Data-Structure

记录考研数据结构的代码

Chapter 2 Link List

1. 在循环双链表中，头结点的prior也要指向尾结点，尾结点的next要指向头结点
2. 静态链表要求预先分配一大片连续的内存空间，指针指的是数组中的相对位置
3. 头指针或尾指针是用来指明链表中的第一个结点或最后一个结点，如果只有尾指针是无法访问到单链表的头结点的！！！（不是都是从头开始要看指针）
4. 头结点是在第一个结点前的一个结点，但无论有没有头结点都有头指针都指向第一个结点

Chapter 3 Stack and Queue

1. 注意栈顶指针是指向第一个元素还是第一个元素前面一个元素
2. n个元素进栈，出栈元素不同排列的个数为

   $$C_{2n}^{n} * \frac{1}{n+1}$$

   卡特兰数
3. 采用共享栈节省存储空间，降低发生下溢的可能
4. 栈用于递归、迷宫求解、括号匹配、深度优先、进制转换
5. 队列用于缓冲区、CPU竞争资源（页面替换算法）
6. 对于同一个问题的递归问题求解和非递归问题求解，非递归算法的效率通常高一些，递归求解效率低，但代码量少
7. 在计算输入输出受限的双端队列时，先划出一个道路然后右边可以进可以出，只在左边进行受限的限制
8. 十字链表和三元组表适合稀疏矩阵
9. 在计算矩阵元素在数组中下标时一定要注意数组是否从0开始！！！，如果说存入C语言的数组就默认从0开始！！，还要注意是行优先还是列优先！！！

Chapter 4 String

1. ```
   * 朴素模式匹配算法 模式串（可能存在的子串）
   * 若模式串长度为m，主串长度为n，则
   * 匹配成功的最好时间复杂度为:O(m)
   * 匹配失败的最好时间复杂度为:O(n-m+1)=O(n-m)~O(n)
   * 最坏时间复杂度：最多需要（n-m+1）*m O(nm)
   ```

2. ```
   * 获得next数组
   * 串的前缀：包含第一个字符，且不包括最后一个字符的子串
   ```

* 串的后缀：包含最后一个字符，且不包含第一个字符的子串
* 当第j个字符匹配失败，由前1~j-1个字符组成的串记为S，则：next[j]=S的最长相等前缀后缀长度+1
* 特别地next数组放弃了第一个next[0]
* next[1]=0,next[2]=1
* next[1]=0的原因是表示模式串应该右移一位，主串当前指针后移一位再和模式串的第一个字符进行比较
* 获得nextval数组
* 提出的nextval数组的目的在于如果返回的j值所对应的字母与之前的字母相同就不应该多跳转一回，应当直接赋给之前的字母所对应的next值
* 用nextval数组可以优化KMP算法
* 注意位序是从0开始还是从1开始，如果从0开始就全部减一
*KMP算法平均时间复杂度为O（n+m）

Chapter 5 Tree and Binary Tree

1. 树的性质：
    - 结点数 = 总度数 + 1（根节点）
    - 度为m的树和m叉树的区别是

      前者：（先有结点再定义）

        1. 任意结点的度<=m，
        2. 至少有一个结点度数=m
        3. 一定是非空树，至少有m+1个结点

      后者：（先定义后有结点）

        1. 任意结点度数<=m
        2. 允许所有结点的度都<m
        3. 可以是空树
    - 度为m的树第i层至多有m^(i-1)个结点
    - m叉树第i层至多有m^(i-1)个结点
    - 高度为h的m叉树至多有(m^h-1)/(m-1)个结点
    - 高度为h的m叉树至少有h个结点
    - 高度为h，度为m的树至少有h+m-1个结点
    - 有n个结点的m叉树的最小高度为[logm(n*(m-1)) + 1]向上取整（只有小数就加一）
2. 几种二叉树：
    - 满二叉树：一颗高度为h，且含有2^(h-1)个结点的树
        1. 只有最后一层有叶子结点
        2. 不存在度为1的结点
        3. 按层序1开始编号，结点i的左孩子为2*i*，右孩子为2i+1;结点i的父节点为[i/2]（向下取整）
    - 完全二叉树：当且仅当其每个结点都与高度为h的满二叉树中编号为1~n的结点一一对应时，称为完全二叉树
        1. 只有最后两层可能有叶子结点

2. 最多只有一个度为1的结点
3. 按层序1开始编号，结点i的左孩子为2*i*，右孩子为2i+1;结点i的父节点为[i/2] （向下取整）
4. 当i<=[n/2] （向下取整）为分支节点，i>[n/2]为叶子结点

- 二叉排序树：一颗二叉树或空二叉树

1. 左子树上所有结点的关键字均小于根节点的关键字
2. 右子树上所有结点的关键字均大于根节点的关键字
3. 左子树和右子树又各是一颗排序二叉树

- 平衡二叉树：树上任一结点的左子树和右子树的深度之差不超过1

3. 二叉树常考性质：
   - 设非空二叉树中度为0、1和2的结点个数分别为n0、n1、n2则n0 = n2+1
   - 二叉树的第i层最多有2^(i-1)个结点
4. 完全二叉树常考性质
   - 具有n个结点的完全二叉树的高度h为log2(n+1)向上取整， log2（n）+1向下取整
5. 叶子结点也可以是根节点
6. 当tag为0时，先序线索二叉树找不到先序前驱
7. 当tag为0时，后序线索二叉树找不到后序后继
8. 双亲表示法和孩子表示法：顺序存储结点数据，前者保存父节点在数组中的下标，后者保存孩子链表头指针，前者找父节点方便找孩子不方便，后者找孩子方便，找父节点不方便
9. 孩子兄弟表示法：用二叉链表存储树——左孩子右兄弟 常用于树与二叉树的相互转换
10. 森林与二叉树的转化：森林中各个树的根节点之间视为兄弟关系，用二叉链表存储森林-左孩子右兄弟
11. 二叉排序树（BST） 平衡二叉树（AVL） 查找长度（ASL）
12. 二叉排序树的查找效率主要取决于树的高度，平衡二叉树的平均查找长度ASL为O（log2n）,当只有一个右子树时，平均查找长度为O（n）
13.

树	**森林**	**二叉树**
先根遍历	先序遍历	先序遍历
后根遍历	中序遍历	中序遍历

14. 平衡二叉树深度为h含有的最少结点n(h) = n(h-1) + n(h-2) + 1 n0=1,n1=1,n2=2
15. 平衡二叉树AVL查找时间复杂度不超过高度h
16. 平衡二叉树插入后调整平衡从最小的不平衡树开始，若是LL和RR就调整该结点的L或者R结点使之平衡，若是LR或RL则调整该结点的LR孩子或者RL孩子使之平衡
17. 哈夫曼树也称最优二叉树
18. 哈夫曼树中的结点的带权路径长度：从树的根到该结点的路径长度与该结点上权值的乘积
19. 树的带权路径长度：树中所有**叶结点**的带权路径之和（WPL）
20. 前缀编码用哈夫曼树得出
21. 哈夫曼树可以有很多个，但是带权路径长度最小值唯一

Chapter 6 Graph

1. ```
   /*邻接矩阵法
    * 空间复杂度O(|V|^2)只和顶点数有关与实际边数无关
    * 适合用于存储稠密图，不适于稀疏图
    * 可以用压缩矩阵
    * 表示方式唯一
    * 要找相邻的边必须遍历对应行或列
    */
   ```

2. ```
   /*邻接表法
    * 空间复杂度（有向图） O(|V|+|E|) 无向图O(|v|+2|E|)
    * 表示方式不唯一
    * 存储稀疏图
    * 找有向图的入边很不方便要遍历整个表
    */
   ```

3. ```
   /*十字链表法
    * 只能用于存储有向图
    * 空间复杂度O(|V|+|E|)
    */
   ```

4. ```
   /*临界多重表
    *用于存储无向图
    *顶点结点：data + firstEdge（与该顶点相连的第一条边）
    *边结点： i + j + info + iLink（依附于顶点i的下一条边） + jLink(依附于顶点j的下一条边)
    *空间复杂度O（|V|+|E|）
    *删除边、删除节点等操作很方便
    */
   ```

5. Prim算法适于边稠密图 时间复杂度为O（|V|^2）从某一个顶点开始构建，每次将代价最小的新顶点纳入生成树
6. Kruskal算法 每次选择一条权值最小的边 使这条边的两头连通 时间复杂度O（|E|log2|E|）
7. 连通分量是指无向图中的极大连通子图（为本身图不是连通的设计的）
8. 生成树是包含所有顶点的极小连通子图（保证了连通，极小边数）
9. 无向图的度是依附于该顶点边的数目，有向图边的数目是入度+出度
10. 对于带权图的邻接矩阵，0和无穷都可以用于表达边不存在 无穷更多用于表达对角线上的值不存在
11. 子集不一定能构成子图，要保证子集本身先是个图
12. 强连通分量就是有向图的极大连通子图
13. 深度优先遍历可以用来判断图里是否存在回路，广度优先各访问序列之间没关系
14. 邻接表：
    1. 深度优先遍历 时间复杂度O（|V|+|E|） 空间复杂度O(|V|)
    2. 广度优先遍历 时间复杂度O（|V|+|E|） 空间复杂度O(|V|)
    3. 广度优先生成树和深度优先生成树不唯一
15. 邻接矩阵

    1. 深度优先遍历 时间复杂度O（|V|^2） 空间复杂度O（|V|）
    2. 广度优先遍历 时间复杂度O（|V|^2） 空间复杂度O（|V|）
    3. 广度优先生成树和深度优先生成树唯一
16. 单源最短路径（一对多）：
    1. BFS（无权图）
    2. Dijkstra（有权图，无权图）时间复杂度与Prim算法相同都是O(|V|^2) 不能用于带有负权值的图 因为在计算最短路径的时候 是只能增不能减的
17. 各顶点之间最短路径：Flord（有权图，无权图）时间复杂度O（|V|^3） 空间复杂度O（|V|^2）不能解决负权回路的图
18. 有向无环图：DAG(Directed Acyclic Graph)
19. AOV网（Activity On Vertex）一定是个DAG图
20. 拓扑排序
    1. 从AOV网中选择一个没有前驱的顶点
    2. 从网中删除该顶点和所有以它为顶点的有向边
    3. 重复1,2直到AOV网为空或当前网中不存在无前驱的顶点为止
21. 逆拓扑排序
    1. 从AOV网中选择一个没有后继的顶点
    2. 从网中删除该顶点和所有以它为终点的有向边
    3. 重复1,2直到AOV网为空或当前网中不存在无前驱的顶点为止
    4. 采用邻接表是很低效的 而邻接矩阵很方便 或者使用逆邻接表
22. 拓扑排序和逆拓扑排序序列不唯一
23. 图中有环，则不存在拓扑排序序列或逆拓扑排序序列
24. AOE网（Activtiy On Edge NetWork）
25. 从源点到汇点的路径可能有多条，具有最大路径长度的路径成为关键路径，而关键路径上的活动称为关键活动，完成整个工程的最短时间就是关键路径的长度，若关键路径不能按时完成，则整个工程的完成时间就会延长
26. 关键活动耗时增加，整个工程的完成时间延长
27. 若缩短关键活动的时间，可以缩短整个工程的工期
28. 当缩短到一定程度时，关键活动可能变成非关键活动
29. 可能存在多条关键路径，只提高一条关键路径上的关键活动的速度并不能缩短整个工程周期，只有加快那些包含在所有关键路径上的关键活动才能达到缩短工期的目的。
30. 最小生成树是权值之和最小的生成树，如果没有权值相同的边，寻找过程唯一，使用不同的算法得到的最小生成树也是唯一的
31. 如果有权值相同的边，最小生成树不一定唯一（不一定起作用）
32. 生成树一定是连通的！！
33. 深度优先遍历和拓扑排序一定能判断有向图是否有回路，关键路径有争议，求关键路径的前提是无环图，求之前要判断是否有环
34. 在计算拓扑排序之前，会存储当前顶点的入度的计算
35. 用DFS遍历一个有向无环图，并在DFS算法退栈返回时打印结点是逆拓扑排序
36. 若一个有向图具有有序的拓扑排序序列，它的邻接矩阵一定是三角矩阵（只有小->大或者大->小），但如果是一个三角矩阵不能说明拓扑序列唯一，拓扑序列唯一也不能确定唯一有向无环图。

37. 求关键路径是以拓扑排序和逆拓扑排序为基础的
38. 一个活动的最早发生时间=指该活动弧的起点所表示的事件的最早发生时间
39. 一个活动的最晚发生时间=指该活动弧的终点所表示的事件的最晚发生时间与该活动所需时间之差
40. 弧尾就是起始点，弧头是终点

Chapter 7 Search

1. B树又称多路平衡查找树，B树中所有结点的孩子个数的最大值称为B树的阶，通常用m表示
2. 一颗m阶B树有或为空树，或满足如下性质：
    - 树种每个结点最多有m棵子树，至多有m-1个关键字
    - 若根节点不是终端结点，则至少有两棵子树
    - 除根节点外的所有非叶结点至少有[m/2] (向上取整),即至少含有[m-2] (向上取整) - 1个关键字
    - 所有的叶结点都出现字啊同一层次，并且不带信息
    - 终端结点是最后一层的结点
    - 叶子结点是不存在的那一层的结点
    - 根节点的子树在[2,m]内，关键字数在[1，m-1]，其他结点的子树[m/2] （向上取整），到m内，关键字数[m/2] （向上取整）到m-1
    - 对于任意节点，其所有子树的高度都相同
    - 关键字的值：子树0<关键字1<子树1
    - B树的高度不包括叶子结点即失败节点
    - 含n个关键字的m阶B树，最小高度和最大高度是多少
    - h>=logm(n+1)
    - h<=log[m/2] (向上取整)(n+1)/ + 1
    - n个关键字的B树必有n+1个叶子结点
    - B树在插入时每次一定要插入到终端结点，如果插入后超出上限就在[m/2]取整出分裂，然后将[m/2放入父节点]
    - 分裂如果影响到根节点，可能会导致B树的高度加一
    - 删除时
        1. 如果关键字在终端结点，且没有低于下限就直接删除
        2. 若被删除的关键字在非终端节点，则用直接前驱或直接后继来代替被删除的关键字
        3. 直接前驱：当前关键字左侧指针所指子树中的最右下的元素
        4. 直接后继：当前关键字右侧指针所指子树中的最左下的元素
        5. 如果删除终端结点后，低于下限，如果有兄弟用当前节点的后继和后继的后继来填补空缺，如果有左兄弟用当前节点的前驱和前驱的前驱来填补空缺
        6. 如果删除终端结点后，低于下限，如果兄弟都为[m/2]-1就进行合并以及双亲节点中的关键字进行合并

-

3. m阶B+树的性质
   - 每个分支节点最多有m棵子树
   - 非叶根节点至少有两棵子树，其他分支节点至少有[m/2]（向上取整）个子树
   - 叶子结点是最后一层节点（可能包含多个关键字）
   - 结点的子树个数与关键字个数相等
   - 所有叶结点包含全部的关键字以及指向相应记录的指针，叶结点将关键字按大小排序排列，并且相邻叶结点按大小顺序相互链接起来
   - B+树支持顺序查找
   - 所有分支节点仅包含它的各个子节点中关键字最大值以及指向其子节点的指针
   - 根节点的关键字数n在1到m内，其他结点为[m/2]（向上取整）到m内
   - 叶子结点包含全部关键字，非叶结点中出现过的关键字会出现在叶结点

     而B树中各个结点包含的关键字是不重复的
   - 在B+树中，叶结点包含信息，所有非叶结点仅起到索引作用，非叶结点中的每个索引项只含有对应子树的最大关键字和指向该子树的指针，不含有该关键字对应记录的地址
   - B+树的阶更大，树高更矮，读磁盘次数变少
4. 掌握要点：

   $1.B$树的阶数是孩子个数的最大值（包括叶子结点（不存在的））
   $2.B$树的根节点的关键字 $\in [1，m-1]$
   $3.B$树根节点的子树 $\in [2,m]$
   $4.B$树除了根节点外其他的非叶结点的关键字数目 $\in [\lceil (m/2) \rceil -1,m-1]$
   $5.B$树除了根节点外其他的非叶结点的子树数目 $\in [\lceil (m/2) \rceil ,m]$
   $6.B$树只支持多路查找，不支持顺序查找，而$B+$树支持多路查找和顺序查找(因为叶结点
   $7.B+$树是应文件系统而产生的的B树的变形，更适于操作系统的文件索引和数据库索引
   $8.B+$树的叶结点包括了所有的关键字信息，所有非叶结点仅起到索引作用，非叶结点中
   $9.B$树插入的时候要从终端结点开始插入
   $10.$拥有n个关键字的B树的叶结点是$n+1$个(查找失败的情况)
   $11.B+$树每个分支节点最多有m棵子树与B树不同，结点的子树个数与关键字个数相同，其他每个分支节点都至少含有$\lceil (m/2) \rceil$棵子树
   $12.$计算n个关键字的m阶B树的最小高度$(m-1)*(1+m+m^2+...+m^{h-1})=m^h-1$
   $13.$计算n个关键字的m阶B树的最大高度$(1+2+2*\lceil m/2 \rceil +...+$第$h$层$2*(\lceil m/2 \rceil)^{h-2}$
   所以第$h+1$层叶子结点（失败结点）为$2*(\lceil m/2 \rceil)^{h-2})$,由$n$个关键字的树一定有$n+1$个
   $n+1 \geqq 2*(\lceil m/2 \rceil)^{h-2}) \Rightarrow h \leqq log_{\lceil m/2 \rceil}^{\frac{n+1}{2}}+1$
5. 散列表查找
   - 查找成功的平均查找长度
   - 查找失败的平均查找长度

Chpater 8 Sort

1. 插入排序
   - 直接插入排序 最好情况O(n)，空间复杂度O(1) 最差O(n^2) 稳定

- 折半插入排序 时间复杂度O(n^2) 空间复杂度O(1) 稳定
- 希尔插入排序 时间复杂度O(n^1.3) 不稳定 空间复杂度O(1) 不稳定

2. 交换排序
   - 冒泡排序 最坏O（n^2） 平均O(n^2) 稳定 每趟排序都会有个元素放到其最终位置 空间复杂度O（1） 冒泡排序不发生交换时就结束
   - 快速排序（基于分治法）

     > 快速排序 最好O(n*log2^n) 最坏O(n^2) 如果本身有序就是最慢的
     > 如果每一个枢轴都将待排序序列划分为两个均匀的部分，则递归深度最小，算法效率最高
     > 优化时可以改变枢轴元素的选择
     > 快速排序是所有内部的排序算法平均性能最优秀的算法 O(n*log2^n) 稳定性 不稳定
     > 注意一趟排序和一次划分的区别 一次划分可以确定一个元素的最终位置，一趟排序也许可以确定多个元素的最终位置

3. 选择排序
   - 简单选择排序 空间复杂度O（1），时间复杂度O（n^2） 与元素状态无关 不稳定
   - 堆排序 初始堆是O（n）空间复杂度O(1) 时间复杂度O(n*log2^n) 稳定性 不稳定

     插入时要放到表尾巴 删除时要用表尾元素代替再进行下坠

     对堆进行建堆时，选择分支节点 n/2向下取整 从后往前一次遍历

     堆本质上就是一个完全二叉树 采用顺序存储

4.