数据结构知识点概括

第一章 概 论

数据就是指能够被计算机识别、存储和加工处理的信息的载体。

数据元素是数据的基本单位，可以由若干个数据项组成。数据项是具有独立含义的最小标识单位。

数据结构的定义：

• 逻辑结构：从逻辑结构上描述数据，独立于计算机。• 线性结构：一对一关系。

• 线性结构：多对多关系。

• 存储结构：是逻辑结构用计算机语言的实现。• 顺序存储结构：如数组。

• 链式存储结构：如链表。

• 索引存储结构：• 稠密索引：每个结点都有索引项。

• 稀疏索引：每组结点都有索引项。

• 散列存储结构：如散列表。

• 数据运算。

• 对数据的操作。定义在逻辑结构上，每种逻辑结构都有一个运算集合。

• 常用的有：检索、插入、删除、更新、排序。

数据类型：是一个值的集合以及在这些值上定义的一组操作的总称。

• 结构类型：由用户借助于描述机制定义，是导出类型。

抽象数据类型 ADT：• 是抽象数据的组织和与之的操作。相当于在概念层上描述问题。

• 优点是将数据和操作封装在一起实现了信息隐藏。

程序设计的实质是对实际问题选择一种好的数据结构，设计一个好的算法。算法取决于数据结构。

算法是一个良定义的计算过程，以一个或多个值输入，并以一个或多个值输出。

评价算法的好坏的因素：• 算法是正确的；

• 执行算法的时间；

• 执行算法的存储空间（主要是辅助存储空间）；

• 算法易于理解、编码、调试。

时间复杂度：是某个算法的时间耗费，它是该算法所求解问题规模 n 的函数。

渐近时间复杂度：是指当问题规模趋向无穷大时，该算法时间复杂度的数量级。

评价一个算法的时间性能时，主要标准就是算法的渐近时间复杂度。

算法中语句的频度不仅与问题规模有关，还与输入实例中各元素的取值相关。

时间复杂度按数量级递增排列依次为：常数阶 O（1）、对数阶 O（log2n）、线性阶 O（n）、线性对数阶 O（nlog2n）、平方阶 O（n^2）、立方阶 O（n^3）、……k 次方阶 O（n^k）、指数阶 O（2^n）。

空间复杂度：是某个算法的空间耗费，它是该算法所求解问题规模 n 的函数。

算法的时间复杂度和空间复杂度合称算法复杂度。

第二章 线性表

线性表是由 n≥0 个数据元素组成的有限序列。

n=0 是空表；非空表，只能有一个开始结点，有且只能有一个终端结点。

线性表上定义的基本运算：

• 构造空表：Initlist（L）

• 求表长：Listlength（L）

• 取结点：GetNode（L，i）

• 查找：LocateNode（L，x）

• 插入：InsertList（L，x，i）

• 删除：Delete（L，i）

顺序表是按线性表的逻辑结构次序依次存放在一组地址连续的存储单元中。在存储单元中的各元素的物理位置和

逻辑结构中各结点相邻关系是一致的。地址计算：LOCa（i）=LOCa（1）+（i-1）*d；（首地址为 1）

在顺序表中实现的基本运算：

• 插入：平均移动结点次数为 n/2；平均时间复杂度均为 O（n）。

• 删除：平均移动结点次数为（n-1）/2；平均时间复杂度均为 O（n）。

线性表的链式存储结构中结点的逻辑次序和物理次序不一定相同，为了能正确表示结点间的逻辑关系，在存储每个结点值的同时，还存储了其后继结点的地址信息（即指针或链）。这两部分信息组成链表中的结点结构。

一个单链表由头指针的名字来命名。

单链表运算：

• 建立单链表 • 头插法：s->next=head；head=s；生成的顺序与输入顺序相反。平均时间复杂度均为 O（n）。

• 尾插法：head=rear=null；if（head=null） head=s；else r->next=s；r=s； 平均时间复杂度均为 O（n）

• 加头结点的算法：对开始结点的操作无需特殊处理，统一了空表和非空表。

• 查找 • 按序号：与查找位置有关，平均时间复杂度均为 O（n）。

• 按值：与输入实例有关，平均时间复杂度均为 O（n）。

• 插入运算：p=GetNode（L，i-1）；s->next=p->next；p->next=s；平均时间复杂度均为 O（n）

• 删除运算：p=GetNode（L，i-1）；r=p->next；p->next=r->next；free（r）；平均时间复杂度均为 O（n）

单循环链表是一种首尾相接的单链表，终端结点的指针域指向开始结点或头结点。链表终止条件是以指针等于头指针或尾指针。

采用单循环链表在实用中多采用尾指针表示单循环链表。优点是查找头指针和尾指针的时间都是 O（1），不用

遍历整个链表。

双链表就是双向链表，就是在单链表的每个结点里再增加一个指向其直接前趋的指针域 prior，形成两条不同方

向的链。由头指针 head 惟一确定。

双链表也可以头尾相链接构成双（向）循环链表。

双链表上的插入和删除时间复杂度均为 O （1）。

顺序表和链表的比较： • 基于空间：

•顺序表的存储空间是静态分配，存储密度为 1；适于线性表事先确定其大小时采用。

• 链表的存储空间是动态分配，存储密度＜1；适于线性表长度变化大时采用。

• 基于时间：

• 顺序表是随机存储结构，当线性表的操作主要是查找时，宜采用。

• 以插入和删除操作为主的线性表宜采用链表做存储结构。

• 若插入和删除主要发生在表的首尾两端，则宜采用尾指针表示的单循环链表。

第三章 栈和队列

栈（Stack）是仅限制在表的一端进行插入和删除运算的线性表，称插入、删除这一端为栈顶，另一端称为栈底。表中无元素时为空栈。栈的修改是按后进先出的原则进行的，我们又称栈为 LIFO 表（Last In First Out）。通常栈有

顺序栈和链栈两种存储结构。

栈的基本运算有六种： • 构造空栈：InitStack（S）

• 判栈空： StackEmpty（S）

• 判栈满： StackFull（S）

• 进栈： Push（S，x）

• 退栈： Pop（S）

• 取栈顶元素：StackTop（S）

在顺序栈中有“上溢”和“下溢”的现象。 •“上溢”是栈顶指针指出栈的外面是出错状态。

•“下溢”可以表示栈为空栈，因此用来作为控制转移的条件。

顺序栈中的基本操作有六种：•构造空栈 •判栈空 •判栈满 •进栈 •退栈 •取栈顶元素

链栈则没有上溢的限制，因此进栈不要判栈满。链栈不需要在头部附加头结点，只要有链表的头指针就可以了。

链栈中的基本操作有五种：• 构造空栈 • 判栈空 • 进栈 • 退栈 • 取栈顶元素

队列（Queue）是一种运算受限的线性表，插入在表的一端进行，而删除在表的另一端进行，允许删除的一端称

为队头（front），允许插入的一端称为队尾（rear） ，队列的操作原则是先进先出的，又称作 FIFO 表（First In

First Out） .队列也有顺序存储和链式存储两种存储结构。

队列的基本运算有六种： • 置空队：InitQueue（Q）

• 判队空：QueueEmpty（Q）

• 判队满：QueueFull（Q）

• 入队：EnQueue（Q，x）

• 出队：DeQueue（Q）

• 取队头元素：QueueFront（Q）

顺序队列的“假上溢”现象：由于头尾指针不断前移，超出向量空间。这时整个向量空间及队列是空的却产生了“上
溢”现象。

为了克服“假上溢”现象引入循环向量的概念，是把向量空间形成一个头尾相接的环形，这时队列称循环队列。

判定循环队列是空还是满，方法有三种：

• 一种是另设一个布尔变量来判断；

• 第二种是少用一个元素空间，入队时先测试（（rear+1）%m = front）？ 满：空；

• 第三种就是用一个计数器记录队列中的元素的总数。

队列的链式存储结构称为链队列，一个链队列就是一个操作受限的单链表。为了便于在表尾进行插入（入队）的
操作，在表尾增加一个尾指针，一个链队列就由一个头指针和一个尾指针唯一地确定。链队列不存在队满和上溢
的问题。在链队列的出队算法中，要注意当原队中只有一个结点时，出队后要同进修改头尾指针并使队列变空。

第四章 串

串是零个或多个字符组成的有限序列。

• 空串：是指长度为零的串，也就是串中不包含任何字符（结点）。

• 空白串：指串中包含一个或多个空格字符的串。

•在一个串中任意个连续字符组成的子序列称为该串的子串，包含子串的串就称为主串。

• 子串在主串中的序号就是指子串在主串中首次出现的位置。

• 空串是任意串的子串，任意串是自身的子串。

串分为两种： • 串常量在程序中只能引用不能改变；

• 串变量的值可以改变。

串的基本运算有： • 求串长 strlen（char*s）

• 串复制 strcpy（char*to，char*from）

• 串联接 strcat（char*to，char*from）

• 串比较 charcmp（char*s1，char*s2）

• 字符定位 strchr（char*s，charc）

串是特殊的线性表（结点是字符），所以串的存储结构与线性表的存储结构类似。串的顺序存储结构简称为顺序串。

顺序串又可按存储分配的不同分为：

• 静态存储分配：直接用定长的字符数组来定义。优点是涉及串长的操作速度 快，但不适合插入、链接操作。

• 动态存储分配：是在定义串时不分配存储空间，需要使用时按所需串的长度分配存储单元。

串的链式存储就是用单链表的方式存储串值，串的这种链式存储结构简称为链串。链串与单链表的差异只是它的 结
点数据域为单个字符。

为了解决“存储密度”低的状况，可以让一个结点存储多个字符，即结点的大小。

顺序串上子串定位的运算：又称串的“模式匹配”或“串匹配”，是在主串中查找出子串出现的位置。在串匹配中，将主串称为目标（串），子串称为模式（串）。这是比较容易理解的，串匹配问题就是找出给定模式串 P 在给定目标串 T 中首次出现的有效位移或者是全部有效位移。最坏的情况下时间复杂度是 O（（n-m+1）m），假如 m 与 n 同阶
的话则它是 O（n^2）。链串上的子串定位运算位移是结点地址而不是整数

第五章 多维数组

数组一般用顺序存储的方式表示。

存储的方式有： • 行优先顺序，也就是把数组逐行依次排列。PASCAL、C

• 列优先顺序，就是把数组逐列依次排列。FORTRAN

地址的计算方法： •按行优先顺序排列的数组：LOCa（ij）=LOCa（11）+（（i-1）*n+（j-1））*d.

•按列优先顺序排列的数组：LOCa（ij）=LOCa（11）+（（j-1）*n+（i-1））*d.

矩阵的压缩存储：为多个相同的非零元素分配一个存储空间；对零元素不分配空间。

特殊矩阵的概念：所谓特殊矩阵是指非零元素或零元素分布有一定规律的矩阵。

稀疏矩阵的概念：一个矩阵中若其非零元素的个数远远小于零元素的个数，则该矩阵称为稀疏矩阵。

特殊矩阵的类型： •对称矩阵：满足 a（ij）=a（ji）。元素总数 n（n+1）/2.I=max（i，j），J=min（i，j），LOCa（ij）=LOC（sa[0]）+（I*（I+1）/2+J）*d.

• 三角矩阵： • 上三角阵：k=i*（2n-i+1）/2+j-i，LOCa（ij）=LOC（sa[0]）+k*d.

• 下三角阵：k=i*（i+1）/2+j，LOCa（ij）=LOC（sa[0]）+k*d.

• 对角矩阵：k=2i+j，LOCa（ij）=LOC（sa[0]）+k*d.

稀疏矩阵的压缩存储方式用三元组表把非零元素的值和它所在的行号列号做为一个结点存放在一起，用这些结点组成的一个线性表来表示。但这种压缩存储方式将失去随机存储功能。加入行表记录每行的非零元素在三元组表中的
起始位置，即带行表的三元组表。

第六章 树

树是 n 个结点的有限集合，非空时必须满足：只有一个称为根的结点；其余结点形成 m 个不相交的子集，并称
根的子树。

根是开始结点；结点的子树数称度；度为 0 的结点称叶子（终端结点）；度不为 0 的结点称分支结点（非终端结
点）；除根外的分支结点称内部结点；

有序树是子树有左，右之分的树；无序树是子树没有左，右之分的树；森林是 m 个互不相交的树的集合；

树的四种不同表示方法：• 树形表示法；• 嵌套集合表示法；• 凹入表示法 • 广义表表示法。

二叉树的定义：是 n≥0 个结点的有限集，它是空集（n=0）或由一个根结点及两棵互不相交的分别称作这个根的
左子树和右子树的二叉树组成。

二叉树不是树的特殊情形，与度数为 2 的有序树不同。

二叉树的 4 个重要性质： •二叉树上第 i 层上的结点数目最多为 2^（i-1）（i≥1）。；

• 深度为 k 的二叉树至多有（2^k）-1 个结点（k≥1）；

•在任意一棵二叉树中，若终端结点的个数为 n0，度为 2 的结点数为 n2，则 n0=n2+1；

• 具有 n 个结点的完全二叉树的深度为 int（log2n）+1.

满二叉树是一棵深度为 k，结点数为（2^k）-1 的二叉树；完全二叉树是满二叉树在最下层自右向左去处部分结点；

二叉树的顺序存储结构就是把二叉树的所有结点按照层次顺序存储到连续的存储单元中。（存储前先将其画成完全
二叉树）

树的存储结构多用的是链式存储。BinTNode 的结构为 lchild|data|rchild，把所有 BinTNode 类型的结点，加上一个指向根结点的 BinTree 型头指针就构成了二叉树的链式存储结构，称为二叉链表。它就是由根指针 root 唯一确定的。
共有 2n 个指针域，n+1 个空指针。
根据访问结点的次序不同可得三种遍历：先序遍历（前序遍历或先根遍历），中序遍历（或中根遍历）、后序遍历（或
后根遍历）。时间复杂度为 O（n）。
利用二叉链表中的 n+1 个空指针域来存放指向某种遍历次序下的前趋结点和后继结点的指针，这些附加的指针就称为“线索”，加上线索的二叉链表就称为线索链表。线索使得查找中序前趋和中序后继变得简单有效，但对于查找指定结
点的前序前趋和后序后继并没有什么作用。
树和森林及二叉树的转换是唯一对应的。
转换方法： • 树变二叉树：兄弟相连，保留长子的连线。

• 二叉树变树：结点的右孩子与其双亲连。

• 森林变二叉树：树变二叉树，各个树的根相连。

树的存储结构：• 有双亲链表表示法：结点 data | parent，对于求指定结点的双亲或祖先十分方便，但不适于求指定结
点的孩子及后代。

•孩子链表表示法：为树中每个结点 data | next 设置一个孩子链表 firstchild，并将 data | firstchild 存放在一个向量中。

• 双亲孩子链表表示法：将双亲链表和孩子链表结合。

• 孩子兄弟链表表示法：结点结构 leftmostchild |data | rightsibing，附加两个分别指向该结点的最左孩子和右邻兄弟的
指针域。

树的前序遍历与相对应的二叉树的前序遍历一致；树的后序遍历与相对应的二叉树的中序遍历一致。

树的带权路径长度是树中所有叶结点的带权路径长度之和。树的带权路径长度最小

的二叉树就称为最优二叉树
（即哈夫曼树）。

在叶子的权值相同的二叉树中，完全二叉树的路径长度最短。

哈夫曼树有 n 个叶结点，共有 2n-1 个结点，没有度为 1 的结点，这类树又称为严格二叉树。

变长编码技术可以使频度高的字符编码短，而频度低的字符编码长，但是变长编码可能使解码产生二义性。如 00、01、0001 这三个码无法在解码时确定是哪一个，所以要求在字符编码时任一字符的编码都不是其他字符编码的
前缀，这种码称为前缀码（其实是非前缀码）。

哈夫曼树的应用最广泛地是在编码技术上，它能够容易地求出给定字符集及其概率分布的最优前缀码。哈夫曼编码的构造很容易，只要画好了哈夫曼树，按分支情况在左路径上写代码 0，右路径上写代码 1，然后从上到下到叶结
点的相应路径上的代码的序列就是该结点的最优前缀码。

第七章 图

图的逻辑结构特征就是其结点（顶点）的前趋和后继的个数都是没有限制的，即任意两个结点之间之间都可能相关。
图 GraphG=（V，E），V 是顶点的有穷非空集合，E 是顶点偶对的有穷集。
有向图 Digraph：每条边有方向；无向图 Undigraph：每条边没有方向。
有向完全图：具有 n*（n-1）条边的有向图；无向完全图：具有 n*（n-1）/2 条边的无向图；
有根图：有一个顶点有路径到达其它顶点的有向图；简单路径：是经过顶点不同的路径；简单回路是开始和终端重
的简单路径；
网络：是带权的图。
图的存储结构：

- 邻接矩阵表示法：用一个 n 阶方阵来表示图的结构是唯一的，适合稠密图。
- 无向图：邻接矩阵是对称的。
- 有向图：行是出度，列是入度。

建立邻接矩阵算法的时间是 O（n+n^2+e），其时间复杂度为 O（n^2）

- 邻接表表示法：用顶点表和邻接表构成不是唯一的，适合稀疏图。
- 顶点表结构 vertex | firstedge，指针域存放邻接表头指针。
- 邻接表：用头指针确定。 • 无向图称边表；
- 有向图又分出边表和逆邻接表；
- 邻接表结点结构为 adjvex | next，

时间复杂度为 O（n+e）。，空间复杂度为 O（n+e）。。

图的遍历： • 深度优先遍历：借助于邻接矩阵的列。使用栈保存已访问结点。

- 广度优先遍历：借助于邻接矩阵的行。使用队列保存已访问结点。

生成树的定义：若从图的某个顶点出发，可以系统地访问到图中所有顶点，则遍历时经过的边和图的所有顶点
构成的子图称作该图的生成树。

最小生成树：图的生成树不唯一，从不同的顶点出发可得到不同的生成树，把权值最小的生成树称为最小生成树
（MST）。

构造最小生成树的算法： •Prim 算法的时间复杂度为 O（n^2）与边数无关适于稠密图。

- Kruskal 算法的时间复杂度为 O（lge），主要取决于边数，较适合于稀疏图。

最短路径的算法：• Dijkstra 算法，时间复杂度为 O（n^2）。• 类似于 prim 算法。

拓扑排序：是将有向无环图 G 中所有顶点排成一个线性序列，若<u，v>∈E（G），则在线性序列 u 在 v 之前，
这种线性序列称为拓扑序列。
拓扑排序也有两种方法：

- 无前趋的顶点优先，每次输出一个无前趋的结点并删去此结点及其出边，最后得到的序列即拓扑序列。

- 无后继的结点优先：每次输出一个无后继的结点并删去此结点及其入边，最后得到的序列是逆拓扑序列。

第八章 排序

记录中可用某一项来标识一个记录，则称为关键字项，该数据项的值称为关键字。
排序是使文件中的记录按关键字递增（或递减）次序排列起来。

- 基本操作：比较关键字大小；改变指向记录的指针或移动记录。
- 存储结构：顺序结构、链表结构、索引结构。

经过排序后这些具有相同关键字的记录之间的相对次序保持不变，则称这种排序方法是稳定的，否则排序算法是不稳定的。

排序过程中不涉及数据的内、外存交换则称之为“内部排序”（内排序），反之，若存在数据的内外存交换，则称之为外排序。

内部排序方法可分五类：插入排序、选择排序、交换排序、归并排序和分配排序。

评价排序算法好坏的标准主要有两条：执行时间和所需的辅助空间，另外算法的复杂程序也是要考虑的一个因素。

插入排序：• 直接插入排序： • 逐个向前插入到合适位置。

• 哨兵（监视哨）有两个作用： • 作为临变量存放 R[i]

• 是在查找循环中用来监视下标变量 j 是否越界。

• 直接插入排序是就地的稳定排序。时间复杂度为 O（n^2），比较次数为（n+2）（n-1）/2；移动次数为（n+4）（n-1）/2；

• 希尔排序： • 等间隔的数据比较并按要求顺序排列，最后间隔为 1.

•希尔排序是就地的不稳定排序。时间复杂度为 O（n^1.25），比较次数为（n^1.25）；移动次数为（1.6n^1.25）；

交换排序：•冒泡排序：•自下向上确定最轻的一个。•自上向下确定最重的一个。•自下向上确定最轻的一个，后自上向下确定最重的一个。

• 冒泡排序是就地的稳定排序。时间复杂度为 O（n^2），比较次数为 n（n-1）/2；移动次数为 3n（n-1）/2；

• 快速排序：• 以第一个元素为参考基准，设定、动两个指针，发生交换后指针交换位置，直到指针重合。重复直到排序完成。

• 快速排序是非就地的不稳定排序。时间复杂度为 O（nlog2n），比较次数为 n（n-1）/2；

选择排序：• 直接选择排序： • 选择最小的放在比较区前。

• 直接选择排序就地的不稳定排序。时间复杂度为 O（n^2）。比较次数为 n（n-1）/2；

• 堆排序 • 建堆：按层次将数据填入完全二叉树，从 int（n/2）处向前逐个调整位置。

• 然后将树根与最后一个叶子交换值并断开与树的连接并重建堆，直到全断开。

• 堆排序是就地不稳定的排序，时间复杂度为 O（nlog2n），不适宜于记录数较少的文件。

归并排序： • 先两个一组排序，形成（n+1）/2 组，再将两组并一组，直到剩下一组为止。

• 归并排序是非就地稳定排序，时间复杂度是 O（nlog2n），

分配排序：• 箱排序： • 按关键字的取值范围确定箱子数，按关键字投入箱子，链接所有非空箱。

• 箱排序的平均时间复杂度是线性的 O（n）。

• 基数排序：• 从低位到高位依次对关键字进行箱排序。

• 基数排序是非就稳定的排序，时间复杂度是 O（d*n+d*rd）。

各种排序方法的比较和选择： • 待排序的记录数目 n；n 较大的要用时间复杂度为 O（nlog2n）的排序方法；

• 记录的大小（规模）；记录大最好用链表作为存储结构，而快速排序和堆排序在链表上难于实现；

• 关键字的结构及其初始状态； • 对稳定性的要求；

• 语言工具的条件； • 存储结构； • 时间和辅助空间复杂度。

第九章 查找

查找的同时对表做修改操作（如插入或删除）则相应的表称之为动态查找表，否则称之为静态查找表。

衡量查找算法效率优劣的标准是在查找过程中对关键字需要执行的平均比较次数（即平均查找长度 ASL）。

线性表查找的方法： • 顺序查找：逐个查找，ASL=（n+1）/2；

• 二分查找：取中点 int（n/2）比较，若小就比左区间，大就比右区间。用二叉判定树表示。ASL=（Σ（每层结点数*层数））/N.

• 分块查找。要求“分块有序”，将表分成若干块内部不一定有序，并抽取各块中的最大关键字及其位置建立有序索引表。

二叉排序树（BST）定义是：二叉排序树是空树或者满足如下性质的二叉树： • 若它的左子树非空，则左子树上所有结点的值均小于根结点的值；

• 若它的右子树非空，则右子树上所有结点的值均大于根结点的值；

• 左、右子树本身又是一棵二叉排序树。

二叉排序树的插入、建立、删除的算法平均时间性能是 O（nlog2n）。

二叉排序树的删除操作可分三种情况进行处理： • *P 是叶子，则直接删除*P，即将*P 的双亲*parent 中指向*P 的指针域置空即可。

• *P 只有一个孩子*child，此时只需将*child 和*p 的双亲直接连接就可删去*p.

•*p 有两个孩子，则先将*p 结点的中序后继结点的数据到*p，删除中序后继结点。

关于 B-树（多路平衡查找树）。它适合在磁盘等直接存取设备上组织动态的查找表，是一种外查找算法。建立的方式是从下向上拱起。

散列技术：将结点按其关键字的散列地址存储到散列表的过程称为散列。散列函数的选择有两条标准：简单和均匀。

常见的散列函数构的造方法：

• 平方取中法：hash=int（（x^2）%100）

• 除余法：表长为 m，hash=x%m

• 相乘取整法：hash=int（m*（x*A-int（x*A））；A=0.618

• 随机数法：hash=random（x）。

处理冲突的方法：• 开放定址法： • 一般形式为 hi=（h（key）+di）%m1≤i≤m-1，开放定址法要求散列表的装填因子 α ≤1.

• 开放定址法类型： • 线性探查法：address=（hash（x）+i）%m；

• 二次探查法：address=（hash（x）+i^2）%m；

• 双重散列法：address=（hash（x）+i*hash（y））%m；

•拉链法： •是将所有关键字为同义词的结点链接在同一个单链表中。

• 拉链法的优点： • 拉链法处理冲突简单，且无堆积现象；

• 链表上的结点空间是动态申请的适于无法确定表长的情况；

• 拉链法中 α 可以大于 1，结点较大时其指针域可忽略，因此节省空间；

• 拉链法构造的散列表删除结点易实现。

• 拉链法也有缺点：当结点规模较小时，用拉链法中的指针域也要占用额外空间，还是开放定址法省空间。

第十章 排序

10.1 排序的基本概念

10.2 插入排序

10.3 选择排序

10.4 交换排序

本章主要知识点：

排序的基本概念和衡量排序算法优劣的标准，其中衡量标准有算法的时间复杂度、空间复杂度和稳定性

直接插入排序，希尔排序

直接选择排序，堆排序

冒泡排序，快速排序

10.1 排序的基本概念

1.排序是对数据元素序列建立某种有序排列的过程。

2.排序的目的：便于查找。

3.关键字是要排序的数据元素集合中的一个域，排序是以关键字为基准进行的。

关键字分主关键字和次关键字两种。对要排序的数据元素集合来说，如果关键字满足数据元素值不同时该关键字的值也一定不同，这样的关键字称为主关键字。不满足主关键字定义的关键字称为次关键字。

4.排序的种类：分为内部排序和外部排序两大类。

若待排序记录都在内存中，称为内部排序；若待排序记录一部分在内存，一部分在外存，则称为外部排序。

注：外部排序时，要将数据分批调入内存来排序，中间结果还要及时放入外存，显然外部排序要复杂得多。

5.排序算法好坏的衡量标准：

(1)时间复杂度—— 它主要是分析记录关键字的比较次数和记录的移动次数。

(2)空间复杂度——算法中使用的内存辅助空间的多少。

(3)稳定性——若两个记录 A 和 B 的关键字值相等，但排序后 A、B 的先后次序保持不变，则称这种排序算法是稳定的。

10.2 插入排序

插入排序的基本思想是：每步将一个待排序的对象，按其关键字大小，插入到前面已经排好序的一组对象的适当位置上，直到对象全部插入为止。

简言之，边插入边排序，保证子序列中随时都是排好序的。

常用的插入排序有：直接插入排序和希尔排序两种。

10.2.1 直接插入排序

1、其基本思想是：

顺序地把待排序的数据元素按其关键字值的大小插入到已排序数据元素子集合的适当位置。

例 1：关键字序列 T=（13，6，3，31，9，27，5，11），

请写出直接插入排序的中间过程序列。

初始关键字序列：【13】, 6, 3, 31, 9, 27, 5, 11

第一次排序：　　【6, 13】, 3, 31, 9, 27, 5, 11

第二次排序：　　【3, 6, 13】, 31, 9, 27, 5, 11

第三次排序：　　【3, 6, 13，31】, 9, 27, 5, 11

第四次排序：　　【3, 6, 9, 13，31】, 27, 5, 11

第五次排序：　　【3, 6, 9, 13，27, 31】, 5, 11

第六次排序：　　【3, 5, 6, 9, 13，27, 31】, 11

第七次排序：　　【3, 5, 6, 9, 11，13，27, 31】

注：方括号 [　]中为已排序记录的关键字，下划横线的 关键字表示它对应的记录后移一个位置。

2.直接插入排序算法

```
public static void insertSort(int[] a){
        int i, j, temp;
        int n = a.Length;
        for(i = 0; i < n - 1; i ++){
            temp = a[i + 1];
            j = i;
            while(j > -1 && temp < a[j]){
          a[j + 1] = a[j];
          j --;
            }
            a[j + 1] = temp;
        }
}
```

初始关键字序列：【13】, 6, 3, 31, 9, 27, 5, 11

第一次排序：　　【6, 13】, 3, 31, 9, 27, 5, 11

第二次排序：　　【3, 6, 13】, 31, 9, 27, 5, 11

3、直接插入排序算法分析

(1)时间效率：当数据有序时，执行效率最好，此时的时间复杂度为 O(n)；当数据基本反序时，执行效率最差，此时的时间复杂度为 O(n2)。所以当数据越接近有序，直接插入排序算法的性能越好。

(2)空间效率：仅占用 1 个缓冲单元——O（1）

(3)算法的稳定性：稳定

8.2.2 希尔（shell）排序 （又称缩小增量排序）

1、基本思想：把整个待排序的数据元素分成若干个小组，对同一小组内的数据元素用直接插入法排序；小组的个数逐次缩小，当完成了所有数据元素都在一个组内的排序后排序过程结束。　　2、技巧：小组的构成不是简单地“逐段分割”，而是将相隔某个增量 d 的记录组成一个小组,让增量 d 逐趟缩短（例如依次取 5,3,1），直到 d＝1 为止。

3、优点：让关键字值小的元素能很快前移，且序列若基本有序时，再用直接插入排序处理，时间效率会高很多。

例 2：设待排序的序列中有 12 个记录，它们的关键字序列 T=(65，34，25，87，12，38，56，46，14，77，92，23)，请写出希尔排序的具体实现过程。

```
public static void shellSort(int[] a, int[] d, int numOfD){
    int i, j, k, m, span;
    int temp;
    int n = a.Length;
    for(m = 0; m < numOfD; m ++){   //共 numOfD 次循环
        span = d[m];                //取本次的增量值
        for(k = 0; k < span; k ++){   //共 span 个小组
            for(i = k; i < n-span; i = i + span){
                temp = a[i+span];
                j = i;
```

```
                    while(j > -1 && temp < a[j]){
                        a[j + span] = a[j];
                        j = j - span;
                    }
                    a[j + span] = temp;
                }
            }
        }
}
```

算法分析：开始时 d 的值较大，子序列中的对象较少，排序速度较快；随着排序进展，d 值逐渐变小，子序列中对象个数逐渐变多，由于前面工作的基础，大多数记录已基本有序，所以排序速度仍然很快。

时间效率：O(n(log2n)2)

空间效率：O（1）——因为仅占用 1 个缓冲单元

算法的稳定性：不稳定

练习：

1. 欲将序列（Q, H, C, Y, P, A, M, S, R, D, F, X）中的关键码按字母升序重排，则初始 d 为 4 的希尔排序一趟的结果是？

答： 原始序列： Q, H, C, Y, P, A, M, S, R, D, F, X

shell 一趟后： P,A,C,S,Q,D,F,X,R,H,M,Y

2. 以关键字序列（256，301，751，129，937，863，742，694，076，438）为例，写出执行希尔排序（取 d=5,3,1）算法的各趟排序结束时，关键字序列的状态。

解：原始序列: 256，301，751，129，937，863，742，694，076，438

希尔排序第一趟 d=5 256 301 694 076 438 863 742 751 129 937

第二趟 d=3 076 301 129 256 438 694 742 751 863 937

第三趟 d=1 076 129 256 301 438 694 742 751 863 937

10.3 选择排序

选择排序的基本思想是：每次从待排序的数据元素集合中选取关键字最小（或最大）的数据元素放到数据元素集合的最前（或最后），数据元素集合不断缩小，当数据元素集合为空时选择排序结束。

常用的选择排序算法：

（1）直接选择排序

（2）堆排序

10.3.1 直接选择排序

1、其基本思想

每经过一趟比较就找出一个最小值，与待排序列最前面的位置互换即可。

（即从待排序的数据元素集合中选取关键字最小的数据元素并将它与原始数据元素集合中的第一个数据元素交换位置；然后从不包括第一个位置的数据元素集合中选取关键字最小的数据元素并将它与原始数据集合中的第二个数据元素交换位置；如此重复，直到数据元素集合中只剩一个数据元素为止。）

2、优缺点

优点：实现简单

缺点：每趟只能确定一个元素，表长为 n 时需要 n-1 趟

例 3：关键字序列 T= （21，25，49，25*，16，08），请给出直接选择排序的具体实现过程。

原始序列： 21，25，49，25*，16，08

第 1 趟 08，25，49，25*，16，21

第 2 趟 08，16， 49，25*，25，21

第 3 趟 08，16， 21，25*，25，49

第 4 趟 08，16， 21，25*，25，49

第 5 趟 08，16， 21，25*，25，49

```
public static void selectSort(int[] a){
    int i, j, small;
    int temp;
    int n = a.Length;
    for(i = 0; i < n - 1; i ++){
        small = i;                          //设第 i 个数据元素最小
        for(j = i + 1; j < n; j ++)      //寻找最小的数据元素
            if(a[j] < a[small]) small = j;      //记住最小元素的下标
        if(small != i){    //当最小元素的下标不为 i 时交换位置
```

```
                    temp = a[i];
                    a[i] = a[small];
                    a[small] = temp;
                }
        }
}
```

3、算法分析

时间效率： O(n2)——虽移动次数较少，但比较次数仍多。

空间效率：O（1）——没有附加单元（仅用到 1 个 temp)

算法的稳定性：不稳定

4、稳定的直接选择排序算法

例：关键字序列 T= （21，25，49，25*，16，08），请给出稳定的直接选择排序的具体实现过程。

原始序列： 21，25，49，25*，16，08

第 1 趟 08， 21， 25， 49， 25 *, 16

第 2 趟 08，16， 21，25，49 ，25 *

第 3 趟 08，16， 21，25，49 ，25 *

第 4 趟 08，16， 21，25，49 ，25 *

第 5 趟 08，16， 21，25，25 * ，49

```
public static void selectSort2(int [] a){
        int i,j,small;
        int temp;
        int n = a.Length;
        for(i = 0; i < n-1; i++){
                small = i;
            for(j = i+1; j < n; j++){          //寻找最小的数据元素
                    if(a[j] < a[small]) small = j;     //记住最小元素的下标
            }
            if(small != i){
            temp = a[small];
                    for(j = small; j > i; j--)          //把该区段尚未排序元素依次后移
                a[j] = a[j-1];
            a[i] = temp;                          //插入找出的最小元素
            }
        }
}
```

8.3.2 堆排序

1. 什么是堆？ 2. 怎样建堆？ 3. 怎样堆排序？

堆的定义：设有 n 个数据元素的序列 k0，k1，…，kn-1，当且仅当满足下述关系之一时，称之为堆。

解释：如果让满足以上条件的元素序列 （k 0 ，k 1 ，…，kn－ 1 ）顺次排成一棵完全二叉树，则此树的特点是：

树中所有结点的值均大于（或小于）其左右孩子，此树的根结点（即堆顶）必最大（或最小）。

例 4：有序列 T1=（08, 25, 49, 46, 58, 67）和序列 T2=（91, 85, 76, 66, 58, 67, 55）,判断它们是否 “堆”?

2. 怎样建堆？

步骤：从第一个非终端结点开始往前逐步调整，让每个双亲大于（或小于）子女，直到根结点为止。

终端结点（即叶子）没有任何子女，无需单独调整

例：关键字序列 T= (21，25，49，25*，16，08），请建最大堆。

解：为便于理解，先将原始序列画成完全二叉树的形式：

这样可以很清晰地从(n-1-1)/2 开始调整。

```
public static void createHeap(int[] a, int n, int h){
        int i, j, flag;
        int temp;
        i = h;
        j = 2 * i + 1;              // j 为 i 结点的左孩子结点的下标
        temp = a[i];
        flag = 0;
```

```
        while(j < n && flag != 1){
        //寻找左右孩子结点中的较大者,j 为其下标
            if(j < n - 1 && a[j] < a[j + 1]) j++;
            if (temp >= a[j])                //a[i]>=a[j]
                flag = 1;    //标记结束筛选条件
            else{                //否则把 a[j]上移
                a[i] = a[j];
                i = j;
                j = 2 * i + 1;
            }
        }
        a[i] = temp;
}
```

利用上述 createHeap（a,n,h）函数，初始化创建最大堆的过程就是从第一个非叶子结点 a[i]开始，到根结点 a[0]为止，循环调用 createHeap（a,n,h）的过程。

初始化创建最大堆算法如下：

```
public static void initCreateHeap(int[] a){
        int n = a.Length;
        for(int i = (n-1-1) / 2; i >= 0; i --)
            createHeap(a, n, i);
}
```

3. 怎样进行整个序列的堆排序？

*基于初始堆进行堆排序的算法步骤：

堆的第一个对象 a[0]具有最大的关键码，将 a[0]与 a[n-1]对调，把具有最大关键码的对象交换到最后;

再对前面的 n-1 个对象，使用堆的调整算法，重新建立堆（调整根结点使之满足最大堆的定义）。结果具有次最大关键码的对象又上浮到堆顶，即 a[0]位置;

再对调 a[0]和 a[n-2]，然后对前 n-2 个对象重新调整，…如此反复，最后得到全部排序好的对象序列。

例 5：对刚才建好的最大堆进行排序：

```
public static void heapSort(int[] a){
    int temp;
    int n = a.Length;
    initCreateHeap(a);                    //初始化创建最大堆
    for(int i = n - 1; i > 0; i --){  //当前最大堆个数每次递减 1
    //把堆顶 a[0]元素和当前最大堆的最后一个元素交换
        temp = a[0];
        a[0] = a[i];
        a[i] = temp;
        createHeap(a,i, 0);                    //调整根结点满足最大堆
    }
}
```

4、堆排序算法分析：

时间效率：O(nlog2n)。

空间效率：O(1)。

稳定性： 不稳定。

练习 1：以下序列是堆的是（ ）

{75,65,30,15,25,45,20,10}　　B. {75,65,45,10,30,25,20,15}

C. {75,45,65,30,15,25,20,10}　　D. {75,45,65,10,25,30,20,15}

练习 2：有一组数据{15,9,7,8,20,1,7*,4}，建成的最小堆为（ ）。

A.{1,4,8,9,20,7,15,7*}　　B.{1,7,15,7*,4,8,20,9}

C.{1,4,7,8,20,15,7*,9}　　D.以上都不对

练习 3：已知序列{503，87，512，61，908，170，897，275，653，462}，写出采用堆排序对该序列作非递减排列时的排序过程。

排序好的序列为：61，87，170，275，462，503，512，653，897，908

10.4 交换排序

交换排序的基本思想是：利用交换数据元素的位置进行排序的方法。

交换排序的主要算法有：

1) 冒泡排序
2) 快速排序

10.4.1 冒泡排序

1、基本思路：每趟不断将记录两两比较，并按“前小后大”（或“前大后小”）规则交换。

2、优点：每趟结束时，不仅能挤出一个最大值到最后面位置，还能同时部分理顺其他元素；一旦下趟没有交换发生，还可以提前结束排序。

例：关键字序列 T=(21，25，49，25*，16，08)，请按从小到大的顺序，写出冒泡排序的具体实现过程。

初态：21，25，49， 25*，16， 08

第 1 趟 21，25，25*，16， 08 ， 49

第 2 趟 21，25， 16， 08 ，25*，49

第 3 趟 21，16， 08 ，25， 25*，49

第 4 趟 16，08 ，21， 25， 25*，49

第 5 趟 08，16， 21， 25， 25*，49

3、冒泡排序算法

```
public static void bubbleSort(int[] a){
        int i, j, flag=1;
        int temp;
        int n = a.Length;
        for(i = 1; i < n && flag == 1; i++){
                flag = 0;
                for(j = 0; j < n-i; j++){
                        if(a[j] > a[j+1]){
                                flag = 1;
                                temp = a[j];
                                a[j] = a[j+1];
                                a[j+1] = temp;
                        }
                }
        }
}
```

4、冒泡排序的算法分析

时间效率：O（n2)—因为要考虑最坏情况（数据元素全部逆序），当然最好情况是数据元素已全部排好序，此时循环 n-1 次，时间复杂度为 O（n)

空间效率：O（1） —只在交换时用到一个缓冲单元

稳 定 性： 稳定 —25 和 25*在排序前后的次序未改变

练习：关键字序列 T=(31，15，9，25，16，28)，请按从小到大的顺序，写出冒泡排序的具体实现过程。

初态：31，15，9， 25，16， 28

第 1 趟 15，9，25，16， 28， 31

第 2 趟 9，15， 16， 25，28，31

第 3 趟 9，15， 16，25， 28，31

1、基本思想：设数组 a 中存放了 n 个数据元素，low 为数组的低端下标，high 为数组的高端下标，从数组 a 中任取一个元素（通常取 a[low]）做为标准元素，以该标准元素调整数组 a 中其他各个元素的位置，使排在标准元素前面的元素均小于标准元素，排在标准元素后面的均大于或等于标准元素。这样一次排序过程结束后，一方面将标准元素放在了未来排好序的数组中该标准元素应位于的位置上，另一方面将数组中的元素以标准元素为中心分成了两个子数组，位于标准元素左边子数组中的元素均小于标准元素，位于标准元素右边子数组中的元素均大于等于或标准元素。对这两个子数组中的元素分别再进行方法类同的递归快速排序。算法的递归出口条件是 low≥high。

例、关键字序列 T=(60，55，48，37，10，90，84，36)，请按从小到大的顺序，写出快速排序的具体实现过程。

快速排序算法各次快速排序过程

3、快速排序算法

```
public static void quickSort(int[] a, int low, int high){
        int i, j;
        int temp;
        i = low;
        j = high;
        temp = a[low];   //取第一个元素为标准数据元素
                        while(i < j){
                                        //在数组的右端扫描
```

```
            while(i < j && temp <= a[j]) j--;
            if(i < j){
                a[i] = a[j];
                i++;
            }
                                //在数组的左端扫描
                                while(i < j && a[i] < temp) i++;
            if(i < j){
                a[j] = a[i];
                j--;
            }
            }
        a[i] = temp;
        if(low < i) quickSort(a, low, i-1);       //对左端子集合递归
        if(i < high) quickSort(a, j+1, high);          //对右端子集合递归
}
```

4、快速排序算法分析：

时间效率：一般情况下时间复杂度为 O(nlog2n)，最坏情况是数据元素已全部正序或反序有序，此时每次标准元素都把当前数组分成一个大小比当前数组小 1 的子数组，此时时间复杂度为 O(n2)

空间效率：O（log2n）—因为递归要用栈

稳 定 性： 不 稳 定 —因为有跳跃式交换。

练习：已知序列{503,87,512,61,908,170,897,275,653,462}，给出采用快速排序对该序列作非递减排序时每趟的结果。

第一趟：【462 87 275 61 170】 503 【 897 908 653 512】

第二趟：【170 87 275 61 】 462 503 【512 653】 897 【908】

第三趟：【61 87】 170 【275】 462 503 512 【653】 897 908

第四趟：61 【 87】 170 275 462 503 512 653 897 908

最后排序结果：61 87 170 275 462 503 512 653 897 908

1.插入排序是稳定的，选择排序是不稳定的。

2.堆排序所需要附加空间数与待排序的记录个数无关。

3.对有 n 个记录的集合进行快速排序，所需时间确定于初始记录的排列情况，在初始记录无序的情况下最好。

4.直接插入排序在最好情况下的时间复杂度为（ A ）

A.O(n) B.O(nlog2n) C. O(log2n) D.O(n2)

5.数据序列{8,9,10,4,5,6,20,1,2}只能是（ C ）算法的两趟排序后的结果。

A.直接选择排序 B.冒泡排序 C.直接插入排序 D.堆排序

6.用直接插入排序对下面 4 个序列进行递增排序，元素比较次数最少的是（C ）

A.94,32,40,90,80,46,21,69 B.32,40,21,46,69,94,90,80

C.21,32,46,40,80,69,90,94 D.90,69,80,46,21,32,94,40

7. .以下排序算法中，（B ）不能保证每趟排序至少能将一个元素放到其最终位置上。

A. 快速排序 B.希尔排序 C.堆排序 D.冒泡排序

8.对关键字{28,16,32,12,60,2,5,72}序列进行快速排序，第一趟从小到大一次划分结果为（B ）

A.(2,5,12,16)26(60,32,72) B.(5,16,2,12)28(60,32,72)

C.(2,16,12,5)28(60,32,72) D.(5,16,2,12)28(32,60,72)

9.若用冒泡排序对关键字序列{18，16，14，12，10，8}进行从小到大的排序，所需进行的关键字比较总次数是（ B ）。

A.10 B.15 C.21

D.34

10. 一组记录的关键字为{45，80，55，40，42，85}，则利用堆排序的方法建立的初始堆为（ B ）。

A.{85， 80， 45， 40， 42， 55} B.{85， 80， 55， 40， 42， 45}

C.{85，80，55，45，42，40} D.{85，55，80，42，45，40}

第十章 文件

文件是性质相同的记录的集合。记录是文件中存取的基本单位，数据项是文件可使用的最小单位，数据项有时称字段或者属性。

文件 • 逻辑结构是一种线性结构。

• 操作有：检索和维护。并有实时和批量处理两种处理方式。

文件 • 存储结构是指文件在外存上的组织方式。

• 基本的组织方式有：顺序组织、索引组织、散列组织和链组织。

• 常用的文件组织方式：顺序文件、索引文件、散列文件和多关键字文件。

评价一个文件组织的效率，是执行文件操作所花费的时间和文件组织所需的存储空间。

检索功能的多寡和速度的快慢，是衡量文件操作质量的重要标志。

顺序文件是指按记录进入文件的先后顺序存放、其逻辑顺序和物理顺序一致的文件。主关键字有序称顺序有序文件，否则称顺序无序文件。

一切存储在顺序存储器（如磁带）上的文件都只能顺序文件，只能按顺序查找法存取。

顺序文件的插入、删除和修改只能通过复制整个文件实现。

索引文件的组织方式：通常是在主文件之外建立一张索引表指明逻辑记录和物理记录之间一一对应的关系，它和主文件一起构成索引文件。

索引非顺序文件中的索引表为稠密索引。索引顺序文件中的索引表为稀疏索引。

若记录很大使得索引表也很大时，可对索引表再建立索引，称为查找表。是一种静态索引。

索引顺序文件常用的有两种：

• ISAM 索引顺序存取方法：是专为磁盘存取文件设计的，采用静态索引结构。

• VSAM 虚拟存储存取方法：采用 B+树作为动态索引结构，由索引集、顺序集、数据集组成。

散列文件是利用散列存储方式组织的文件，亦称为直接存取文件。

散列文件

• 优点是：文件随机存放，记录不需要排序；插入删除方便；存取速度快；不需要索引区，节省存储空间。

• 缺点是：不能进行顺序存取，只能按关键字随机存取，且询问方式限地简单询问，需要重新组织文件。

多重表文件：对需要查询的次关键字建立相应的索引，对相同次关键字的记录建一个链表并将链表头指针、长度、次关键字作为索引表的索引项。

倒排表：次关键字索引表称倒排表，主文件和倒排表构成倒排文件。