval[7]=2;

j=8,next[8]=2,str[8]=b,与第二位的b相等，->nextval[8]=1;

## 5、数组和广义表。

### （1）掌握数组在以行为主和以列为主的存储结构中的地址计算方法；

### （2）掌握矩阵压缩存储时的下标变换方法，了解以三元组表示稀疏矩阵的方法；

#### 压缩矩阵

##### 对称矩阵下标对应关系

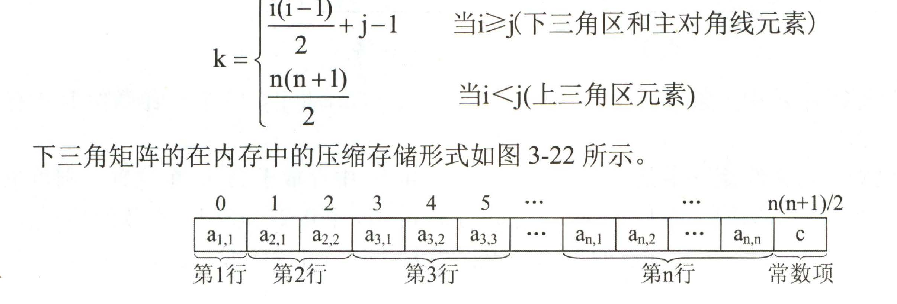
m=max(i,j)

n=min(i,j)

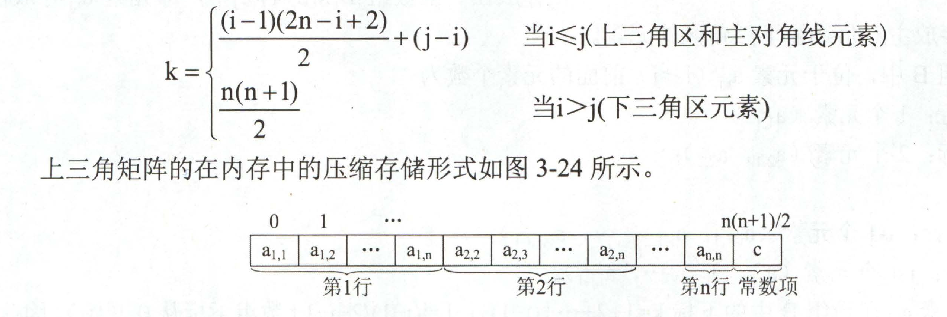
下标k=[ m\*(m-1)/2+n-1];

##### 三角矩阵

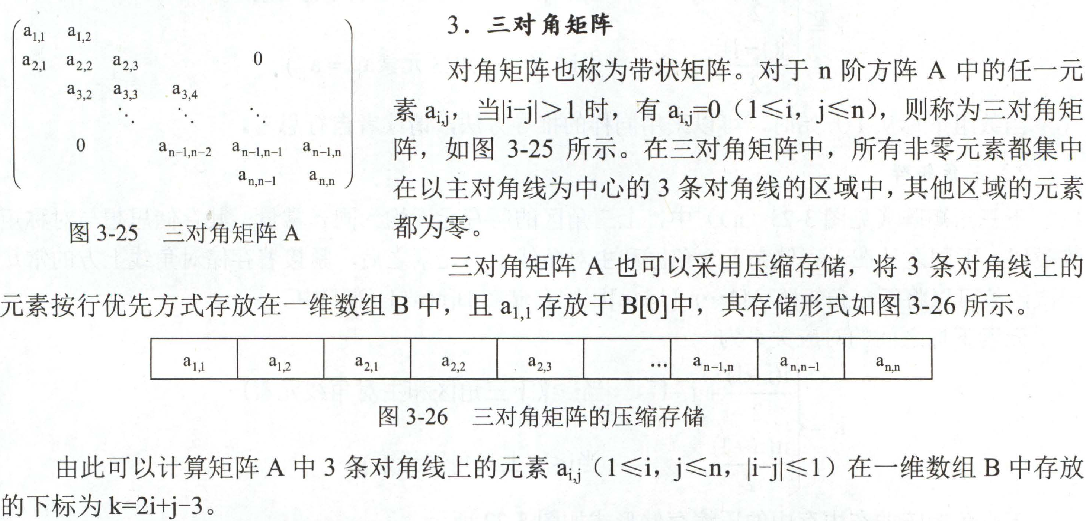
###### 下三角



###### 上三角



###### 三对角



#### 稀疏矩阵

m\*n的矩阵，t个非0元素

##### 稀疏因子

=t/(m\*n);

三元组表示法

(i,j,e); 对应行列坐标和元素值。

### （3）理解 广义表的定义及其存储结构，理解广义表的头尾和子表两种分析方法。

LS=(a1,a2,a3,a4,a5…an)

其中 ai 可以是元素（称为原子），也可以是子表。

第一个元素称为表头，其余的称为表尾

## 6、树和二叉树。

结点的度：结点的子树数量

页结点：度为0的结点，

树的度：树中结点的度的最大值

树的深度：树中结点的最大层次

### （0）二叉树

#### 特点：

每个结点最多两颗子树，二叉树中不存在度大于2的结点

严格区分左子树和右子树

#### 基本形态（5种）

空二叉树

只有一个根结点

根结点只有左树

根结点只有右树

根结点既有左树又有右树

#### 满二叉树

叶结点出现在最下层，

除叶结点外，所有的结点度都是2

#### 完全二叉树

叶节点只在最下面两层出现；

最下层的叶结点都在最左边；

倒数第二层的叶结点在右边

#### 性质

1. 二叉树的第i层最多2^( i-1)个结点
2. 深度为k的二叉树，最多2^k -1个结点
3. 任何二叉树，叶结点数目n0,度为二的结点数目n2，则n0=n2+1
4. n个结点的二叉树的深度为log2 n+1；
5. 对于一颗n个结点的完全二叉树进行标号，
   1. i=1，结点i为根节点；i>1 ,i/2为父节点；
   2. 2i>n，则结点i无左结点；否则其左子结点为2i；
   3. 2i+1>n，则节点i无右节点，否则其右子节点为2i+1；

### （1）熟练掌握二叉树的结构特点和性质，掌握二叉树各种存储结构及 构建方法；

已知前中序遍历可以唯一确定二叉树结构

已知中后序遍历可以唯一确定二叉树结构；

已知前后序遍历**不可以**唯一确定二叉树结构；

结论：

要确定一个二叉树的结构，必须知道中序遍历的顺序，和前后任意一个遍历顺序

### （2）掌握按先序、中序、后序和层次次序遍历二叉树的算法，理解二叉树的线索 化实质和方法

#### 线索二叉树

在一般二叉树T中，

若T有左子节点，则left指向左子节点，

否则，left指向其前驱节点；

若T有右子节点，则right指向其右子节点，

否则，right指向其后继节点。

为了区分左右指针指向的是子节点还是前驱或后继结点，

node节点添加两个标志lbit,rbit,

值为0表示指向子节点，

值为1表示指向前驱或者后继

### （3）利用二叉树的遍历求解实际问题；

### （3）掌握树的各种存储结构及其特点， 掌握树的各种运算的实现算法；

双亲表示法

### （4）掌握建立最优二叉树和哈夫曼编码的方法。

#### 哈夫曼树，又称最优二叉树

哈夫曼树中不存在度为1的节点。

设二叉树有m个结点，每个叶结点赋予一个权值，

带权路径长度WPL = 叶节点的权值\*(所在层次-1) ，再求和。

##### 构造哈夫曼树

##### 哈夫曼算法思想

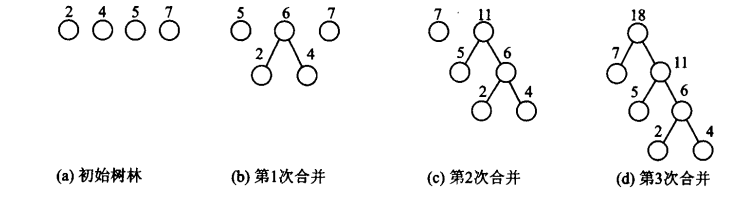
1. 将给定权值从小到大排列(w1,w2,w3…wm)，并构造森林F=(t1,t2…tm),

此时，其中每个树ti的左右节点都为null，根节点的权值为wi；

1. 把F中树根节点的权值最小的两颗二叉树t1,t2合并成一颗新的二叉树t，

t的权值为w1+w2，把t按权值顺序放入到F中，删除t1，t2

1. 重复步骤2



##### 哈夫曼编码

## 7、图。

### （1）熟练掌握图的基本概念，会构建各种图的存储结构；

图：是由顶点的有穷非空集合和顶点之间边的集合组成 表示为G(V,E)，V是顶点的集合，E是边的集合

无向边：顶点v1和v2之间的边没有方向，用无序偶对（vi,vj）表示。

有向边：顶点v1和v2之间的边有方向，也称弧，用有序偶对<vi,vj>表示。

无向完全图：无向图中，任意两个顶点之间都存在边，

n个顶点的无向完全图有[n\*(n-1)]/2 条边。

有向完全图：有向图中，任意两个顶点之间存在有方向的边，

n个顶点的有向完全图有n\*(n-1)条边。

稀疏图：有很少条边的图。

稠密图：有很多条边的图。

权：图的边或弧上相关的数据。

网：带权的图。

子图：图中的一部分。

邻接点：同一条边的两个顶点。

顶点的度：与顶点有关的边的数量 ，记为TD(v)。

入度：对于有向图，终止与该顶点边的数量，(其他顶点指向该顶点的边的数量) 。

出度：对于有向图，该顶点指向其他顶点的边的数量。

路径的长度：路径上边或弧的数量。

回路（环）：从第一个顶点到最后一个顶点相同的路径。

连通图：顶点v1到顶点v2之间有路径。

联通分量：无向图中的极大联通子图。

强连通图：图中每对顶点之间都存在路径。

连通图生成树：一个极小的连通子图，包含全部n个顶点，但只有n-1条边。

邻接矩阵：使用一维数组存顶点，二维数组存边(弧)的关系。

对于无向图：二维数组中不为0的行（列）的个数为顶点的度。

对于有向图：二维数组中 行出列入。

邻接表：使用一维数组存顶点，每个顶点vi的所有邻接点构成一个线性表，使用单列表存储

### （2）掌握深度优先搜索遍历图和广度优先搜索遍历图的算法；

dfs

从一个顶点出发，访问完整个链上的所有顶点；

如果是连通图，一次就完成遍历；

如果不是联通图，在从另一个没被访问的顶点出发，重复执行。

bfs：

从一个顶点出发，访问完该顶点多有相关的顶点后，

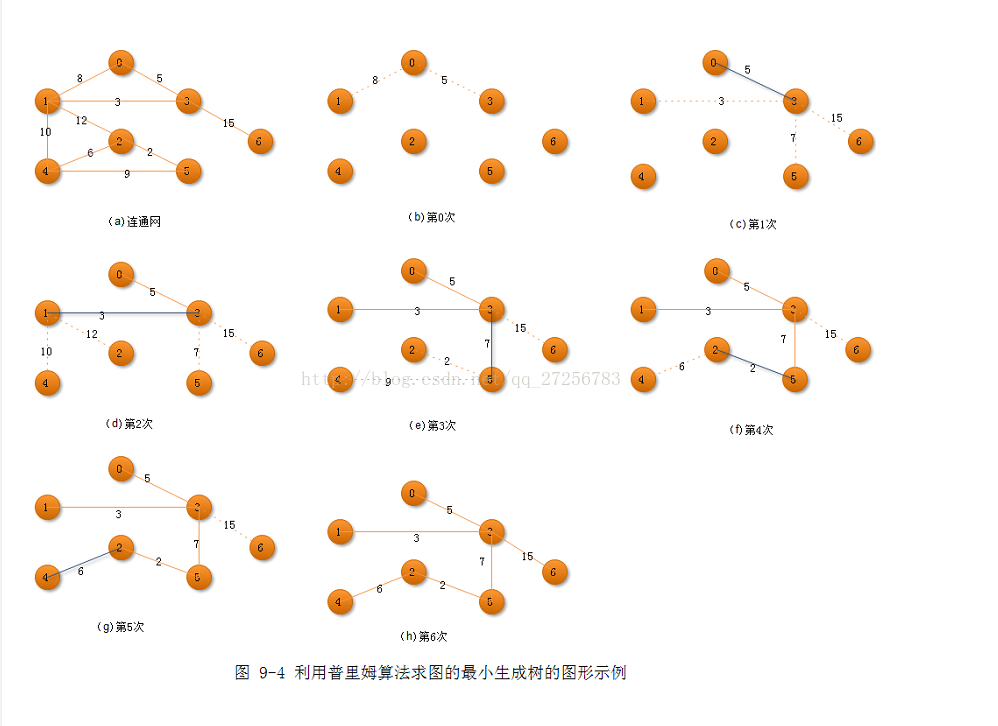
再从另一个顶点出发，重复执行。

### （3）灵活运用图的遍历算法求解各种路径问题，包 括最小生成树﹑最短路径﹑拓扑排序﹑关键路径等。

#### 最小生成树：

##### 普利姆算法：

已某个顶点为起点，找权值最小的顶点连在一起



##### 克里斯卡尔算法：

#### 最短路径：

不带权图

顶点之间经过的边最少

带权图

权值最小。第一个顶点是源点，最后一个顶点为终点。

迪杰斯拉算法：

按路径长度递增的次序产生最短路径的算法。

解决了某个源点到其他各顶点的距离。 时间复杂度O(n^2)

弗洛伊德算法

#### 关键路径

https://www.jianshu.com/p/1857ed4d8128

## 8、查找。

### （1）熟练掌握各种静态查找和动态查找算法，会计算查找成功时和失败时的 平均查找长度；

#### 静态查找表

只做查找操作的表，

查询某个特定的数据元素是否在表中；

检索某个特定的数据元素的属性

#### 动态查找表

查找过程中同时插入或者删除操作。

插入

删除

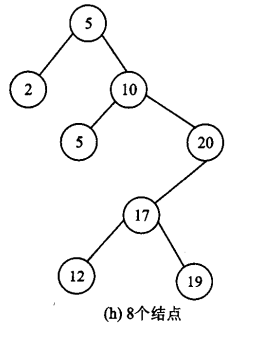
#### 二叉树查找长度

确定一个元素在树中位置所需进行比较次数的期望

##### 内路径长度

从二叉树根节点到某节点所经过的分支数目

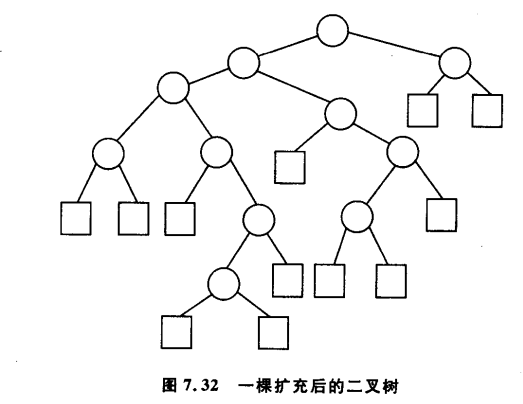
IPL= 层数\*当前层子节点的数目 ，再求和。



IPL=1\*2 + 2\*2 +3\*1 +4\*2=17

##### 外路径长度

为了分析查找失败时的查找长度，把二叉树空的结点补全



n=11

IPL =1\*2 + 2\*2+ 3\*3 +4\*2 +5\*1=26

EPL=2\*2 + 3\*1 +4\*4 +5\*3 +6\*2=50

EPL=IPL+2n

##### 平均查找长度

查找成功时：ASL=(IPL+n)/n

查找失败时：ASL=EPL/n=(IPL+2n)/n

平均： ASL=( IPL+EPL+n)/(n+n+1)=(2IPL+3n)/(2n+1)

#### 顺序查找

又称线性查找，从表中的第一个数据开始逐个比较查找

最好的情况第一个位置找到，O(1)；

最差的情况最后一个位置找到，O(n);

找不到的话：O(n+1)

平均查找次数：(n+1)/2

时间复杂度：O(n)

#### 有序表查找

1. 折半查找
2. 插值查找
3. 斐波那契查找

#### 线性索引查找

1. 稠密索引

在线性索引中，将数据集的每个记录对应一个索引项。

对于稠密索引表，关键字是按顺序排列的

1. 分块索引

把数据集的记录分成若干块，

块内无序：数据块内元素不要求有序

块间有序：每个数据块看作整体，各个数据块是有序的

分块的索引:

最大关键字：记录块中最大的关键字

存储了块中记录的个数。

存储了指向块中首个元素的指针。

查找步骤：

在分块索引表中找到指定数据关键字的块，

在块中按关键字查找指定数据。

平均查找长度：

设n个记录分成m块，每个块中有t条记录。

显然n=m\*t；

设L1为查找索引表的平均查找长度，L1的平均长度为（m+1）/2

设L2为块内的平均查找长度，L2=（t+1）/2

分块查找的平均查找长度为：

ASW=L1+L2 =(m+1)/2+(t+1)/2

=(m+t)/2+1

=(n/t+t)/2+1

最佳情况是分的块数m与块中的记录数t相同

即n=m\*t=t^2;

ASW=(n/t+t)/2+1

=t+1

=根号n+1

1. 倒序索引

索引项的通用结构：

次关键字：

记录号表：真实记录对应的位置等等。

### （2）掌握二叉排序树的建立、插入和删除过程，掌握二叉平衡树的建立和旋转平衡方法；

#### 二叉排序树

#### 平衡二叉树

定义：

是一种二叉排序树，其中每一个结点的左子树和右子树的高度差至多是1.

平衡因子：

二叉树结点上左子树深度减去右子树深度的值。

平衡因子的值只能是 1 ，0，-1。

实现原理:

在构造二叉排序树时，每当插入一个结点时，先检查是否破坏了树的平衡性，若是，找出最小不平衡子树。在保持二叉排序树的前提下，调整最小不平衡子树中各结点之间的关系，进行相应的旋转，使之成为二叉平衡树。

### （3）掌握 B-树的建立、插入和删除结点的过程；

#### 多路查找树

每一个结点的孩子数可以多于两个，且每一个结点可以储存多个元素

##### 2-3树：

每一个结点有两个孩子或者三个孩子。

一个2结点包含一个元素和两个孩子(或者没有孩子)，数据排列与二叉树相似，左小右大，但是这两个节点要么没有孩子，要么两个孩子都在，不能只有一个；

一个3结点包含一小一大两个元素和三个孩子(或没有)；

2-3树所有叶结点都在同一层上；

##### 2-3-4树

上层结点中最多三个元素，插入元素时，将结点中小于和大于被插入元素的元素分组。

##### B树

是一种平衡的多路查找树。，2-3，2-3-4树是它的特例。

结点最大的孩子数目称为B树的阶；

因此，2-3树是3阶B树，

2-3-4树是4阶B树。

###### 一个m阶的B树具有如下性质

1. 如果根节点不是叶节点，则至少有两颗子树；
2. 所有的叶节点在同一层；
3. 每一个非根的分支结点都有k-1个元素和k个孩子，其中 2/m<=k<=m,

每一个叶子节点n都有k-1个元素

###### n个结点的m阶B树查找分析

第一层至少1个结点，第二层至少2个结点，

由于除根结点外，每个分支结点至少有m/2个子树，

第三层至少有2\*(m/2)个结点，

这样第k+1层 至少有2\*[(m/2)^k-1],而k+1层就是叶结点

#### B+树

在B树中，每一个元素在该树种只出现一次。

在B+树中，出现在分支结点中的元素会被当做它们在该分支结点位置的中序后继者中再次出现。此外，每一个叶子结点都会保存一个指向后一个叶子结点的指针。

##### 对于m阶B+树特点

1、有n个子树的结点包含有n个关键字；

2、所有的叶节点包含全部关键字的信息，及指向含这些关键字记录的指针，

叶子结点本身依照关键字的大小自然排序。

3、所有分支结点可以看成是索引，结点中仅含有最大(小)的关键字

### （4）熟练掌握哈希表的构造方法和处理冲突的方法。

#### 哈希表查找

哈希函数 p=f(x)

散列技术是记录的存储位置和它的关键字之间的函数关系。

#### 散列函数构造方法

##### 构造原则

1. 计算简单
2. 分布均匀

##### 直接定址法

直接取关键字的某个线性函数值作为散列地址。

f(key)=a\*key+b a,b为常数

特点是简单、均匀，一般不会产生冲突，但需要事先知道关键字的分布情况，适合数据量少且连续的情况

##### 数字分析法

使用关键字的一部分来计算散列存储位置。

适合处理关键字位数较大的数字，需要事先知道关键字的若干位分布均匀的话，可以采用

##### 平方取中法

将关键字平方后，在平方结果中取若干位作为散列地址。

##### 折叠法

将关键字从左到右分割成若干等份，对这几部分求和，取求和的结果的若干位作为散列地址。

折叠法事先不需要知道关键字的分布，适合关键字位数较多的情况。

##### 除留余数法

最为常用的构造方法。

构造函数方法：f(key)=key mod p

若散列表长度为m，p通常为小于或等于m的最小质数或不好含20质因子的合数。

##### 随机数法

选择一个随机数作为散列地址。

f(key)=random(key)。

#### 构造散列函数考虑的因素

1. 计算散列地址所需的时间；
2. 关键字的长度；
3. 散列表的大小；
4. 关键字的分布情况；
5. 记录查找的频率

#### 处理哈希冲突的方法

##### 开放定址法

一旦发生冲突，就去寻找下一个空的散列地址

f(key) = (f(key)+d) mod m

这种方法称为线性探测法

缺点：当关键字分布均匀的时候容易产生堆积

##### 再散列函数法

需要多准备几个散列函数，当发生冲突时，使用下一个散列函数重新计算散列地址。

这种方法不会产生关键字堆积，但是会增加计算的时间。

##### 链地址法

使用桶链的方式存储，参考java的map集合

##### 公共溢出区法

建立一个公共溢出区，将产生冲突的元素都放到这个区域中，

## 9、排序。

#### 基本概念

内排序

待排序的数据放在内存中

外排序

由于数据巨大，待排序的数据不能同时放在内存中，在排序过程中，需要和外存进行数据交换

影响排序算法的性能因素：

1. 时间性能
2. 辅助空间
3. 算法的复杂性

### （1）掌握各种排序算法，包括插入类、交换类、选择类、归并类排序及基数 排序；

#### 冒泡排序

##### 算法思想

两两比较相邻记录的关键字，如果反序则进行交换，直到没有反序的记录为止。

##### 算法描述

两层循环，

外层循环从0开始，内层循环从外层的位置循环，(即内层参与循环的元素始终都是在外层当前循环元素的后面的元素)，当外层循环的值大于内层循环的值时，说明是反序(约定小->大为正序)，需要进行交换。

##### 时间复杂度 O(n^2)

#### 简单选择排序

##### 算法思想

两层循环，内层循环在 外层循环的当前元素后一个开始，设置一个标志记录外层循环当前的值，在内层循环中，如果有比这个标志小的，则修改标志。

内层循环执行一遍后，验证标志是否改变，是则交换元素。

##### 时间复杂度O(n^2)

#### 直接插入排序

##### 算法思想

把n个待排序元素看成是一个有序表和一个无序表，开始时，有序表长度为1，无序表长度为n-1，每次排序过程中从无序表中取一个元素插入到有序表中，循环执行。

设置一个标志，两层循环，外层循环从第二个元素开始，循环中进行判断，如果后项>前向，

标志记录后项，开始内层反向循环，从标志项的前一项开始循环，将每一项向后移动一位，

内层循环结束后，将标志项放到正确的位置。

##### 时间复杂度O(n^2) (较插入和冒泡高些)

#### 希尔排序

##### 算法思想

跳跃式排序，选取一个gap=n，则每次比较a[0]与a[0+n],a[1]与a[1+n]，每循环一次，修改减少gap的值，直至gap=1

##### 时间复杂度O(n^3/2)

#### 堆排序

是对简单选择排序的一种优化。

堆是具有以下性质的完全二叉树：

每个结点的值都大于或等于其左右结点的值称为大顶堆；

k\_i >= k\_2i

k\_i >= k\_2i+1

每个结点的值都小于或等于其左右结点的值称为小顶堆。

k\_i <= k\_2i

k\_i <= k\_2i+1

其中1 < i < n/2

##### 算法思想

利用堆结构进行排序。

将待排序数组构造成一个大顶堆积，此时整个序列的最大值就是堆顶的根节点，将它与末尾元素交换，在把前面的元素构造成一个堆，重复执行。

##### 时间复杂度

运行时间主要消耗在初始堆和重构堆上面，

其中构建堆的时间复杂度O(n);

重构堆的时间复杂度为O(nlogn)；

因此总的时间复杂度是**O(nlogn)**

#### 归并排序

##### 算法思想

利用归并的思想实现排序

原理：

假设初始序列有n个记录，则可以看成是有n个有序的子序列，每个序列的长度为1，然后两两归并，得到ceil（n/2）个长度为2或1的子序列，重复执行，直至有一个长度为n的序列，这种方法成为2路归并排序。

##### 时间复杂度

#### 快速排序

##### 算法思想

是冒泡排序的一种改进。

通过一趟排序，将待排序的序列分成两部分，其中一部分的关键字均比另一部分小。对着两部分重复执行

##### 时间复杂度

### （2）能够对各种排序方法进行比较分析，如稳定性、时间和空间性能等，了解各种排 序方法的特点和不同并灵活应用；

#### 稳定性比较

插入、冒泡、归并、基数是稳定性排序，

选择、希尔、快速、堆积是非稳定性排序。

#### 复杂性比较

1. 平均情况下，希尔、快排、堆积、归并能达到较快的排序速度，其中快排最快，

从空间看，堆积排序最节省空间

1. 平均情况下，插入、冒泡排序较慢，但当待排序数列开始就局部有序，这两种算法可以达到较快的速度



#### 特点

1. 数据规模较大，不要求稳定性，宜使用快排，
2. 数据规模较大，要求稳定性，空间充足，宜使用归并
3. 数据规模较大，不要求稳定性，元素部分有序，宜使用堆积、归并
4. 数据规模较小，要求稳定性，元素基本有序，宜使用插入
5. 数据规模较小，不要求稳定性，宜使用选择

### （3）理解外部排序的主要思想和过程。