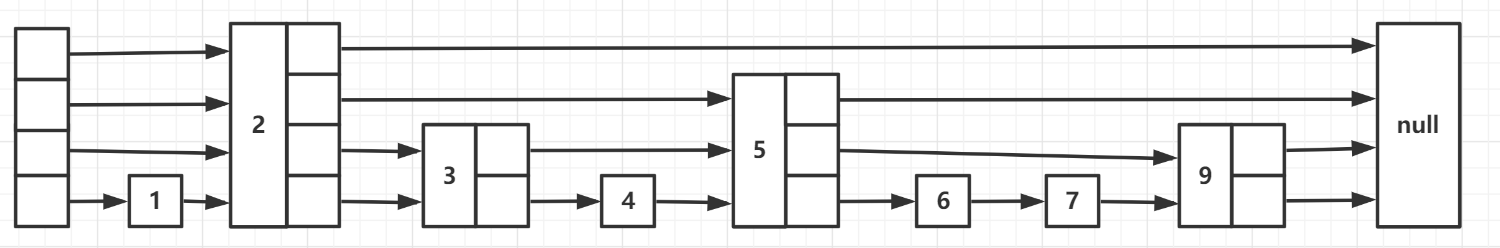
**跳跃列表/跳跃表/跳表(Skip List)：**



有序链表的查询时间复杂度为O(n)。

跳跃表为了加快查询速度，通过建立“索引”，消耗额外的存储空间，换取更快的查询速度，达到了O(log n)。

实现结构：

Skip-List {

int maxLevel;

List<Node> head;

}

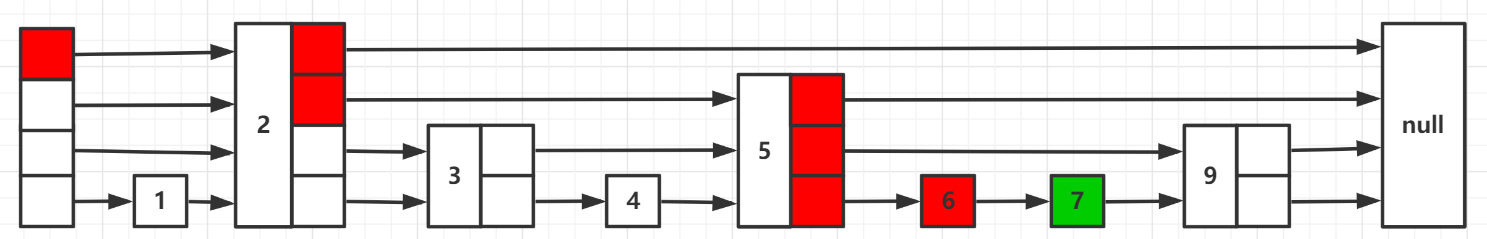
Node {

Value;

List<Node> nodeList;

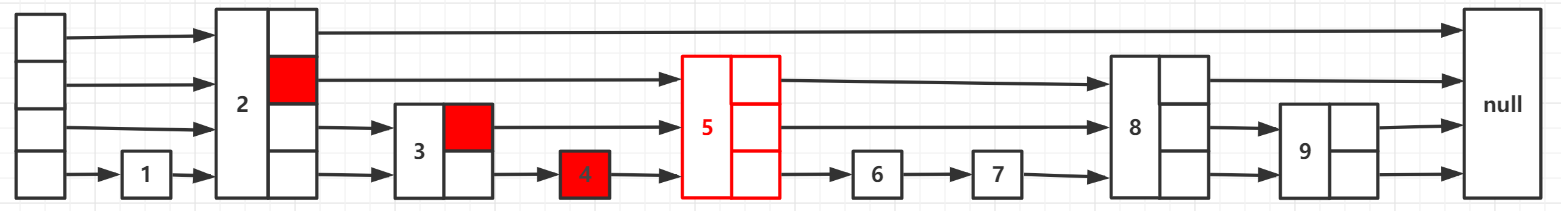
}

**查找：**



如图：查找7，红色节点为其查找路径。

**删除：**

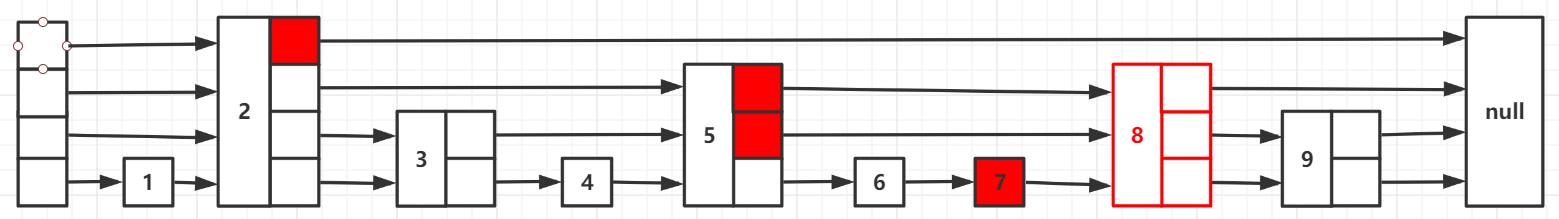


如图：删除5。

找到节点5，并记录每一层可达该节点的node；

删除节点5，并通过记录的node调整指针。

**插入：**



1.找到要插入的位置：节点7后面，并记录对应node。

2.使用随机算法，生成新的节点（随机算法决定新节点的高度）。

3.通过记录的node调整指针。

决定节点高度的随机算法示例（不同版本的实现算法会有不同，此处只做为一个示例说明）：

**int randomLevel( p, maxLevel ){**

**int level = 1;**

**// random() 返回 [0, 1) 的随机数**

**while ( random() < p && level < maxLevel )**

**level ++;**

**return level;**

**}**

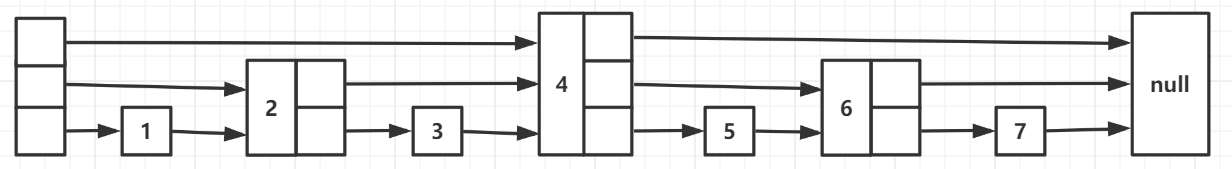
由上例可以看出，跳跃表可以通过设置不同的P值，在查找代价和存储代价做出对应的调整和取舍。

P值太小，level小的节点多，存储花费小，但查找效率会降低。

P值不是越大越好，P值过大时，level大的节点多，花费了空间不说，查找效率也没有提升。

Redis中的跳跃表实现，P=1/4，maxLevel=32。

如下图，为一个高度均衡的跳跃表：



此时的平均查找速度达到最佳。（二分查找）

**结论：**

1. Skip List是一种随机化的数据结构，基于并联的链表，其效率可比拟于二叉查找树（对于大多数操作需要O(log n)平均时间）。基本上，Skip List是对有序的链表增加上附加的前进链接，增加是以随机化的方式进行的，所以在链表中的查找可以快速的跳过部分元素 (因此得名)。
2. Skip List不像平衡树等数据结构那样提供对最坏情况的性能保证：由于用来生成Skip List采用随机选取元素进入更高层的方法，在小概率情况下会生成一个不平衡的Skip List（最坏情况例如最底层仅有一个元素进入了更高层，此时跳跃列表的查找与普通列表一致）。

但是在实际中它通常工作良好，随机化平衡方案也比平衡二叉查找树等数据结构中使用的确定性平衡方案容易实现。

1. Skip List在并行计算中也很有用：插入可以在Skip List不同的部分并行地进行，而不用对数据结构进行全局的重新平衡。
2. Skip List提高效率的方式是通过空间换时间。
3. 通过调整随机算法，可以在查找代价和存储代价之间获取平衡。

与同样高效的红黑树比较：

1. 并发效果好，只涉及左右的指针调整。而红黑树在恢复平衡时涉及到的节点较多。
2. 范围查询方便。定位到节点后，遍历最底层链表即可。
3. 实现简单。
4. 内存占用比红黑树大。