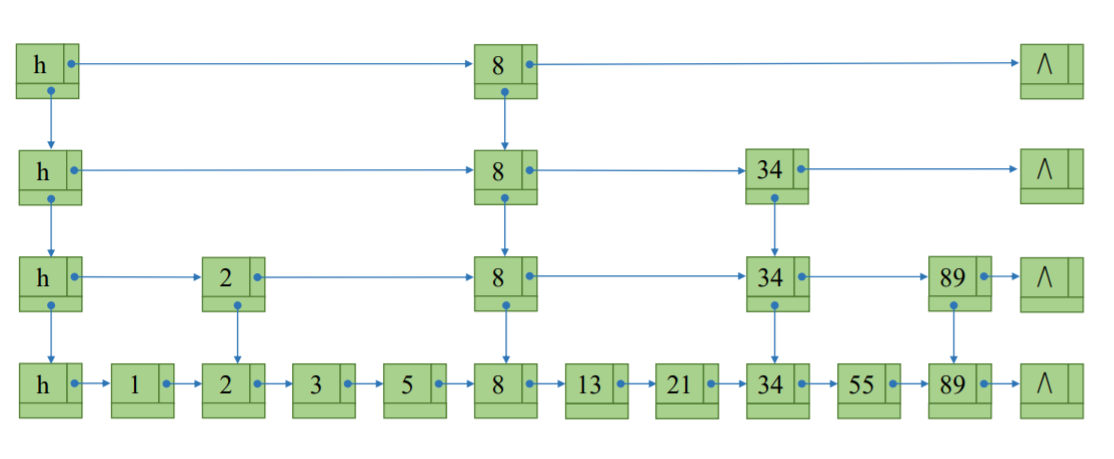
平均查找长度(Average Search Length, ASL)：查找过程中关键字的平均比较次数，类似于计算查找次数的期望值

**静态查找方法**

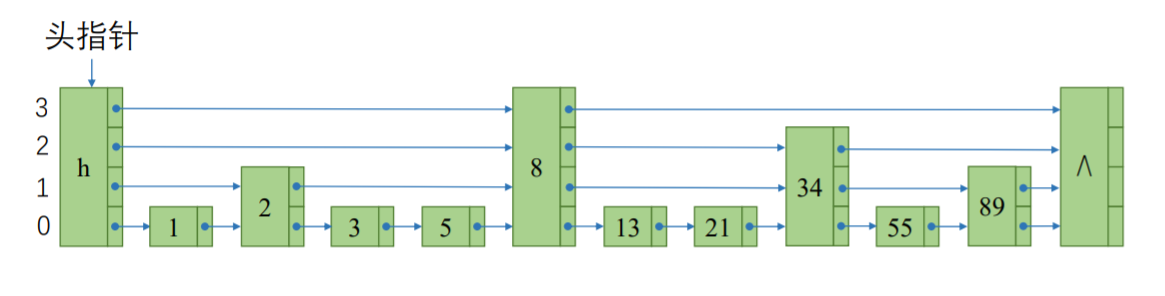
* 顺序查找：ASL（设成功概率p）=(1-p/2)(n-1)，若成功ASL=(n+1)/2≈n
* 折半查找：其过程生成一棵二叉判定树，可补为满二叉树，若成功ASL=[(n+1)/n] log2(n+1)-1≈logn
* 分块查找(Blocking Search)又称索引顺序查找：块间有序，块内无序，建立一个索引表，其结点记录了每个块的最大关键字和起始指针。分块方案最优时，若成功ASL=√n+1≈√n
* 跳跃表/跳表(Skip List)：使链表实现折半查找

建立每个层级的链接链表，从最高层向右查找，若大于该结点且遇到空右结点则向下降一级继续向右查找，若小于该结点则回退一结点再向下降一级继续向右查找

* 四联表节点：设置上下左右四个指针域
* 简化表节点：设置向下向后两个指针域



* 再简化（代码实现）：将各层级的数据域合并，用数组存储不同层级的向后指针域



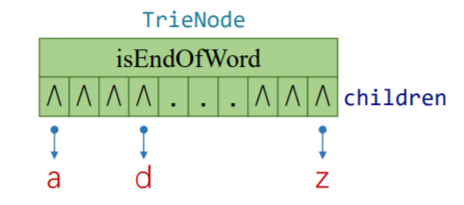
某结点层数的增减随机生成

**动态查找方法**（普遍基于树结构）

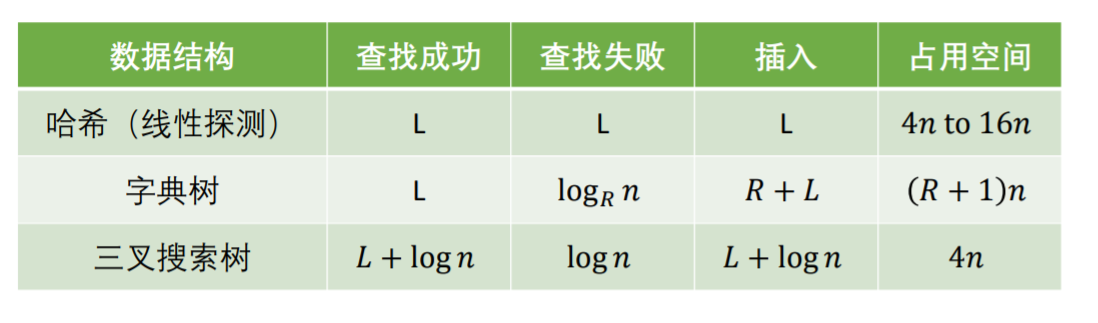
* BST
* 伸展树 (splay tree)：每次查找后利用旋转变换（一字型或之字形）将被该结点移动到根节点。这样最近查询成功的节点均在根节点附近，提高了重复访问效率。
* B-树 (B-tree) （类及其查找函数的代码实现）：（查找插入删除均为O(logn)）

1. m阶/叉：一个节点处最多有(m-1)个数据域和m个指针域（可形象理解为相互间隔）。根节点至少有两个子节点，其他节点至少有 𝑚/2 个子节点。
2. 节点内和节点间均排序。
3. 插入：向节点内插入，若节点内数据数量超过树的阶数，则该节点以中央一个或几个为父节点进行分裂。
4. 删除：删除后需调整。Why？

* 字典树 (trie) （类以及查找插入函数的代码实现）
* 字符串字典树：以字符作为关键字key以数字为值value时，从根节点到叶节点的每一条路径表示一个字符串。缺点之一为空间代价大。



* 三叉搜索树：字符串字典树的改进，将节点字符进行排序生成排序树

（理解？）

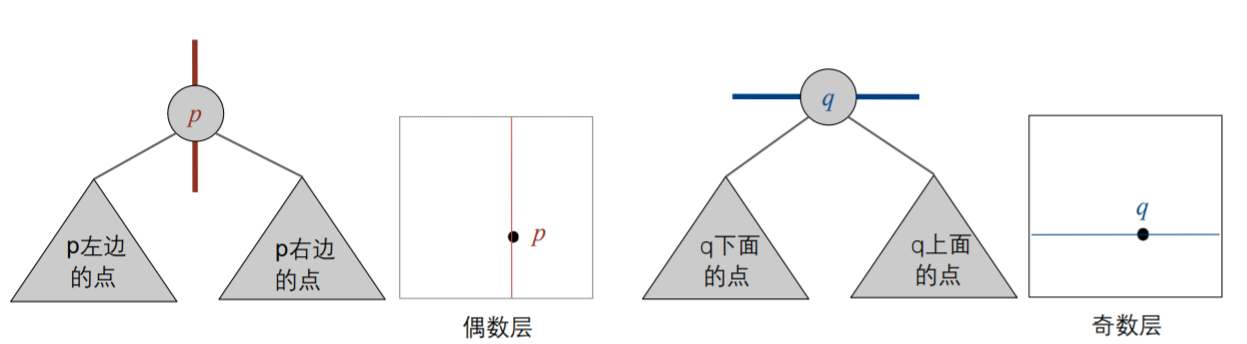
动态查找在几何计算上的应用

* 正交线段相交检测

扫描线算法：一般在x-轴从左到右扫描，遇到左侧端点则将y坐标插入二叉排序树，遇到右侧端点则将y坐标从二叉排序树中删除，遇到垂直片段则在y坐标之间做范围查找。（转二维为一维）

* k-d 树

可用树划分一个二维空间，如二叉空间分割树(BSP tree)。



范围查找复杂度：一般情况O(log n +m)，最差情况O(√n+m)

最近邻查找复杂度：一般情况O(log n)，最差情况O(n)

搜索问题建模：农夫过河问题（搜索树&状态空间图）

无信息搜索算法（基本规律：按总动作代价从小到大遍历）

* 回溯算法
* 深度优先搜索算法（需假设所有动作代价为零）
* 广度优先搜索算法（需假设所有动作代价为常数）
* 迭代深入搜索算法（在一定深度内使用深度优先搜索，再逐步加大深度）
* 一致代价搜索算法

有信息搜索算法（探索状态空间）（综合考虑状态结点到最终状态的代价）

* 贪心算法（直接探索同层级内离目标状态最近的节点）
* A\* 搜索算法（状态结点代价除了计算起点到该状态的总动作代价还要加上该状态到最终状态的代价即距离）