前言

951 数据结构官方指定教材为荣正的数据结构与算法分析。

本资料不是课本的简化版本，不能作为独立的辅导书

因为 951 对知识点的考察非常细节，所以该资料目的，是让同学们能在学习荣正课本时候，知道课本哪些内容是坑，要重点看；哪些内容不重要，大致看一下就 ok。

因此，本资料并不是将荣正的课本内容进行简单地压缩，而是将需要格外注意的细节反复强调，以帮助同学们掌握重点。

使用方法：

(1) 资料严格按照荣正目录安排，所以大家看一小节荣正课本，再看一下该小节对应的资料了解重点和坑点，最后再去看一遍课本

(2) 仅有知识点是不够的，大家要把课本后面的习题做完，不会的，看参考答案进行总结。有一些知识只有习题里才有，认真把这些点记好便于下一轮复习

(3) 大家做其他习题集时候，比如王道和天勤，一定要注意辅导书和荣正课本的一些细微偏差，特别是定义方面，要严格按照荣正来

绪论

- 数据元素是数据的基本单位
- 数据项是数据的最小单位
- 数据结构：二元组（D，R），D 是数据，R 是关系，可考判断题，混淆 D 和 R 的含义
- 数据结构包含三部分
    - 逻辑结构
    - 存储结构
    - 在逻辑和存储结构上进行的操作
- 抽象数据类型包含三部分
    - 数据对象
    - 数据关系
    - 操作集合
- 逻辑结构：线性和非线性
- 存储结构
    - 顺序存储
    - 链式存储
    - 索引存储
    - 散列存储
- 算法必须满足的性质
    - 输入性
    - 输出性
    - 有穷性
    - 确定性
    - 可行性
- 注意：算法必须满足的性质，和算法尽量达到的性质要区分，比如，可读性这种性质，是尽量达到的，而非必须达到
- 算法不等于程序，算法不一定要用代码实现，性质上也不同，比如，算法必须满足有穷性，而程序可以不满足
- 程序结构设计：常用层次结构，包括自顶向下和自底向上

线性表

一．线性表的基本概念及运算

- 基本概念
    - 1.定义：线性表是具有相同特性数据元素的一个有限序列
    - 2.表示：(a1, a2, ..., an)，注意这是逻辑表示，下标从 1 开始
    - 3.线性表中的数据元素，可以是简单元素，如一个数字，一个字母，也可以是复杂的数据结构，如一个结构体
    - 4.线性表中的数据元素具有相同特性，不可以在表中出现两种及以上种类的数据结构，如，a1 是一个字母，a2 是一个数字，这种情况是不允许出现的
    - 5.线性表长度可用 n 来表示，n 可以为 0，代表该线性表是空表；但是 n 不可以为无穷大，它必须是一个有限的数
    - 6.逻辑特征（对于非空线性表来讲）
        - 有且仅有一个开始结点 a1；有且仅有一个终端结点 an
        - 除终端结点外，线性表中其他所有结点都有且仅有一个直接后继
        - 除开始结点外，线性表中其他所有结点都有且仅有一个直接前趋
- 基本运算
    - 1.在荣正 67 页，定义了很多线性表的基本运算。特别注意 Get 运算和 Locate 运算的区别，一个是按位置查找，一个是按值查找，由此引出了在特定存储方式下时间复杂度的不同
    - 2.注意这些运算的定义和具体实现是有区别的
        - 运算的定义是在逻辑结构上的，比如，线性表逻辑上元素下标从 1 开始，Insert 运算和 Delete 运算的定义，下标也是从 1 开始的
        - 运算的具体实现是在存储结构上进行的，比如，一个顺序表（线性表的顺序存储），删除它的第一个元素，在逻辑上，就是删除下标为 1 的元素，而在实际的存储中（用数组存储），是要删除下标为 0 的数组元素
    - 3.除了这些基本运算，还存在很多根据基本运算衍生出的复杂运算，在以后做题过程中积累即可

二．顺序表

- 基本概念

- 1.线性表的顺序存储结构：用一组地址连续的存储单元依次存储线性表的元素，可随机存取

- 2.注意荣正 68 页和 69 页的两个公式，这对之后学习数组计算相关问题时有用

- 3.注意荣正 69 页，顺序表结构体的定义，各个数据域的含义
  - 这个结构体的定义，已经是计算机运行的代码了，所以是从存储结构上来说的，其中 last 的含义也是在存储结构上来说的，它表示的是在存储结构（数组）中的最后元素的位置。
  - 向量就是一维数组
  - 逻辑下标和存储下标相差 1，比如逻辑上线性表从下标 1 开始，顺序存储中，下标从 0 开始

  - last 表示线性表终端结点在向量空间中的位置，而向量（一维数组）下标从 0 开始，所以线性表的当前长度为 last+1
- 4.特点：以元素在计算机内物理位置上的紧邻表示线性表中元素之间相邻的逻辑关系

- 基本运算

  - 插入运算
    - 之前说过，运算的定义是在逻辑结构上的，故定义线性表的插入操作时，下标从 1 开始，也就是说，长度为 n 的线性表，可插入的位置为 1 到 n+1

    - 但是在实际写代码中，又必须把逻辑结构对应到存储结构上。这里插一句，记住，只要遇到代码，那就是进入存储结构层面了

    - 逻辑上将新元素插入到线性表第 i 个位置：下标从 1 开始计算，将原本的第 i 个到第 n 个元素都往后挪一个位置。这样只有第 1 个到第 i-1 个位置没有被挪动，此时第 i 个位置空出来，新元素插入，表长加 1。

    - 存储结构上将新元素插入到线性表第 i 个位置：上述逻辑定义的下标减 1，对应到存储结构中即可，注意元素移动顺序为从后往前。也就是说，在代码中，数组下标为 0 到 i-2 这 i-1 个元素没有挪动，下标为 i 到 n-1 都挪动一个位置，现在空出来下标为 i-1 的位置给新元素。

    - 综合以上，再次提醒，运算的定义是在逻辑结构上说的，实际代码是在存储结构上写的。一句话，你想干什么，是逻辑结构；你想怎么干，是存储结构。
  - 删除运算

    - 可以删除的元素下标为 1 到 n（逻辑）

        - 逻辑上删除第 i 个位置的元素：删除第 i 个位置的元素，第 i+1 到第 n 号元素都前移一个位置，这样表长就减少 1。

        - 存储结构上删除第 i 个位置的元素：从数组下标为 i 的元素开始，到下标为 n-1 的元素为止，依次前移一个位置，即，让数组下标为 i 的元素覆盖 i-1 号元素，后面元素同理。

        - 注意：我们说“删除 i 号元素”实际上是在逻辑结构上说的，实际上在数组中，要删除的 i 号元素就是数组下标为 i-1 的元素。
    - 算法分析

        - 设表长为 n

        - 在顺序表中，插入运算和删除运算，时间复杂度都为 O(n)

        - 顺序表中插入一个元素，需要移动的元素个数的平均值为：n/2

        - 在顺序表中删除一个元素，需要移动的元素个数的平均值为：(n-1) / 2

        - n 较大时，顺序表进行插入和删除，效率很低

        - 看一下荣正 76 页，顺序表的优缺点。为克服顺序表的缺点，引出线性表的链式存储结构。

三．链表

- 单链表
    - 定义：用一组任意的存储单元来存放线性表的元素，这组存储单元可以连续也可以不连续。

    - 一般用头指针 head 来表示链表。
        - 链表带头结点时，head 指向头结点
        - 链表不带头结点时，head 指向第一个元素结点

        - 总之，不管链表是否带头结点，head 都指向这个链表的第一个结点。
    - 因为链表中的结点都是动态生成的，除了第一个结点由 head 指向以外，其他结点，都由其前趋结点中的 next 指针来指向，所以链表不能像顺序表那样，想拿哪个就拿哪个。想拿链表中的元素时候，必须从 head 开始，从头往后挨个找
- 单链表的基本运算

    - 单链表的建立
        - 头插法

        - 荣正 78 页代码
        - 头插法生成的链表中结点的次序，和输入的顺序相反
        - 算法时间复杂度为 O(n)

    - 尾插法
        - 荣正 79 页代码
        - 尾插法生成的链表中结点的词语，和输入的顺序相同
        - 算法时间复杂度为 O(n)
    - 头结点
        - 951 考试中，只有链栈默认不带头结点，其他的链表都默认带头结点
        - 优点
            - 各结点操作统一，无须进行特殊处理
            - 空表和非空表的处理统一
- 单链表的查找
    - 按序号查找
        - 单链表不能随机存取，只能顺序存取，所以找到 i 号结点，必须从头开始找
        - 查找的定义依然是在逻辑上来讲的，如表长为 n，则有效查找序号为 1 到 n
        - 存储结构上，可以把头结点当成 0 号结点
        - 时间复杂度：O(n)
    - 按值查找
        - 时间复杂度：O(n)
- 链表的插入
    - 注意荣正 81 页，结点插入链表时，指针的操作顺序
    - 单纯的插入操作，时间复杂度为 O(1)，但是由于插入位置的前趋指针不一定直接给出，所以要从头开始找，查找的时间复杂度为 O(n)
    - 荣正 82 页例题，因为需要从头开始找到新结点的插入位置，所以时间复杂度为 O(n)，主要开销在查找插入位置
    - 82 页例题为前插操作，我们平时说的链表插入操作，一般是后插
    - 83 页提到的改进前插性能，思路为：用后插操作，交换两个元素的值，便达到前插的效果
- 链表的删除
    - 注意荣正 83 页，链表中删除结点时，指针的操作顺序

        - 单纯的删除操作，时间复杂度为 O(n)，单由于删除位置的前趋指针不一定直接给出，所以要从头开始找，因此总时间复杂度为 O(n)
    - 链表合并
        - 两个递增链表合并为一个递增（递减）链表，用尾插法（头插法）
        - 84 页例题 2-5
- 循环链表
    - 将单链表中最后一个结点的 next 指针指向头结点，便构成了单循环链表
    - 默认带头结点
    - 运算与单链表大部分相同，区别在于对最后一个结点的处理不同
    - 表示
        - 用头指针表示的单循环链表中，查找开始结点的时间复杂度为 O(1)，查找终端结点的复杂度为 O(n)
        - 用尾指针表示的单循环链表中，查找开始结点和终端结点的时间复杂度都是 O(1)
        - 因此，实际应用中多采用尾指针表示的循环单链表
    - 以后做题过程中，问到“对某种循环单链表进行 XX 操作，时间复杂度是多少”，这种题目一定要注意，操作之后，依然要保证链表是循环的，单纯操作可能复杂度是 O(1)，但是要保证链表的循环特性，复杂度可能就是 O(n)
- 双向链表
    - 荣正 87 页，双链表的插入和删除操作，指针的操作顺序
    - 单纯插入和删除操作，时间复杂度为 O(1)，但是实际题目中，主要开销还是在寻找插入和删除的位置，因此复杂度为 O(n)

四．对比与总结

- 顺序表与链表对比
    - 存储密度：顺序表大，链表小
    - 运算时间
        - 存取：顺序表随机存取，查找元素快，精准定位；链表只能顺序存取，必须从头开始找
        - 插入和删除：顺序表要移动大量的元素；链表无需大量移动元素，但是查找插入/删除位置需要一定时间
    - 静态链表：不需要实际的指针来模拟动态存储结构
- 总结
    - 时间复杂度

        - 顺序表
            - 插入运算：O(n)
            - 删除运算：O(n)
        - 链表
            - 建立头插：O(n)
            - 建立尾插：O(n)
            - 按序号查找：O(n)
            - 按值查找：O(n)
            - 单纯插入：O(1)
            - 单纯删除：O(1)
            - 完整插入：O(n)
            - 完整删除：O(n)
    - 存储方式与存取方式
        - 顺序表是线性表的顺序存储方式，它具有随机存取的特性
        - 链表是线性表的链式存储方式，它具有顺序存取的特性
    - 逻辑结构存储结构
        - 951 对算法的定义，是逻辑上定义，存储上实现，注意各自下标开始的序号
        - 没看到代码，只提出做什么操作，这是逻辑定义
        - 看到了代码，这可就是存储结构上玩真格的了
        - 各章要注意，除了第四章串逻辑上下标从 0 开始，其他线性的数据结构，逻辑上下标从 1 开始。
        - 注意荣正课本和其他考研辅导书上定义的区别，以荣正课本为主

栈和队列

一. 栈

- 基本概念
    - 定义：栈是限定仅在表尾进行插入和删除运算的线性表（也就是在栈顶进行操作，但是顺序栈的栈顶是表尾，链栈的栈顶是表头，定义不够严谨）
    - 表尾称为栈顶，表头称为栈底
    - 注意，栈是一种线性表，只不过对栈的运算都是在栈顶进行，其他性质都和之前讲过的线性表相同，包括逻辑上和存储上

- 表示：S=(a1, a2, ..., an)

- 基本运算
  - 入栈
  - 出栈
  - 取栈顶元素
  - 栈的置空
  - 栈的判空

- 栈的用途
  - 汇编处理程序中的句法识别
  - 表达式的计算
  - 回溯问题
  - 括号匹配
  - 函数递归调用

- 栈的顺序存储——顺序栈
  - 栈的顺序存储结构
    - 顺序栈是运算受限的顺序表
    - 顺序栈的运算在表尾进行，也就是说，表尾才是栈顶
    - 用向量来存储，也就是一维数组
    - 荣正 96 页顺序栈的结构体定义
    - 注意，栈空时，Top 的值为-1
    - 顺序栈的几种运算简单描述
      - 置空：让 Top 值为-1
      - 判空：判断 Top 值是否为-1，或者判断 Top 是否小于 0
      - 入栈：先让 Top 值加 1，再把元素存入
      - 出栈：先让元素移除，再让 Top 值减 1，注意和取栈顶运算的区别
      - 取栈顶元素：直接取栈顶，Top 值不用改变，注意和出栈运算的区别，也就是是否需要改变 Top 值
  - 共享栈
    - 目的：节约存储空间，降低上溢发生的概率
    - 最常用的是两个栈共享存储空间
      - 栈底在两端，对应到长为 n 的数组上，栈底分别为-1 和 n

        - 设栈顶分别为 Top0 和 Top1，则初始时，Top0 为-1，Top1 为 n
        - 栈 0 的第一个元素为 0 号，栈 1 的第一个元素为 n-1 号
        - 0 号栈入栈时候，Top0 增加，1 号栈入栈时候，Top1 减少；0 号栈出栈时候，Top0 减少，Top1 增加
        - 两个栈栈顶相遇时，栈满，即 Top0 + 1 == Top1 时，栈满

        - 多个栈共享存储空间，简单了解即可
- 栈的链式存储——链栈
    - 链栈是运算受限的单链表
    - 使用链栈可以解决顺序栈产生的空间浪费和上溢问题
    - 链栈的运算在表头进行，也就是说，表头才是栈顶，这点要和顺序栈区分
    - 注意，在 951 中，链式的结构，只有链栈和链串是默认不带头结点的，其他链式结构默认带头结点
    - 荣正 99 页的链栈结构体定义，以及入栈出栈代码要掌握
    - Top 是栈顶指针，它唯一确定一个链栈，与单链表的头指针 head 作用基本相同，只不过 head 一般指向头结点，而 Top 直接指向栈顶元素结点
- 栈的应用
    - 过程递归调用：荣正 101 页，递归调用过程要理解
    - 地图染色问题：不重要，大致了解即可
    - 表达式求值：代码不需要了解，但是要了解过程，可以出小题

二．队列

- 基本概念
    - 队列是一种运算受限的线性表
    - 只允许在表的一端进行插入，在另一端进行删除
    - 允许插入的一端称为队尾，允许删除的一端称为队头
    - 其他性质和线性表相同
    - 运算
        - 入队
        - 出队
        - 取队头元素
        - 置空
        - 判空
- 队列的顺序存储——顺序队
    - 注意荣正 107 页，传统顺序队入队和出队时候，头指针和尾指针变化过程

- 头指针始终指向第一个元素之前一个位置，尾指针指向队尾元素

- 引入循环队列
  - 头指针始终指向第一个元素之前一个位置，尾指针指向队尾元素
  - 循环队列克服假上溢，而非真上溢
  - 出队时，头指针先加 1，再出队；入队时，尾指针先加 1，再入队。与顺序栈的操作顺序相区分

  - 循环队列始终有一个空间是空闲的，即，用长为 n 的数组表示循环队列，最多能存储 n-1 个元素

  - 有些需要技巧的题目，可能会让长为 n 的数组表示的循环队列，存满，不用浪费一个空间，之后会遇到这样的题目
- 注意，循环队列是队列的顺序表示
- 循环队列运算简单描述

  - 置空：设队列长为 n，则初始时，头尾指针都为 n-1

  - 判空：头指针等于尾指针

  - 入队：先让尾指针加 1，再让元素进队

  - 出队：先让头指针加 1，再让元素出队

  - 取队头元素：直接取序号为(头指针+1)的元素，头指针不要动

- 队列的链式存储——链队
  - 链队由一个头指针和一个尾指针唯一确定
  - 链队是一个特殊的单链表，注意，表头就是队头，表尾就是队尾
  - 链队默认带头结点
  - 运算：见荣正 110 页

    - 置空：头指针和尾指针都指向头结点

    - 判空：头尾指针相等

    - 入队

    - 出队：若当前队列长度大于 1，则直接出队即可；若当前队列仅有一个元素结点，则头指针和尾指针都要修改

    - 取队头元素：不必修改头指针
- 队列的应用：知道这两个应用是使用队列这种数据结构即可，有兴趣可以看看过程
  - 划分子集问题
  - 离散事件仿真
- 双端队列
  - 两个栈的栈底连在一起

- 输入受限的双端队列
- 输出受限的双端队列
- 以上两种双端队列做一道题即可理解

串和数组

一．串及其运算

- 基本概念
  - 串是由零个或多个字符组成的有限序列
  - 表示： S = "a0a1...an-1"，注意要带双引号
  - 串也是一种线性表，只不过它的数据元素仅由字符组成
  - 串的数据元素，即字符，不要想当然认为字符就是字母，实际上，字符包含很多，除了字母，还有数字等等
  - 注意，串的数据元素是单个字符
  - 逻辑定义上与之前学过的线性结构稍有不同，串的下标从 0 开始
  - 串中所包含的字符个数称为串的长度
  - 长度为 0 的串称为空串，注意，空格串不是空串，空串不包含任何字符，而空格也是字符
  - C 语言中，将串存储在数组中时，会在串的最后一个字符之后，再自动加一个字符，即'\0'，称为结束符
  - 注意，在计算串长度时候，虽然串后面跟着结束符'\0'，但是这个结束符并不参与串长度的计算
  - 比如，S = "abcde"，它在数组中存储时候是这样，"abcde\0"，但是计算串 S 长度时候，依然是 5
  - 串中任意个连续的字符组成的子序列称为串的子串，包含子串的串称为主串
  - 注意，子串的定义中，要求连续，它必须是主串中连续的一部分
  - 空串是任意串的子串，任意串是其自身的子串

- 子串的第一个字符在主串中的序号（下标），定义为该子串在主串中的位置，注意这个序号从 0 开始，没错，逻辑上就是从 0 开始。

- 荣正 121 页开始，那些算法，好好思考一下过程，不需要记住。考试如果要用这些函数，会给出用法

二．串的存储结构

- 顺序存储——顺序串
  - 一片连续的存储单元来存储串，一个字符占一个字节

  - 注意荣正 124 页顺序串的结构体定义以及图 4-1
- 链式存储——链串

  - 一个链串由一个头指针唯一确定，注意图 4-2，链串默认没有头结点
- 索引存储

  - 带长度的索引表

  - 带末指针的索引表

  - 带特征位的索引表

三．串运算的实现

- 串在大多数情况下的存储方式是顺序存储
- 基本运算的实现：细节见荣正 127 页
  - 串的连接运算

  - 求子串运算

  - 子串定位
    - 子串定位又称为串的模式匹配，本节最重点
    - 分两种，串的朴素匹配，以及 KMP 算法
    - 朴素匹配
      - 设主串 S 长为 n，子串长为 m
        - 最好情况下算法平均时间复杂度为 O(m+n)
        - 最坏情况下算法时间复杂度为 O(m*n)
        - 荣正 130 页朴素匹配算法代码要过一遍

    - 改进的模式匹配算法——KMP
      - 要求：会求 next 数组
      - 算法时间复杂度为 O(m)
      - KMP 优点：指示目标串的指针不需要回溯

        - 注：一般情况下，朴素匹配算法时间复杂度近似为 O(m+n)，故至今仍被采用

四．数组的定义和运算

- 一维数组
    - 注意下标依然从 0 开始，逻辑上和存储上都是从 0 开始
    - 每个元素由值和一个下标确定
    - 一维数组是一种元素数目固定的线性表
- 二维数组
    - 每个元素由值和一对下标确定
    - 二维数组整体上来看，是非线性的，所以遇到题目，如”以下哪种数据结构是线性的“，不要选数组，因为二维数组不是线性的
    - 二维及多维数组，可以看作是线性表的推广
- 数组的性质
    - 数据元素数目固定
    - 数据元素具有相同的类型
    - 数据元素的下标关系具有上下界的约束，并且下标有序
- 数组的运算
    - 存取
    - 修改
    - 注意，不要把数组和线性表的顺序存储混为一谈，虽然线性表的顺序存储情况下，有很多种运算，但是单纯的数组运算，通常只有存取和修改

五．数组的顺序存储结构

- 前面说过，数组的两种基本运算是存取和修改，它一般不做插入和删除运算
- 在讨论数组这种数据结构时候，它可以是一维，也可以是多维，但是，计算机内的存储单元，是一维的
- 接上条，我们现在知道，二维数组，就像一个矩阵，但是它在计算机存储单元内，所有元素是排成一行的
- 接上条，所以，必须规定好下标的优先级
- 二维数组，两种优先级，即行优先和列优先

- 多维数组，可按照正下标顺序优先或者逆下标顺序优先
- 荣正 135 页的公式好好看一下

六．矩阵的压缩存储

- 定义：所谓压缩存储，是指对多个值相同的元素只分配一个空间，对零元素不分配空间
- 注意压缩的含义：用更小的存储空间来存储矩阵中的必要元素，比如下面说的一维数组或者三元组
- 特殊矩阵
    - 定义：对于值相同的元素或者零元素的分布具有一定规律的矩阵，称为特殊矩阵
    - 对角矩阵
        - 设 n*n 矩阵
        - 带状
        - 最简单情况：只在主对角线上含有非零元素，此时可用长为 n 的一维数组来存储
        - 接上条，非零元素在对角矩阵中的下标，与存储非零元素的一维数组的下标，可以找到对应关系
        - 接上条，这个对应关系，建议手动硬推，尽量别用公式，很容易用错
    - 三角矩阵
        - 设 n*n 矩阵
        - 分为上三角矩阵和下三角矩阵
        - 三角矩阵中，值相同的元素可以只分配一个存储空间，如果这些值相同的元素是 0，则不分配存储空间
        - 主要元素有 n*(n+1) / 2 个，是否需要加一个空间，根据其他元素是否为 0 来决定
        - 三角矩阵中元素的下标和存储该矩阵的一维数组的下标，对应关系的计算，尽量手动硬推
    - 对称矩阵
        - 仅需要 n*(n+1)个空间来存储，即可视为三角矩阵，再存储到一维数组中
    - 以上几种特殊矩阵，总能找到一个关系将其压缩到一维数组中，通过这个关系可以对矩阵的元素进行随机存取

- 稀疏矩阵
  - 定义：含有非零元素以及较多的零元素，但非零元素的分布没有任何规律，这称为稀疏矩阵
  - 稀疏矩阵中零元素数量远大于非零元素数量
  - 稀疏矩阵的两种压缩存储方法：三元组和十字链表
  - 三元组表是稀疏矩阵的一种顺序存储结构
  - 注意荣正 138 页，稀疏矩阵结构体的定义
  - 接上条，结构体定义中，注意，包括了行数，列数，以及非零元素个数

  - 接上条，但是，画三元组表时候，不要加上”行数，列数，非零元素个数“这一行，三元组表第 0 行直接就表示第一个非零元素的信息，注意看图 4-13

  - 稀疏矩阵的转置——三元组表示法下的转置
    - 不可直接交换行列下标
    - 需要重新排列三元组的顺序

    - 设稀疏矩阵列数为 n，非零元素个数为 t，则转置算法时间复杂度为 O(n*t)

    - 通常二维数组表示矩阵时，转置算法时间复杂度为 O(n*m)

    - 因为稀疏矩阵中非零元素个数一般大于行数，所以，压缩存储后进行转置的时间，要大于非压缩存储情况下进行转置的时间

树

一．树的基本概念

- 荣正 142 页树的定义中，树根无前趋，树种其他结点都有前趋
- 143 页的若干术语中，注意树的度，是指树中最大结点度数，不要记成树种所有结点的度数和
- 存储方式包括顺序存储和链式存储
- 链式存储中，每个指针域指向一棵子树的根节点，而没有指针指向树根，所以树的链式存储中，结点数比指针数多 1，多的这一个就是树根
- 上述结论在计算题中经常用到

二．二叉树

- 荣正 144 页和 145 页

- 二叉树的五种基本形态
- 二叉树的两种特殊形态，即满二叉树和完全二叉树
- 注意完全二叉树的另一种描述：若一棵二叉树至多只有最下面两层上的结点度数可以小于 2，并且最下面一层的结点都集中在该层最左边的若干位置上，则称其为完全二叉树
- 在荣正课本的描述下，二叉树既不是有序树也不是无序树
- 二叉树的 4 个性质
    - 性质 3 利用了树中总结点数比总指针数多 1 这个结论
    - 性质 4 描述的是完全二叉树，不是普通二叉树
- 注：不仅仅是二叉树，多叉树的习题也要练，主要用到了总结点数比总指针数多 1；后面的哈夫曼树中，也有多叉的哈夫曼树，遇到相关题目注意公式的推导

三．二叉树的存储结构

- 首先强调，本节所说的存储结构是对二叉树来讲的，注意与后面普通树的存储结构区分
- 顺序存储结构
    - 按照一定的顺序化方式，将二叉树的所有结点存储到一片连续的存储单元中。
    - 结点的顺序将反映出结点之间的逻辑关系
    - 方法
        - 完全二叉树
            - 将结点从上到下，从左到右，从 1 开始编号，然后按照编号从小到大的顺序存入一维数组中。
            - 一维数组中下标为 0 的存储单元空闲
            - 荣正 148 页结点之间序号的关系
        - 普通二叉树
            - 需要添加一些结点来构成完全二叉树，再按照完全二叉树的顺序存储方法来存储
            - 造成空间的浪费
- 链式存储结构
    - 熟悉 149 页二叉链表结构体的定义
    - 在具有 n 个结点的二叉树中，一共有 2n 个指针域，其中，n-1 个用来指示结点的左、右孩子，其余的 n+1 个指针域为空
        - 一共 2n 个指针域：因为二叉树有 n 个结点，每个结点都有左右两个指针
        - n-1 个指针域用来指示结点的左、右孩子：除了树根外，其他每个结点都被一个指针指向

        - n + 1 个指针域为空：2n - （n - 1 ） = n + 1
- 二叉树的建立
    - 顺序存储较简单，只须将二叉树各个结点的值按原有的逻辑关系送入相应的向量单元中即可
    - 链式存储
        - 按照完全二叉树的层次顺序，依次输入结点信息来建立二叉链表，必要时加入虚结点，确保形成完全二叉树
        - 荣正 150 页，建立方法，注意用到的数据结构为队列，了解该队列是怎样操作的，认真读代码

四．二叉树的遍历

- 二叉树的深度优先遍历
    - 仅仅考虑三种顺序：DLR，LDR，LRD
    - 三种递归遍历
        - 先序遍历
        - 中序遍历
        - 后序遍历
        - 好好理解一下荣正 152 页的图 5-8
- 二叉树的广度优先遍历
    - 这是一个非递归遍历方法
    - 用到的数据结构是队列
    - 可以用于求二叉树的宽度
- 深度优先遍历的非递归算法
    - 用到的数据结构是栈
    - 算法时间复杂度为 O(n)
    - 荣正 155 页非递归算法的代码一定要会
- 从遍历序列恢复二叉树
    - 考察手动模拟
    - 只能由先序序列和中序序列，或者由中序序列和后序序列，才能还原出二叉树，由先序序列和后序序列，不可还原出二叉树
- 遍历算法的应用

- 统计一棵二叉树中叶子结点数
- 求二叉树的深度：书上给的是递归，后面做题时候要记住非递归算法
- 表达式与二叉树的关系：951 算法题没出过，所以要尽量熟悉该代码

五．树和森林

- 树的存储结构
  - 注意和之前二叉树的存储结构相区分，这里讲的是普遍的树
  - 常用的三种方法
    - 双亲表示法：静态链表实现
    - 孩子表示法：每个结点建立一个孩子链表，类似后面要说的图的邻接表
    - 孩子兄弟表示法
- 树、森林和二叉树之间的转换
  - 转换的唯一性
  - 树与二叉树的转换
    - 荣正 161 页
    - 注意，任何一棵树转化为二叉树后，二叉树的右子树为空
  - 森林转化为二叉树
    - 先将森林中的每一棵树转换为二叉树（此时各二叉树都没有右子树）
    - 再讲这些二叉树合并为一棵二叉树
  - 注：二叉树转换为森林的过程中，将二叉树拆分为多个二叉树后，这些二叉树还要还原为普通的树

六．线索二叉树

- ltag 和 rtag 值为 0 和分别代表的含义
- n 个结点的二叉树有 n+1 个空指针域，这些空指针域用来存储线索
- 线索二叉树中，一个结点是叶子结点的充要条件：结点的左、右线索标志均为 1
- 中序线索二叉树中，查找中序前趋和中序后继很容易
- 先序线索二叉树中，查找后继简单
- 遍历中序线索二叉树的时间复杂度为 O(n)

七．二叉树的应用

- 哈夫曼树
  - 基本概念
    - 树的路径长度和树的带权路径长度相区分
    - 哈夫曼树又称为最优二叉树
    - n 个结点形成的哈夫曼树不唯一
  - 哈夫曼树的构造
    - n 个结点形成二叉树，需要进行 n-1 次合并，产生 n-1 个新结点，得到的哈夫曼树有 2n-1 个结点
    - 严格二叉树：二叉树中没有度为 1 的结点
    - 除了会构造二叉哈夫曼树，之后做题遇到多叉哈夫曼树，也要掌握
    - 根据荣正 171 页，左小右大原则
  - 哈夫曼编码与哈夫曼树译码
    - 前缀码概念
    - 要求能根据哈夫曼树进行编码与解码
- 二叉排序树
  - 二叉排序树的概念
    - 荣正 175 页概念
    - 对二叉排序树进行中序遍历时，所得到的中序序列是递增的，这条可以出小题
  - 二叉排序树的构造：即插入过程
  - 二叉排序树中的结点删除
    - 待删除结点为叶结点
    - 待删除结点仅有左子树或右子树
    - 待删除结点既有左子树又有右子树：两种删除方式，选择哪种，要根据题目是否要求删除结点之后不能改变树的深度

八．小结

- 哈夫曼树
    - 根据结点权值（注意题目对权值的表达，有时候是字符出现的次数，或其他表示方式），利用左小右大原则，构造哈夫曼树，并对各结点字符进行编码，然后给出 01 序列，根据所构造的哈夫曼树进行解码，写出解码后的字符序列
- 二叉排序树：画出删除结点后的二叉树
- 唯一性
    - 树、二叉树，森林之间的转换唯一
    - n 个结点构成的哈夫曼树不唯一
- 根据先序和中序，或者中序和后序，还原二叉树
- 算法
    - 求二叉树深度：递归和非递归
    - 判断二叉树是否对称：形状和结点值都要对称
    - 求二叉树宽度：利用层次遍历

图

一．图的基本概念

- 荣正 183 页
    - 完全无向图的概念，以及顶点数 n 与边数 e 的关系
    - 完全有向图的概念，以及顶点数 n 与边数 e 的关系
    - 顶点度的概念
    - 一个图中总度数等于总边数的 2 倍
        - 无向图中，一条边可以让它两端的结点各加一个度
        - 有向图中，一条边可以让它两端的结点，其中一个加出度，另一个加入度
    - 连通
        - 无向图中，两个顶点之间有路径，则称这两个顶点连通
        - 有向图中，两个顶点之间有“来和回”的路径，则称这两个顶点是强连通的

        - 注意：有路径，不一定是这俩顶点之间存在一条边，而是可以存在“中转站”，比如，A——B——C，其中 A 和 C 没有直接相连，但是它俩是连通的

    - 连通图和连通分量：荣正 184 页概念

    - 强连通图和强连通分量：荣正 184 页概念

    - 求强连通分量的算法：Tarjan 算法，有兴趣可以看 B 站讲解

    - 补充：弱连通也叫连通。做题时候会遇到这种情况：有向图，但是不是强连通图，如果把所有的有向边看成无向边，那么这个图是连通的，此时称为弱连通。所以，题目中虽然是有向图，但是有时候可以只称为连通，不一定强连通

二．图的存储方法

- 常用存储方法
    - 邻接矩阵，邻接表，十字链表，多重邻接表

    - 本书重点讨论邻接矩阵和邻接表
- 邻接矩阵

    - 建立邻接矩阵算法的时间复杂度：$O(n^2)$

    - 当邻接矩阵是一个稀疏矩阵时，造成空间浪费

    - 一个图的邻接矩阵是唯一的
- 邻接表

    - 由顶点表和邻接链表组成

    - 荣正 187 页和 188 页，邻接表的结构体定义，以及建立邻接表的算法，一定要熟悉，2020 的真题最后一道算法题考到相关知识了，这里易错点就是两个结构体的域容易混淆

    - 荣正 188 页的邻接表画法，顶点表下标从 1 开始，邻接链表下标从 0 开始，这样画并不清晰，所以考试时候可以将顶点表和邻接链表的下标统一

    - 建立邻接表的时间复杂度：O(n+e)

    - 一个图的邻接表不唯一

    - 判断两个顶点之间是否存在一条边的时间复杂度
        - 邻接矩阵：O(1)，随机存取判定即可

        - 邻接表：O(n)，扫描其中一个顶点的整个邻接链表

    - 求图中边的数目时间复杂度
        - 邻接矩阵：$O(n^2)$，要检测整个邻接矩阵
        - 邻接表：O(n)，对每个边表中的结点个数计数即可，总数除以 2 就是总边数

三．图的遍历

- visited 数组的使用，为了避免对同一顶点的重复访问
- 遍历算法对有向图和无向图都适用，荣正课本只讨论了无向图的两种遍历
- 深度优先搜索遍历
    - 类似于树的先序遍历，是树先序遍历的推广
    - 是递归算法
    - 深度优先搜索遍历序列不唯一
    - 利用邻接矩阵存储
        - 荣正 189 页代码要看
        - 时间复杂度：$O(n^2)$
        - 空间复杂度：O(n)
    - 利用邻接表存储
        - 荣正 190 页代码要看
        - 时间复杂度：O(n+2e)
        - 空间复杂度：O(n)
- 广度优先搜索遍历
    - 类似于树的层次遍历
    - 是非递归算法
    - 广度优先搜索遍历序列不唯一
    - 利用邻接矩阵存储
        - 荣正 192 页代码要看
        - 时间复杂度：$O(n^2)$
        - 空间复杂度：O(n)
    - 利用邻接表存储
        - 荣正 192 页代码要看
        - 时间复杂度：O(n+2e)

        - 空间复杂度：O(n)
- 注：两种遍历都用到了 visited 数组，以避免对同一顶点重复访问；另外，注意代码和算法描述，要熟悉

四．生成树和最小生成树

- 生成树（泛指的生成树，不单指最小生成树）定义：一个包含了图中所有顶点的树
- 注意：树是指一个无回路存在的连通图，因此，当我们说最小生成树的时候，指的是连通图的最小生成树，注意连通
- n 个顶点的连通图，生成树包含 n-1 条边
- 生成树是原图的一个极小连通子图
- 得到生成树，要用到遍历算法，因此产生了深度优先搜索生成树和广度优先搜索生成树
- 连通图的生成树不唯一
- 构造最小生成树的方法（上述的概念都是泛指的生成树，接下来讨论的才是最小生成树）
    - Prim 算法和 Kruskal 算法都是针对无向图的
    - Prim 算法和 Kruskal 算法中的图是利用邻接矩阵存储的
    - Prim 算法时间复杂度：$O(n^2)$
    - Kruskal 时间复杂度：一般情况下是 O(elbe)，如果边按权值从小到大存到数组中，时间复杂度为 O(e)
    - Prim 算法适用于稠密图
    - Kruskal 算法适用于稀疏图

五．最短路径

- Dijkstra 算法时间复杂度：$O(n^2)$；空间复杂度：O(n)
- Floyd 算法时间复杂度：$O(n^3)$；空间复杂度：$O(n^2)$
- 两个算法都是利用邻接矩阵存储图
- 两个算法都用于有向图
- 本节知识主要考察手动模拟，故算法流程一定要搞清楚
- 迪杰斯特拉的答题方法，参考荣正 203 页的表格，考试时候是让填写部分数据
- 迪杰斯特拉在 20 和 21 两年真题出过手动模拟，弗洛伊德目前没有考过

六．拓扑排序

- 用于有向图

- AOV 网中有环时，无法得到拓扑序列
- 利用拓扑排序或者深度优先搜索遍历，可以判断有向图中是否有环
- 利用邻接矩阵存储图时，拓扑排序时间复杂度：$O(n^3)$
- 利用邻接表存储图时，拓扑排序时间复杂度：O(n+e)
- 拓扑排序序列不唯一
- 拓扑排序依然是以手动模拟的方式考察，所以要熟悉荣正课本上的算法流程，算法的描述要看，代码中栈的操作顺序也要看。另外，荣正 211 页的图比较晦涩，群里会发视频讲解

七．关键路径

- 用于有向图
- 关键路径可能有多条
- 提前包含在所有关键路径上的关键活动，才一定能加快工程的进度
- 接上条，因为图中可能有多条关键路径，若将其中的某些关键活动提前，则这些关键活动所在的路径可能变为非关键路径。所以必须将所有关键路径上同时包含的活动提前，才能加快工程进度
- 求关键路径算法的时间复杂度：O(n+e)
- 本节知识也是以手动模拟的方式考察，而且计算量较大，要认真对待

八．总结

- 复杂度
    - 建立邻接矩阵的时间复杂度：$O(n^2)$
    - 建立邻接表的时间复杂度：O(n+e)
    - 利用邻接矩阵存储图，深度优先遍历时间复杂度为 $O(n^2)$，空间复杂度为 O(n)
    - 利用邻接表存储图，深度优先遍历时间复杂度为 O(n+2e)，空间复杂度为 O(n)
    - 利用邻接矩阵存储图，广度优先遍历时间复杂度为 $O(n^2)$，空间复杂度为 O(n)
    - 利用邻接表存储图，广度优先遍历时间复杂度为 O(n+2e)，空间复杂度为 O(n)
    - Prim 算法时间复杂度：$O(n^2)$
    - Kruskal 时间复杂度：O(elbe)
    - Dijkstra 算法时间复杂度：$O(n^2)$；空间复杂度：O(n)
    - Floyd 算法时间复杂度：$O(n^3)$；空间复杂度：$O(n^2)$
    - 利用邻接矩阵存储图时，拓扑排序时间复杂度：$O(n^3)$

- 利用邻接表存储图时，拓扑排序时间复杂度：O(n+e)
- 求关键路径算法的时间复杂度：O(n+e)

- 唯一性
  - 一个图的邻接矩阵是唯一的
- 不唯一性
  - 一个图的邻接表不唯一
  - 深度优先搜索遍历序列不唯一
  - 广度优先搜索遍历序列不唯一
  - 连通图的生成树不唯一
  - 拓扑排序序列不唯一

索引结构与散列技术

一．索引结构

- 索引结构包括两部分：索引表和数据表
- 索引表中每一项称为索引项，格式为（关键字，地址）
- 索引表中关键字有序，数据表中关键字有序无序均可
- 根据数据表中关键字是否有序，可分为索引顺序结构和索引非顺序结构
- 线性索引
  - 索引非顺序结构对应稠密索引，索引顺序结构对应稀疏索引
  - 索引结构上的检索步骤
    - 1.查找索引表
    - 2.若索引表上存在该记录，则根据索引项的指示在数据表中读取数据；否则说明数据表中不存在该记录
    - 注：有判断题：只要查找索引表，一定能找到目标数据。这个是错的
  - 分块查找
    - 平均查找长度分两部分：索引表的平均查找长度+数据表的平均查找长度
  - 多级索引
- 倒排表：P221 页的优点+P231 第 4 题题干缺点
- 索引结构一般考小题：分块查找如何分块能使平均查找长度最小

二．散列技术

- 散列函数构造方法
    - 数字选择法
    - 平方取中法
    - 折叠法
    - 除留余数法：最重要
    - 基数转换法
    - 随机数法
- 解决冲突的几种方法（与散列函数构造方法相区分）
    - 开放地址法
        - 线性探查法
        - 二次探查法
        - 随机探查法
    - 拉链法
- 查找及分析
    - 查找成功平均查找长度的计算：除数为关键字的个数
    - 查找失败平均查找长度的计算：除数为能映射到的地址的个数
    - 对于查找失败，如果表长大于能映射到的地址个数，那么再寻找空位置时，比较过程中，所比较的地址是可以超过散列函数所能映射到的地址范围的，见荣正 P226+P230
    - 查找效率与散列函数和解决冲突的方法有关，但是平均查找长度与装填因子有关，注意是否有“平均”二字
- 散列技术一般以手动模拟的方式考察：用线性探查法或用拉链法计算平均查找长度

缩小规模算法

- 分治与递归算法
    - 递归算法能解决的问题
        - 排列问题

        - 整数划分问题
    - 分治算法能解决的问题
        - 二分搜索技术：查找一个已经排好序的表的最好方法
            - 折半查找成功时进行的比较次数最多不超过树的深度
        - 归并排序
            - 时间复杂度为 O(nlbn)
            - 空间复杂度均为 O(n)
        - 快速排序
            - 时间复杂度为 O(nlbn)，空间复杂度为 O(n)
            - 待排序文件的记录有序或基本有序，快速排序蜕化为起泡排序
- 动态规划
    - 与分治法不同的是，适用于动态规划求解的问题，经分解得到的子问题往往不是互相独立的
    - 能解决的问题
        - 矩阵连乘问题
        - 0-1 背包问题
        - 硬币找零问题（贪心算法也可以）
    - 基本要素
        - 最优子结构
        - 重叠子问题
    - 变形：备忘录方法
    - 动态规划递归方式：自底向上
    - 备忘录方法递归方式：自顶向下
    - 应用
        - 图像压缩最优搜索二叉树
            - 时间复杂度和空间复杂度均为 $O(n^2)$
    - 贪心算法
        - 能解决的问题
            - 背包问题
            - 硬币找零问题（动态规划也可以）
        - 应用
            - 哈夫曼编码
            - 单源最短路径

- 第八章，2020 考了贪心算法的描述与简单计算，其他年份都只考小题，大家复习重点就是了解这些算法有什么特征，哪种问题应该用哪种方法解决。另外，贪心算法和动态规划的描述，尽量背下来