# 时间复杂度

### 定义

4种方法：猜（没招了），递归树（右边有两个T），换元法（有根号()和对数()），主定理

### 性质

# 排序

题目：各种排序的时间复杂度，空间复杂度，是否稳定。三个稳定（归并，冒泡，插入）三个不稳定（快排，堆排，选择）

##### 堆排

最大堆：PARENT > TWO CHILDREN

最小堆：PARENT < TWO CHILDREN

分建堆和维护堆，建堆其实也是循环维护堆的过程，看“3-SortingAlgorithms”的PDF

最大堆和最小堆都是二叉堆

# 动态规划

多编程训练吧，这个套路太多了，不好一一列举，本质也是分治法，划分成子问题然后求解。要会证明最优子结构性质，一般把动态转移方程搞出来以后用反证法证明

# 贪心

哈夫曼编码要会，怎么构造哈夫曼树。

拟阵可以表示成。拟阵有两个性质：遗传性和交换性。遗传性是说如果一个集合，那么任一集合C的子集。交换性是说如果

拟阵中所有的最大独立子集（元素个数最多的子集，并且不可以再扩展）都具有相同的大小（即元素个数都相同）

# 均摊分析

思想：一系列操作，有的时间长，有的时间短，怎么算总操作的平均时间？

例子：二进制加法器和数组大小的动态增减。

证明均摊分析：用总的操作时间f(n) / 操作次数n

# 图

图的存储：邻接表（稀疏图），邻接矩阵（稠密图）

有向图的转置：所有的边反向，结点不变

BFS（广搜）是一个不断入队列再出队列的过程。

广搜和深搜的代码背一下

有向无环图的拓扑排序：如果有边，那么u的排序一定在v之前。

强连通分量，用DFS求解

无向图的最小生成树算法。

最小生成树算法：Kruskal, Prim

单源最短路径：Dijkstra, Bellman-Ford

最大流

字符串匹配

朴素暴力算法

Karp算法，后缀函数，前缀函数，KMP算法

KMP算法虽然理论上很快，是线性时间，但实际执行上不行，真正在日常生活中使用的是Boyer-Moore算法，这个算法强调pattern从右往左比，而不是从左往右，这是根本性的区别，因为这种match可以得到pattern更多的信息。

# 字符串匹配

模型：在匹配串T中找出模式串P

朴素的字符串匹配：从左到右依次比较P和T中的每个字符，如果有不符合的，把P右移1位继续开始。优点：实现简单 缺点：速度太慢，没有利用到匹配失败后的信息来加速匹配过程

前缀不包括最后一位，后缀不包括第一位，即String自身既不是前缀，也不是后缀

Rabin-Karp算法的基本思想：把字符串匹配转换成数字的比较，如果数字相等，那么字符串就匹配上了。字符串转成数字用进制转换的思想，字符串的进制数是（字符集大小），如果数字太大超出了整数的表示范围，则要取模，取模后涉及冲突问题，可以再逐位比较进行二次确认。

KMP算法：重点是计算next数组。next数组规定从1开始，n[i]以i结尾而不是i-1。则n[i]就是最长前缀=最长后缀的substring的长度。

Boyer-Moore算法：P从右向左匹配，而不是从左向右。坏字符和好后缀，当MisMatch发生的时候，哪个位移大就用哪个。坏字符的构造(用字符作下标)：

好后缀的构造(用数字作下标):

# 用已知的NPC证明一个新问题是NPC问题

可以用的NPC：团问题，顶点覆盖问题，哈密顿回路问题，旅行商问题，这四个是图的。电路满足问题，三合取范式，子集和问题

套路：先说明这个问题是个NP问题，再证明这个问题可以转化为一个已知的NPC问题，即你要做的就是弄出一个转化算法。

证明NP：对于给定的一个解，能够在多项式时间内验证这个解是否正确，一般都是用易知这种字眼。。。。。

证明NPC：发明一个算法来转换。

# 近似算法

# 多线程算法

三个关键字：spawn和sync，这个是嵌套的多线程；parallel这个是循环的多线程

在并行化代码中去掉三个关键字就得到普通的串行化算法

# 注意

涉及递归的算法设计题，首先把递归式写出来，这样好分析时间复杂度，也能给自己一个更形式化的描述，能更好的去思考这个问题。如果还涉及动态规划，把动态转移方程写出来。

旅行商问题是有权的，哈密顿回路是无权的，只要经过顶点就可以了

快排，一般选数组的第一个数作为pivot，然后j指向最后一个元素，i指向第一个元素，从右边开始--