让算法的效率“跳起来”！

—— 浅谈“跳跃表”的相关操作及其应用

华东师范大学第二附属中学

魏冉

“跳跃表” — 新生的宠儿

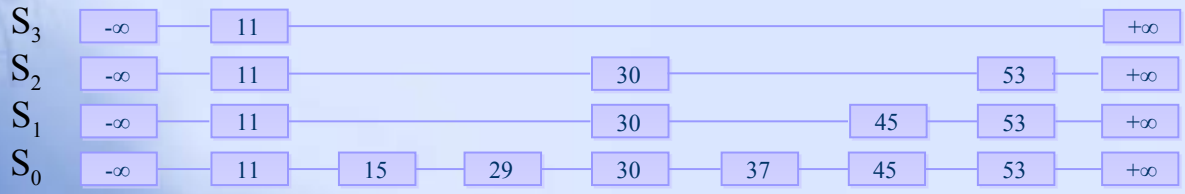
- 跳跃表（Skip List）是1987年才诞生的一种崭新的数据结构，它在进行查找、插入、删除等操作时的时间复杂度均为O(logn)，有着近乎替代平衡树的本领。而且最重要的一点，就是它的编程复杂度较同类的AVL树，红黑树等要低得多，这使得其无论是在理解还是在推广性上，都有着十分明显的优势。

$S_0 \supseteq S_1 \supseteq S_2 \supseteq \ldots \supseteq S_h$

“跳跃表”的结构

- 跳跃表由多条链构成（S_0， S_1， S_2 ……， S_h），且满足如下三个条件：
- 每条链必须包含两个特殊元素：+∞ 和 −∞
- S_0 包含所有的元素，并且所有链中的元素按照升序排列。
- 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：

$$S_0 \supseteq S_1 \supseteq S_2 \supseteq \ldots \supseteq S_h$$



跳跃表的实例

“跳跃表”的时空效率

- 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）

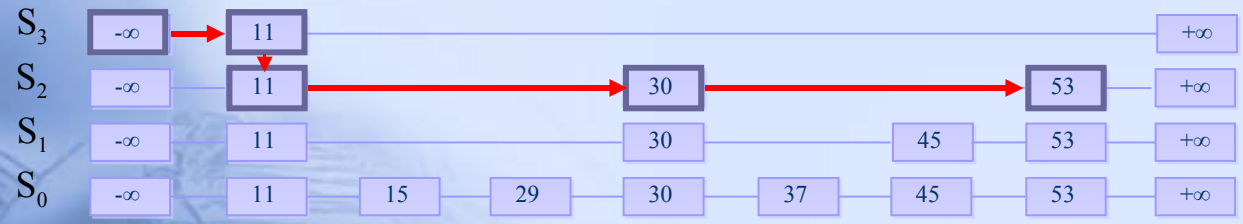
基本操作一　查找

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：

- 从最上层的链（S_h）的开头开始
- 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
  - x=y　　输出*查询成功*，输出相关信息
  - x>y　　从 p 向右移动到 q 的位置
  - x<y　　从 p 向下移动一格
- 如果当前位置在最底层的链中（S_0），且还要往下移动的话，则输出*查询失败*

查找成功！



查询元素 53 的全过程

基本操作二　插入

目的：在跳跃表中插入一个元素 x

- 插入操作由两部分组成：

  查找插入的位置和插入对应元素。

  为了确定插入的“列高”，我们引入一个随机决策模块：

  产生一个 0 到 1 的随机数 r　　r ← random()

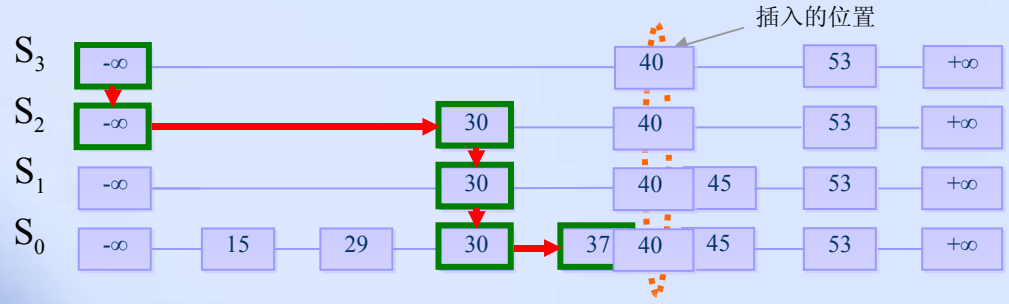
  如果 r 小于一个概率因子 P，则执行方案 A，　if r<p then do A

  否则，执行方案 B　　else do B

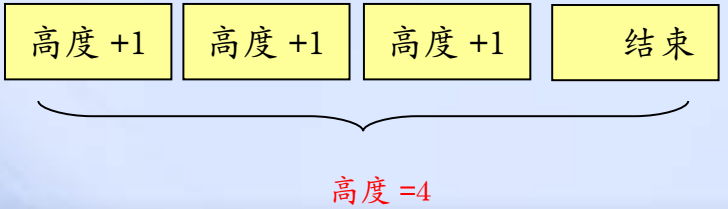
列的初始高度为 1 。插入元素时，不停地执行随机决策模块。如果要求执行的是 A 操作，则将列的高度加 1，并且继续反复执行随机决策模块。直到第 i 次，模块要求执行的是 B 操作，我们结束决策，并向跳跃表中插入一个高度为 i 的列。

基本操作二　插入

- 假设我们现在要插入一个元素 40 到已有的跳跃表中。



随机化模块运行状况：

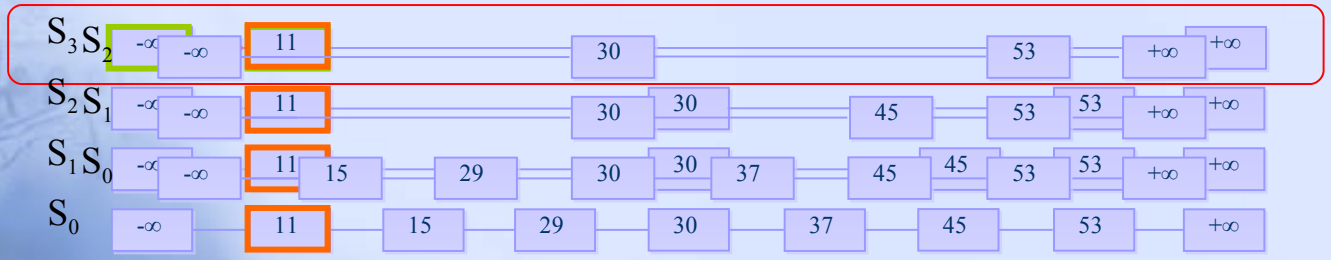


基本操作三　删除

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



概率因子 P 对跳跃表的影响

- 在插入操作中，我们引入了一个概率因子P，它决定了跳跃表的高度，并影响到了整个数据结构的效率。
- 让我们来看看在实际评测过程中，不同的P在时空效率上的差异。

P	平均操作时间	平均列高	总结点数	每次查找跳跃次数（平均值）	每次插入跳跃次数（平均值）	每次删除跳跃次数（平均值）
2/3	0.0024690 ms	3.004	91233	39.878	41.604	41.566
1/2	0.0020180 ms	1.995	60683	27.807	29.947	29.072
1/e	0.0019870 ms	1.584	47570	27.332	28.238	28.452
1/4	0.0021720 ms	1.330	40478	28.726	29.472	29.664
1/8	0.0026880 ms	1.144	34420	35.147	35.821	36.007

跳跃表的应用

- 高效率的相关操作和较低的编程复杂度使得跳跃表在实际应用中的范围十分广泛。尤其在那些编程时间特别紧张的情况下，高性价比的跳跃表很可能会成为你的得力助手。

朋友！

试试跳跃表吧！

您正为找不到合适的数据结构而感到烦恼吗？

您正为自己编写程序的效率高低而感到担忧吗？

它将给您的编程带来超高的效率与无尽的快乐！

您正为陷入 R-B tree 的深渊又无法自拔而感到苦闷吗？

机不可失，时不再来！

详情请致电： 1381xxxxxxx

跳跃表的应用

- NOI2004 Day1 郁闷的出纳员（Cashier）
- 抽象题意：

要求维护这样一个数据结构，使得它支持以下四种操作：

I(x) 插入一个值为 x 元素

A(x) 现有全体元素加上一个值 x

S(x) 现有全体元素减去一个值 x

F(i) 查找现有元素中第 i 大的元素值

（x 为 10^5 级别）

同时还要保持这样一个性质：

现有的元素必须都大于一个特定的值 min，小于 min 的要删去。

我们利用一个虚拟的“零线”来表示 0 在数据结构中的相对位置。这样在进行 A 和 S 操作的时候，只要对“零线”进行调整即可，并不需要对所有元素的值做全面的修改。而在做 S 操作时，为了维持题意中的性质，要注意将值低于（“零线” +min）的元素删除。

支持这些操作的数据结构有很多，比如说线段树，伸展树，跳跃表等。

跳跃表的应用

- NOI2004 Day1 郁闷的出纳员（Cashier）

	I 命令时间复杂度	A 命令时间复杂度	S 命令时间复杂度	F 命令时间复杂度
线段树	O(logR)	O(1)	O(logR)	O(logR)
伸展树	O(logN)	O(1)	O(logN)	O(logN)
跳跃表	O(logN)	O(1)	O(logN)	O(logN)

评测结果：（单位：秒）

	Test1	Test2	Test3	Test4	Test5	Test6	Test7	Test8	Test9	Test10
线段树	0.000	0.000	0.000	0.031	0.062	0.094	0.109	0.203	0.265	0.250
伸展树	0.000	0.000	0.016	0.062	0.047	0.125	0.141	0.360	0.453	0.422
跳跃表	0.000	0.000	0.000	0.047	0.062	0.109	0.156	0.368	0.438	0.375

跳跃表的应用

- HNOI2004　　Day1　　宠物收养所（Pet）
- 抽象题意：

  维护一个数据结构，使得它支持以下两种操作：
  - 插入一个元素　x（$0<x<2^{31}$）
  - 给定一个元素 y，删除现有与 y 差值最小的元素 x　　（|x-y| 为最小）

  所有操作次数不超过200000

思考：~~线段树~~……？

如果采取边做边开空间的策略，勉强可以缓解内存的压力，但此题对内存的要求很苛刻，元素相对范围来说也比较少，如果插入的元素稍微分散一些，就很有可能使得空间复杂度接近O（nlogR）！

好！

- 摆脱实数的约束
- 树状的编程复杂度能够使你很轻松的解决此题！

实数呢？

总结

- 跳跃表作为一种新兴的数据结构，以相当高的效率和较低的复杂度散发着其独特的光芒。和同样以编程复杂度低而闻名的“伸展树”相比，跳跃表的效率不但不会比它差，甚至优于前者。

“跳跃表”这个名字有着其深远的意义！

谢谢大家

Thank you