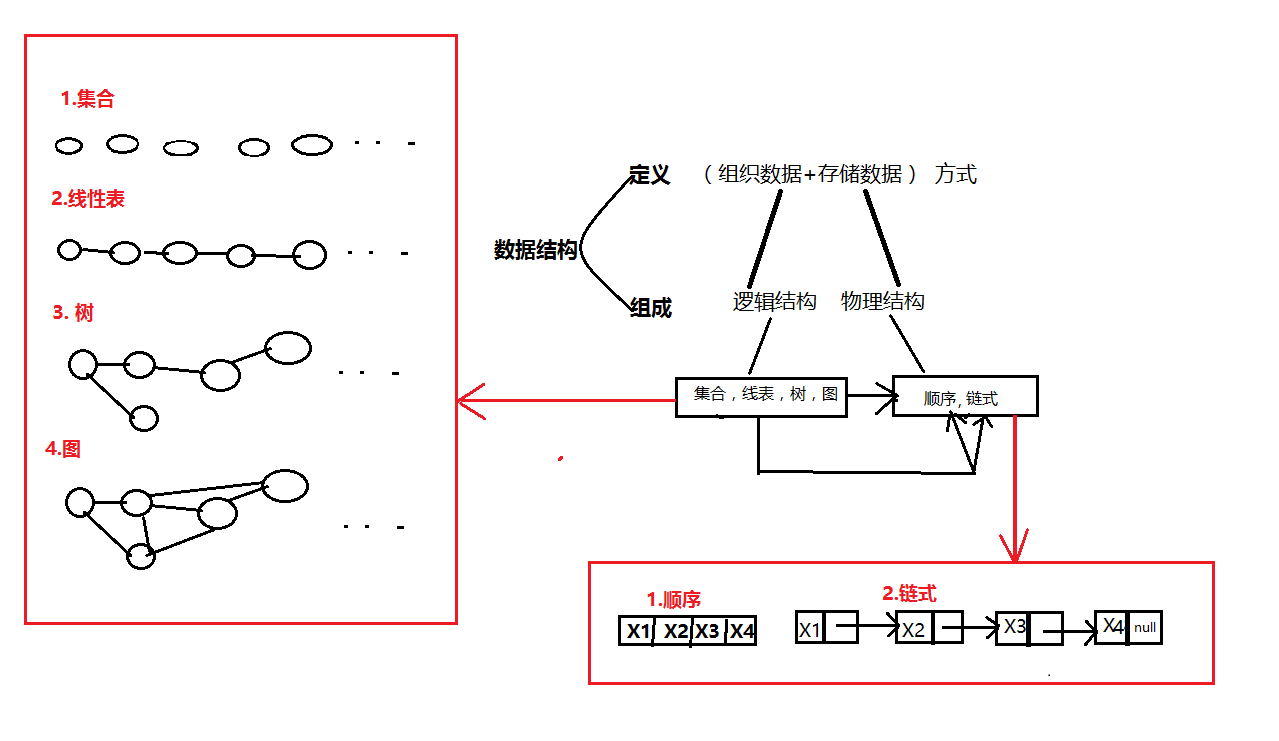
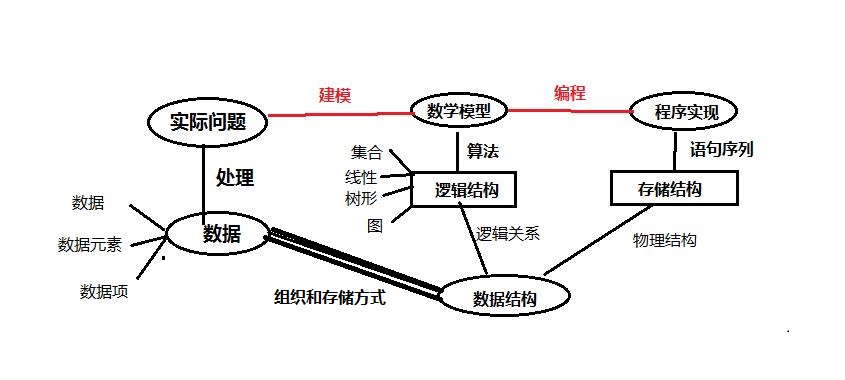
# 数据结构 《简写版》

数据结构的宏观图示:



# 第一章：数据结构基本概念

**数据**：是描述数和字符的集合

**数据元素**：是数据的最小单位，类似于数据库的一行

**数据项：**是具有独立意义的数据的最小单位，类似于数据库的一行的一段

**数据对象**：是指性质相同的数据项的集合，类似于数据库的一列

**数据结构**：是指数据元素和数据元素之间的关系

**存储结构**:顺序存储结构，链式存储结构，索引存储结构，哈希（散列）存储结构

**索引存储**：是指在存储数据的同时，还建立了附加的索引表。

**哈希存储结构：**是在数据的关键字的基础上，通过哈希函数，直接计算出一个数值，将该数据定位该元素 的存储地址。《指适合用快速查找和插入的情况》

**抽象数据类型：**（abstract date type）ADT：是对问题的数据结构和数据的分析。

包括三部分：数据对象、数据关系和基本运算

基本特征：数据抽象和数据封装

数据抽象:强调实体的本质的特征、其所能完成的功能以及它的外部接口

数据封装：对实体进行特点的隐藏

**算法**：是对特定问题求解步骤的一种描述

算法的5个特征：

有穷性：算法的步骤是有穷的

确定性：算法是确定的，不能有二义性

可行性：算法要是可行的，不能是算不出来的

有输入：算法要有输入

有输出：算法要有输出

**算法的分析:**

时间复杂度：分析算法占用cpu的时间

两种方法：**事后统计法**和**事前估计法**

空间复杂度：分析算法占用的cpu的空间

**时间复杂度**

计算算法的频度T（n）：就是原操作的执行次数

时间复杂度O(n)：就是T的最高幂次不带系数

计算递归算法的时间复杂度：变长时空分析：

解决方法：迭代分析：不同的情况进行分析，然后分别求出

**空间复杂度：**算法临时占用的空间

# 第二章：线性表

线性表：零个或多个数据元素的有限序列  
线性表的存储结构：**顺序存储&链式存储**

**顺序表：线性表的顺序存储结构**

**链式表：线性表的链式存储结构**

**有序表：线性表的数据成递增或者递减**

**>>>>二路归并：在处理两个有序表的时候的一种算法**

**：从头开始进行比较，一个一个比较，然后将比较以后较为大的放入新的表，在进行重复的做法。**

顺序表结构：记录长度的变量和存储数据的数组

链式表结构：（每一个节点）存储数据的数据域和表示节点之间的逻辑关系的指针域

链表的分类：单链表、循环链表和双链表

**单链表**：就是一种普通的链表

**双链表**：每一个节点的指针域包含前一个节点和下一个节点的信息的特殊单链表

**循环链表**：尾节点指向首节点的特殊单链表

# 第三章：栈

**栈：**是一种只能在一段进行插入或者删除的**线性表**

**栈的存储方式**：线性、链式

线性栈存储结构：采用顺序存储结构的栈

链式栈的存储结构：采用链式存储结构的栈

线性栈：记录最顶端的位置的**指针**和存储数据的**数组。Top=-1，**因为数组从0开始

链式栈：（每一个节点）有用来存储数据的**数据域**和指向下一个节点的**指针域**

共享栈：将两个栈相加，一个的首为另外一个的尾。

表达式：中缀、后缀和前缀表达式

# 队列

队列：是一种受限制的线性表，指能在一段进行插入，另外一段进行删除（先进先出表）

队列的存储方式：线性、链式

线性队列的存储结构：采用顺序存储结构的队列

链式队列的存储结构：采用链式存储结构的队列

顺序队列：标记队列头和尾的**指针**，保存数据的**数组**

链式队列:

(数据节点)保存数据的变量和记录下一个节点的指针

(链节点)保存队头和队位的指针

**》》队列满了的标志：rear=MAXSIEZ-1**

**循环队列:**将队列的首位进行连接

**》》循环队列满了的标志：队尾指针循环加1等于对头指针**

**》》循环队列是空的标志，对头指针等于队尾指针**

**双端队列:是指两端都可以进行进队个出队操作的队列：先进排在前面，先进先出**

# 第四章：串（字符串）

串：是0个或者是多个字符组成的有限序列

**》》》**串相等：长度相等并且对应位置的字符相等

串的存储结构：顺序和链式

**顺序串**：存储方式为顺序。字符依次存放在一组连续的存储单元里

顺序串的存储方式：非紧缩格式和紧缩式存储格式

**非紧缩式**：一个字节存储一个字符

**紧缩式**：一个字节存放多个字符

**链串：**存储方式为链式。（每一个节点）指针域和数据域

串的匹配模式

目标串：要从该串中寻找指定字符串的字符串**》》》s**串

模式串：用来到s串中寻找的字符串 **》》》t**串

**暴力算法》》》BF》》》Brute-Force**

BF算法：简单匹配算法，从t1和s1开始，要么成功，要么移动到s2，继续从t1开始。

**KMP:针对BF算法的改进版本**

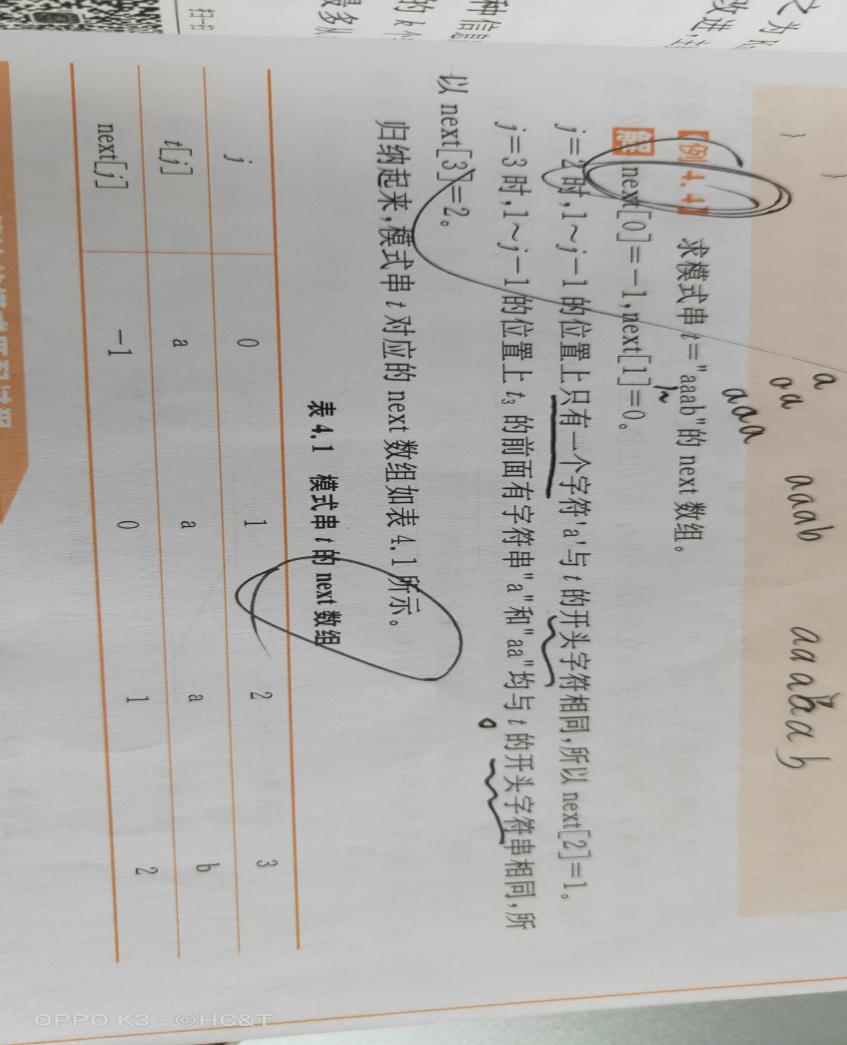
KMP：在BF算法的基础上，加上一个next数组，通过该数组，可以在t串和s串重复度较为高的情况下

快速的完成字符串的匹配。重复度越高，相比于BF算法，更加的快速

**》》》KMP**算法的实现：首先，根据**模式串**的字符模式，写出next数组。

》》》next数组的长度为模式串的字符长度。内容next[0]=-1,next[1]=0(规定的)，

其他的（2~j-1**（j为数组的下标）**），根据t的1~j-1的字符串的子**串**与t字符的从头开始的匹配的字符串的个数

图示：

# 》》》改良版KMP算法

改良版：在原先的基础上，加入一个nextval数组，

》》》nextval数组的值：nextvall[0]=-1

当字符串t【j】=t【next[j]】,nextval[j]=nextval[next[j]]

不相等：nextval【j】=next【j】

# 第五章：递归

种类：**直接递归**和**间接递归**

# **第六章：数组**

一维数组的存储结构：按元素顺序存储到一块连续的内存单元

二维数组的存储结构：按行优先存储和按列优先存储

# 》》》特殊矩阵的压缩

**对称矩阵：二维数组满足 a【j】【i】=a【i】【j】**

**对称矩阵的压缩：**存储上三角和主对角或者下三角和主对角

**上三角矩阵:**下三角均为c常数的n阶矩阵

**下三角矩阵:**上三角均为c常数的n阶矩阵

**上（下）三角矩阵压缩存储：**按照行优先的方法存储，然后在最后一个位置存储常数c

**对角矩阵：**所有非零元素以集中在以主对角线为对称中心的带状区域

**》》》左下角和右下角为0**

**稀疏矩阵：非零元素的数量远少于矩阵的总元素**

压缩的方法： 三元组 和 十字链表

**三元组：**(顺序)存储x 和y，date存储数据

**十字链表**：（链式）：

》》》**头节点**：(两个指针)连接上下和左右，俩个0

》》**》列的头节点只要连接左右的指针**

》》》**数据节点**：行号、列号、值、（两个指针）连接上下和连接左右的

》》》**总头节点**：总行数，总列数，指针

# 广义表

广义表：是线性表的推广，是有限个元素的序列，采用括号表示法为：

GL=（a1，a2，a3）

广义表的特征：

长度：最外层包含元素的个数

深度：包含括弧的重数

广义表的存储结构：只有**链式结构**

节点类型：圆形节点：表示子表 正方形：表示原子

结点结构:

tag:是原子还是子表的标志

Date或者sublist：原子则使用date来存储数据，则子表则采用sublist存储子表的头节点

Link：指针域，存储下一个元素的指针

# 第七章：树

树：是n个有限元素的有限集合

根节点：当n>0的时候，在最上层的结点为 根节点，是整个树的中心

树的逻辑表示方法：图形表示方法，文氏图表示法，凹入表示法，括号表示法

**树结点的相关术语：**

结点的度：该结点的子节点的个数

树的的:该树的所有结点的度的最大值

分支结点：树中结点的度不为0 的结点，也叫非终端结点

路径：一个节点到达另外一个经过的结点

路径长度：一个结点到达另外一个结点经过的结点的个数减1

三结点:孩子节点、兄弟结点、双亲结点

结点层次：从上往下依次增加、以一行为一层

树的高度：该树的层次的最大值

有序树和无序树：如果树结点的子树按照从左往右的次序安排的，并且相对次 序不能随意交换，则是有序树，反之则不是

**树的相关性质：**

树的结点数等于所有结点的度数之和加1

度为m的树的i层，最多有 个结点

高度为h的m次树最有有 个结点

具有n个结点的m次数的最小高度为

**树的遍历：层次遍历、先根遍历和后根遍历**

先根遍历：先访问根结点，在访问根节点的子节点

后根遍历，先访问根节点的字节，在访问根结点

层次遍历：从上往下，一层一层的从左往右遍历

**树的存储方式：存储结点的数据信息，还存储结点之间的逻辑关系**

双亲存储结构:(一种顺序存储结构)，每一个节点存储对应双亲的结点

孩子链存储结构：每一个结点有树的度的个数的指针，用来指向孩子结点

孩子兄弟链：用每一个子节点的第一个结点管理其余的孩子

**二叉树**

二叉树：是一个有限的结点集合，包含根节点、左支数和右支数

性质：非空二叉树上的叶子结点数等于双分支结点数+1

**二叉树和数和森林的转化：**

树转化为二叉树：

兄弟结点之间添加横线，父节点去掉除了长子结点的连接的线，向下45度，ok

森林转化为二叉树

每一个树转化为二叉树，然后把所有的连接起来

二叉树转化为树

将树转化为二叉树的步骤相反实现一次

二叉树转化为森林

将每一个分开，变成多个二叉树，然后每一个依次复原即可

**二叉树的存储结构**

顺序存储结构；一个连续的地址，将每一个从左往右，从上往下，依次标号，按照标号进行存储

链式存储结构：（每一个结点）两个指针域(指向左孩子和右孩子)，数据域，存储数据，形成了叉查 链表

**二叉树的遍历**

先序遍历：根结点-左支树-右支数

中序遍历：左支数-根节点-右支数

后序遍历；左支数-右支数-根结点

层次遍历：从上往下，从左往右，依次遍历

**二叉树的构造：**

根据遍历出来的数据，还原二叉树的结构

**线索二叉树**

先序、中序、后序线索二叉树

根据结点是否有左右结点，对结点的前驱结点和猴急结点进行记录

**哈夫曼树**

哈夫曼树：是带权路径长度wpl（weighted path length）最小的二叉树

**构造哈夫曼树**

过程：选取最小的作为左右结点构造二叉树，然后用生成的二叉树的根结点和剩余的结点，醉成一个 新的二叉树，继续重复该操作，即可

**哈夫曼编码：进行数据压缩**

**并查集：进行元素所属集合的查找和两个元素各自所属元素的合并**

# 第八章：图

**图**：是由两个集合组成（边和顶点）的图形集合

**有向图：**尖括号

**无向图：**边集合有对称性

**图的基本术语**

**端点：**一条边的两个顶点

**邻接点：**一条边的两个端点

**顶点的度、初度和入度：**顶点关联的边叫度、顶点出起点的为出度，顶点为终点的为入度

**完全图：无向图：**两个顶点之间都存在一条边

**有向图：**每两个顶点之间存在着相反的两条边

**路径和路径长度：**顶点到另外一个顶点的经过的边序列

**路径长度：**经过的边的数量或者是路径权值的和

**连通图**：（无向图）任意两个顶点之间是连通的，所有部分是一起的

**连通分量**：连通图则是本身，非连通图有多个

**强连通图**：（有向图）任意顶点之间是连通的

**强连通分量：**（有向图）有向图的极大连通分量

**权和网：**每一条边对应的数值。边上带有权值的图为带权图，也叫为网

**图的存储结构**

**邻接矩阵存储：使用一个二维数组，用来存储图每一个结点之间的关系**

**无向图：**1表示x与y坐标的顶点有连接，反正则没有，成对称图形

**有向图：**数字表示x与y坐标的顶点之间的权值，为0则没有

**邻接表：**对于有向表

**头结点：**指针和数据，指针存储下一个结点，数据域存储顶点的值

**边结点：**数据域：存储结点和权值，指针域：存储下一个结点

**十字链：（有向图）**横向表示该节点的出去的结点的集合，纵向表示该结点的出去的结点

**邻接多重表：无向图的存储**

**图的遍历：深度优先遍历和广度优先遍历**

**生成树和最小生成树**

生成树：连通图的最小连通分量，每一个结点之间有并只有一条路径相邻，增加一个条边，则形成环

最小生成树：边上的权值的和最小的数

求最小生成树的算法：prim算法、克鲁斯卡尔算法（kruskal算法）

**Prim算法：**寻找某一个结点作为起点，然后选取最小的权值的边，然后在将选取的边的终点变成起点， 再重复操作，注意，两个顶点之间只能由一条边

**Kruskal算法：**按照边的权值的大小进行依次选取，直到成功选取顶点数-1条边，在构成环的时候， 则取消这条边

**最短路径**

**路径：**（无权图）一个顶点到达另外一个顶点经过的边s

（有权图）一个顶点到达另外一个顶点的权值之和

**最小路径：**（无权图）一个顶点达到另外一个顶点的最小路径

（有权图）一个顶点到达另外一个顶点的权值之和的最小值

**狄克斯特拉算法(dijkstra)：->>带权有向图**

**--》》》求解一个顶点到其他各个顶点的最小路径**

**实现过程：从第一个结点出发，标出出度的点，记录长度，然后在以终点为起点，继续重复操作，并 且更新第一个顶点到各个顶点之间的距离。**

**弗洛伊德算法(Floyd）->>带权有向图**

**->>>求两顶点之间的最短路径**

**->>>通过一个图的权值矩阵，求出各个顶点对的最短路径**

**实现过程:将所有的结点依次作为中间结点，将起点-中间点-终点和起点-终点的权值长度进行比较，最后**

**将每一个的最短路径进行比较更新，到最后的结点，完成。**