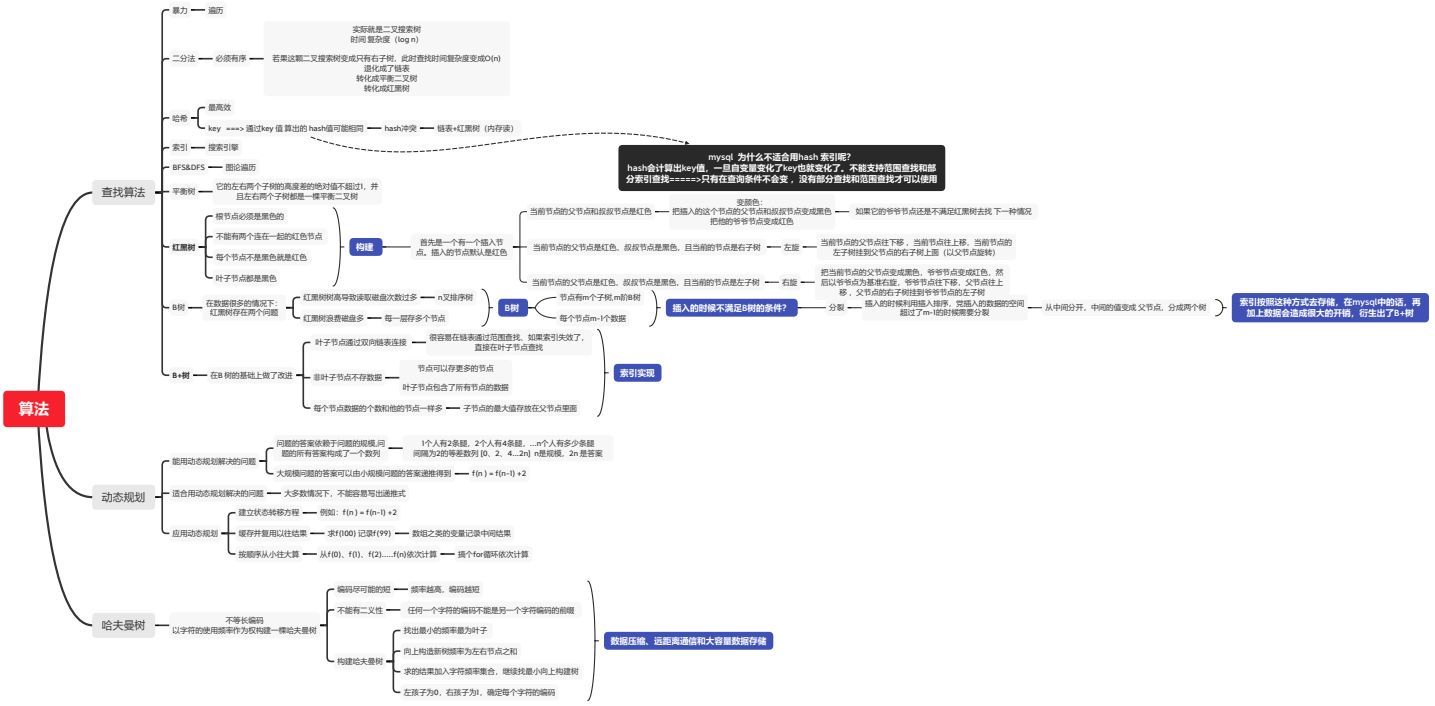
算法
查找算法
暴力
遍历
二分法
必须有序
实际就是二叉搜索树
时间 复杂度（log n）
若果这颗二叉搜索树变成只有右子树，此时查找时间复杂度变成O(n)
退化成了链表
转化成平衡二叉树
转化成红黑树
哈希
最高效
key ===> 通过key 值 算出的 hash值可能相同
hash冲突
链表+红黑树（内存读）
索引
搜索引擎
BFS&DFS
图论遍历
平衡树
它的左右两个子树的高度差的绝对值不超过1，并且左右两个子树都是一棵平衡二叉树
红黑树
根节点必须是黑色的
不能有两个连在一起的红色节点
每个节点不是黑色就是红色
叶子节点都是黑色
构建
首先是一个有一个插入节点，插入的节点默认是红色
当前节点的父节点和叔叔节点是红色
把插入的这个节点的父节点和叔叔节点变成黑色
变颜色：
把他的爷爷节点变成红色
如果它的爷爷节点还是不满足红黑树去找 下一种情况
当前节点的父节点是红色，叔叔节点是黑色，且当前的节点是右子树
左旋
当前节点的父节点往下移，当前节点往上移，当前节点的左子树挂到父节点的右子树上面（以父节点旋转）
当前节点的父节点是红色，叔叔节点是黑色，且当前的节点是左子树
右旋
把当前节点的父节点变成黑色，爷爷节点变成红色，然后以爷爷点为基准右旋，爷爷节点往下移，父节点往上移，父节点的右子树挂到爷爷节点的左子树
B树
在数据很多的情况下：
红黑树存在两个问题
红黑树树高导致读取磁盘次数过多
n叉排序树
红黑树浪费磁盘多
每一层存多个节点
B树
节点有m个子树,m阶B树
每个节点m-1个数据
插入的时候不满足B树的条件？
分裂
插入的时候利用插入排序，觉插入的数据的空间超过了m-1的时候需要分裂
从中间分开，中间的值变成 父节点，分成两个树
索引按照这种方式去存储，在mysql中的话，再加上数据会造成很大的开销，衍生出了B+树
B+树
在B 树的基础上做了改进
叶子节点通过双向链表连接
假设是在链表通过范围查找，如果索引失效了，直接在叶子节点查找
非叶子节点不存数据
节点可以存更多的节点
叶子节点包含了所有节点的数据
每个节点数据的个数和他的节点一样多
子节点的最大值存放在父节点里面
索引实现
mysql 为什么不适合用hash 索引呢?
hash会计算出key值，一旦自变量变化了key也就变化了，不能支持范围查找和部分索引查找=====>只有在查询条件不会变，没有部分查找和范围查找才可以使用
动态规划
能用动态规划解决的问题
问题的答案依赖于问题的规模,问题的所有答案构成了一个数列
1个人有2条腿，2个人有4条腿，...n个人有多少条腿
问隔为2的等差数列 [0、2、4...2n] n是规模，2n 是答案
大规模问题的答案可以由小规模问题的答案递推得到
f(n) = f(n-1) +2
适合用动态规划解决的问题
大多数情况下，不能容易写出递推式
应用动态规划
建立状态转移方程
例如：f(n) = f(n-1) +2
缓存并复用以往结果
求f(100) 记录f(99)
数组之类的变量记录中间结果
按顺序从小往大算
从f(0)、f(1)、f(2).....f(n)依次计算
搞个for循环依次计算
哈夫曼树
不等长编码
以字符的使用频率作为权构建一棵哈夫曼树
编码尽可能的短
频率越高，编码越短
不能有二义性
任何一个字符的编码不能是另一个字符编码的前缀
构建哈夫曼树
找出最小的频率最为叶子
向上构造新树频率为左右节点之和
求的结果加入字符频率集合，继续找最小向上构建树
左孩子为0，右孩子为1，确定每个字符的编码
数据压缩、远距离通信和大容量数据存储