跳跃表(skip list)

  二叉树是我们都非常熟悉的一种数据结构。它支持包括查找、插入、删除等一系列的操作。但它有一个致命的弱点，就是当数据的随机性不够时，会导致其树型结构的不平衡，从而直接影响到算法的效率。  
跳跃表（Skip List）是1987年才诞生的一种崭新的数据结构，它在进行查找、插入、删除等操作时的期望时间复杂度均为O(logn),有着近乎替代平衡树的本领。而且最重要的一点，就是它的编程复杂度较同类的AVL树，红黑树等要低得多，这使得其无论是在理解还是在推广性上，都有着十分明显的优势  
跳跃表由多条链构成（S0，S1，S2 ……，Sh），且满足如下三个条件：  
（1） 每条链必须包含两个特殊元素：+∞ 和 -∞  
（2） S0包含所有的元素，并且所有链中的元素按照升序排列。  
（3） 每条链中的元素集合必须包含于序数较小的链的元素集合，即  
        如果i>j，则Si是Sj的子集。

在对跳跃表有一个初步的认识以后，我们来看一下基于它的几个最基本的操作。

一、 查找  
目的：在跳跃表中查找一个元素x  
 在跳跃表中查找一个元素x，按照如下几个步骤进行：  
i) 从最上层的链（Sh）的开头开始  
ii) 假设当前位置为p，它向右指向的节点为q（p与q不一定相邻），且q的值为y。将y与x作比较  
(1) x=y  输出查询成功及相关信息  
(2) x>y  从p向右移动到q的位置  
(3) x<y  从p向下移动一格

iii) 如果当前位置在最底层的链中（S0），且还要往下移动的话，则输出查询失败

二、插入  
 目的：向跳跃表中插入一个元素x  
 首先明确，向跳跃表中插入一个元素，相当于在表中插入一列从S0中某一位置出发向上的连续一段元素。有两个参数需要确定，即插入列的位置以及它的“高度”。  
 关于插入的位置，我们先利用跳跃表的查找功能，找到比x小的最大的数y。根据跳跃表中所有链均是递增序列的原则，x必然就插在y的后面。  
 而插入列的“高度”较前者来说显得更加重要，也更加难以确定。由于它的不确定性，使得不同的决策可能会导致截然不同的算法效率。为了使插入数据之后，保持该数据结构进行各种操作均为O(logn)复杂度的性质，我们引入随机化算法（Randomized Algorithms）。

我们定义一个随机决策模块，它的大致内容如下：

产生一个0到1的随机数r                            r ← random()  
如果r小于一个常数p，则执行方案A，  if  r<p then do A  
 否则，执行方案B                                     else do B

初始时列高为1。插入元素时，不停地执行随机决策模块。如果要求执行的是A操作，则将列的高度加1，并且继续反复执行随机决策模块。直到第i次，模块要求执行的是B操作，我们结束决策，并向跳跃表中插入一个高度为i的列。  
性质1： 根据上述决策方法，该列的高度大于等于k的概率为p^(k-1)。  
此处有一个地方需要注意，如果得到的i比当前跳跃表的高度h还要大的话，则需要增加新的链，使得跳跃表仍满足先前所提到的条件。

三、删除  
 目的：从跳跃表中删除一个元素x  
 删除操作分为以下三个步骤：  
（1） 在跳跃表中查找到这个元素的位置，如果未找到，则退出  \*  
（2） 将该元素所在整列从表中删除        \*  
（3） 将多余的“空链”删除         \*

【复杂度分析】  
一个数据结构的好坏大部分取决于它自身的空间复杂度以及基于它一系列操作的时间复杂度。跳跃表之所以被誉为几乎能够代替平衡树，其复杂度方面自然不会落后。我们来看一下跳跃表的相关复杂度：

空间复杂度： O(n)    （期望）  
跳跃表高度： O(logn)  （期望）  
相关操作的时间复杂度：  
 查找：  O(logn)  （期望）  
 插入：   O(logn)  （期望）  
 删除：  O(logn)  （期望）  
   
之所以在每一项后面都加一个“期望”，是因为跳跃表的复杂度分析是基于概率论的。有可能会产生最坏情况，不过这种概率极其微小。

一、 空间复杂度分析  O(n)  
假设一共有n个元素。根据性质1，每个元素插入到第i层（Si）的概率为pi-1 ，则在第i层插入的期望元素个数为npi-1，跳跃表的元素期望个数为  ，当p取小于0.5的数时，次数总和小于2n。  
所以总的空间复杂度为O(n)

二、跳跃表高度分析  O(logn)  
根据性质1，每个元素插入到第i层（Si）的概率为p^i ，则在第i层插入的期望元素个数为np^(i-1)。  
考虑一个特殊的层：第1+ 层。  
 层的元素期望个数为 np^0+np^1+...+np^(h-1)，当n取较大数时，这个式子的值接近0，故跳跃表的高度为O(logn)级别的。

三、查找的时间复杂度分析  O(logn)  
我们采用逆向分析的方法。假设我们现在在目标节点，想要走到跳跃表最左上方的开始节点。这条路径的长度，即可理解为查找的时间复杂度。  
设当前在第i层第j列那个节点上。  
i) 如果第j列恰好只有i层（对应插入这个元素时第i次调用随机化模块时所产生的B决策，概率为1-p），则当前这个位置必然是从左方的某个节点向右跳过来的。  
ii) 如果第j列的层数大于i（对应插入这个元素时第i次调用随机化模块时所产生的A决策，概率为p），则当前这个位置必然是从上方跳下来的。（不可能从左方来，否则在以前就已经跳到当前节点上方的节点了，不会跳到当前节点左方的节点）  
设C(k)为向上跳k层的期望步数（包括横向跳跃）  
有：  
C(0) = 0  
C(k) = (1-p)(1+向左跳跃之后的步数)+p(1+向上跳跃之后的步数)  
     = (1-p)(1+C(k)) + p(1+C(k-1))  
C(k) = 1/p + C(k-1)  
C(k) = k/p  
 而跳跃表的高度又是logn级别的，故查找的复杂度也为logn级别。

对于记忆化查找（Search with fingers）技术我们可以采用类似的方法分析，很容易得出它的复杂度是O(logk)的（其中k为此次与前一次两个被查找元素在跳跃表中位置的距离）。

四、插入与删除的时间复杂度分析  O(logn)  
 插入和删除都由查找和更新两部分构成。查找的时间复杂度为O(logn)，更新部分的复杂度又与跳跃表的高度成正比，即也为O(logn)。  
 所以，插入和删除操作的时间复杂度都为O(logn)

最后，概率因子一般用1/2或1/e