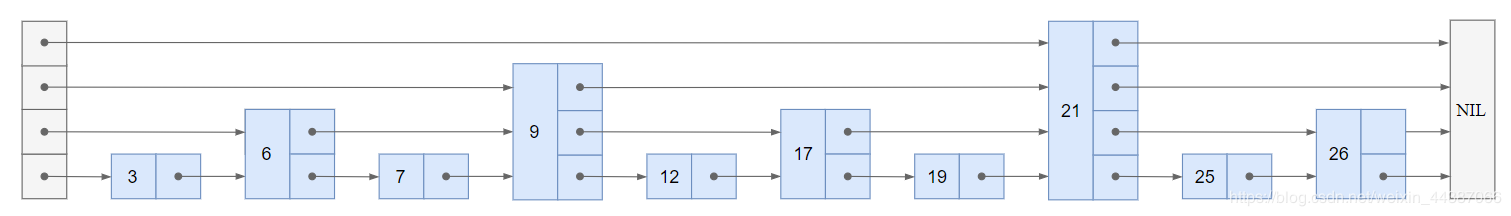
**跳跃表**

跳跃表(skip list) 对标的是平衡树(AVL Tree)，是一种 插入/删除/搜索 都是 O(logn)的数据结构，效率非常高效，z

结构类似于如下:



具体思路就是最下面是一个普通的链表(数据都排好序了)，这个链表包含所有的数据，常规的链表的增删改查的时间复杂度为O(logn)，因为他要找到一个数据只有一个一个从头遍历链表，非常耗时。但是跳跃表的思路就是类似二分查找，为了实现不用从头开始一个一个查找，我们给链表分层，如上所示，如果我要查找数据12，我们先从最上层也就是第四层开始找，发现这层只有21，比我们要找的12大，说明12不在本层，因为本层第一个元素都比他大，然后查找第三层，发现第一个数据是9，然后继续查找，发现下一个是21，说明12在9-12这个区间内，这样就缩小了查找的范围，然后再降低层级，从9开始第二层查找，发现下个节点是17比12打然后确定了12在区间9-17上，然后再次降低层级，从9开始查找，发现下一个刚好就是12，返回

跳跃表的思路就是把链表中的部分节点拿出来单独建链表，可以通过这个链表跳着查找链表中的数据，不用一步一步的去挨个遍历，跳着访问

1.构建跳跃表

参照插入，一个一个插入节点就构建完了跳跃表

2.插入节点

比如我要插入23，然后确定这个节点有几层，通过随机数(抛硬币，初始1如果为正层级数++，直到抛到反面)生成有多少层，假设随机生成了3层，那么我们从第三层开始遍历链表，第一个查找到9，然后查找到21，然后发现后面是null,那么23应该插在21后面，降低层数，节点21第2层开始往后发现下一个是26，那么23就在21和26之间，然后再次降低层级，节点21第1层往后查找，发现后面是25，而且没有下一层了，那么23就插在21和25之间，并且将23插在第二层21和26之间和第三层21到null之间，因为此时的23是三层的节点，必须同时存在第1，2，3层。每次确定在每层的哪个区间的时候就插入

3.删除节点

比如我要删除23，类似上面先查找到23，然后直接类似于普通链表删除掉，每层确定区间的时候就删掉

4.更新节点

先查找到要更新的节点，然后更新数据

5.查找

参照上面的描述

注：关于节点的层级的生成为什么是用随机数而不是每两个就有一个节点的层级树高一级，如果这个跳跃表是静态的这种，如果插入一个数据就很麻烦，首先需要确定这个节点到底是有多少层级，然后插入后可能还需要调整后面的节点的层级，不然的话会破坏结构，所以需要一个动态的跳跃表，就是每个节点的层级都有一定概率的，打个比方，有一百个节点，如果通过抛硬币来决定层级的话，从概率上来讲大概有50个节点是有第二级的，大概有25个节点是有第三级的…，这也符合每层大概有n/(2^k)个节点(k是层级),搜索一个节点的时间复杂度logn