让算法的效率“跳起来”！

—— 浅谈“跳跃表”的相关操作及其应用

上海市华东师范大学第二附属中学　魏冉

【目录】

【关键字】

跳跃表　高效　概率　随机化

【摘要】

本文分为三大部分。

首先是概述部分。它会从功能、效率等方面对跳跃表作一个初步的介绍，并给出其图形结构，以便读者对跳跃表有个形象的认识。

第二部分将介绍跳跃表的三种基本操作——查找，插入和删除，并对它们的时空复杂度进行分析。

第三部分是对跳跃表应用的介绍，并通过实际测试效果来对跳跃表以及其它一些相关数据结构进行对比，体现其各自的优缺点。

最后一部分是对跳跃表数据结构的总结。

【概述及结构】

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。

首先，我们来看一下跳跃表的结构（如图 1）

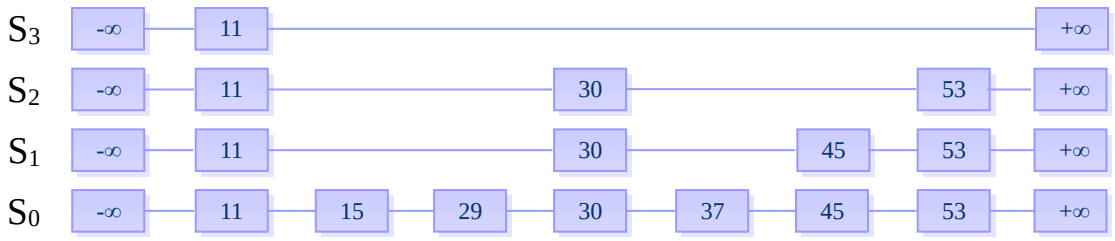

图 1　有 7 个元素的跳跃表

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：

（1）　每条链必须包含两个特殊元素：+∞ 和 −∞

（2）　S_0 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：

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

【基本操作】

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（S_h）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（S_0），且还要往下移动的话，则输出***查询失败***

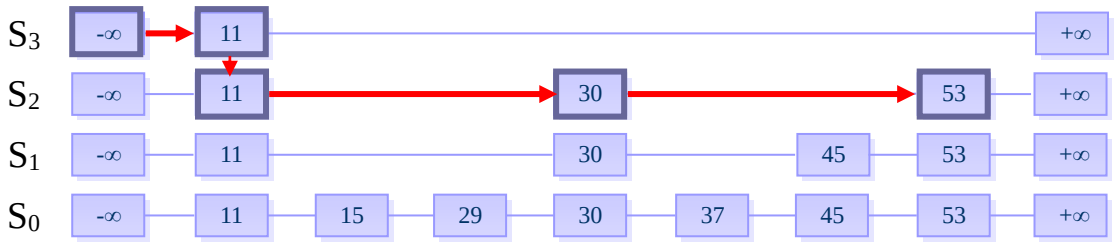

图 2　查询元素 53 的全过程

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 S_0 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。

我们来看一个例子：

假设当前我们要插入元素“40”，且在执行了随机决策模块后得到高度为 4

- 步骤一：找到表中比 40 小的最大的数，确定插入位置

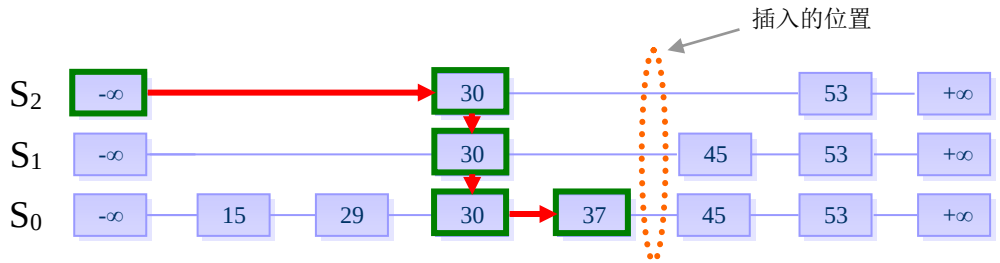

图 3.1　确定插入的位置

- 步骤二：插入高度为 4 的列，并维护跳跃表的结构

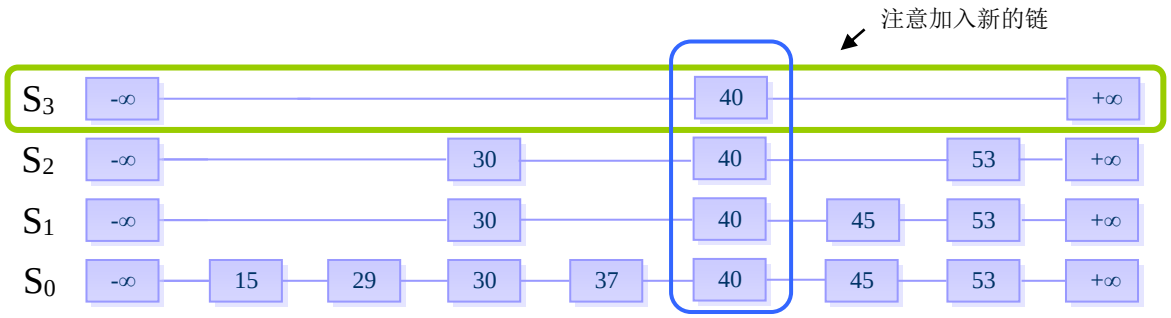

图 3.2　插入高度为 4 的列，并维护跳跃表

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　　*

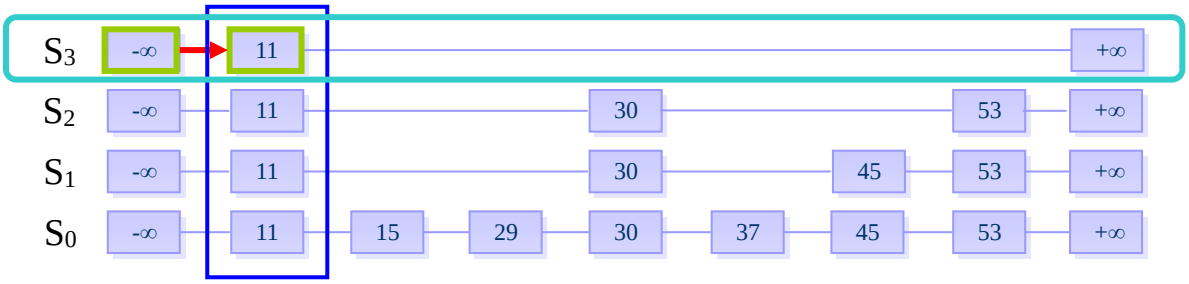

图 4.1　删除元素 11 的全过程

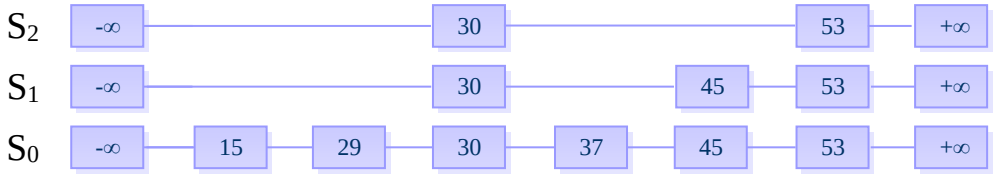

图 4.2　删除以后的结构

四、“记忆化”查找（Search with fingers）

所谓“记忆化”查找，就是在前一次查找的基础上进行进一步的查找。它可以利用前一次查找所得到的信息，取其中可以被当前查找所利用的部分。利用“记忆化”查找可以将一次查找的复杂度变为 O(logk)，**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。**

下面来看一下记忆化搜索的具体实现方法：

假设上一次操作我们查询的元素为 i，此次操作我们欲查询的元素为 j。我们用一个 update 数组来记录在查找 i 时，指针在每一层所“跳”到的最右边的位置。如图 4.1 中橘黄色的元素。（蓝色为路径上的其它元素）

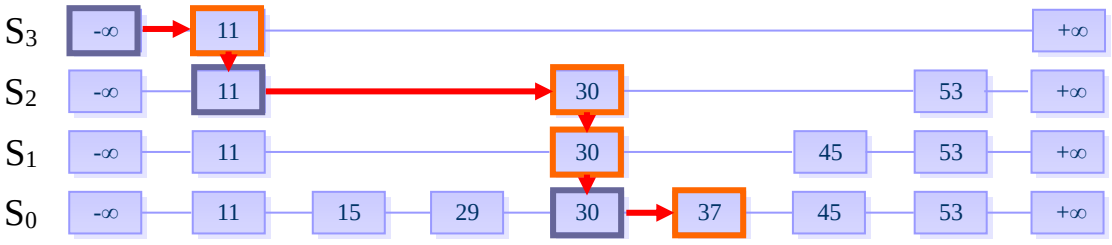

图 4.1　查找元素 37

在插入元素 j 时，分为两种情况：

（1）　i<=j

从 S_0 层开始向上遍历 update 数组中的元素，直到找到某个元素，它向右指向的元素大于等于 j，并于此处开始新一轮对 j 的查找（与一般的查找过程相同）

（2）　i>j

从 S_0 层开始向上遍历 update 数组中的元素，直到找到某个元素小于等于 j，并于此处开始新一轮对 j 的查找（与一般的查找过程相同）

图 4.2 十分详细地说明了在查找了 i=37 之后，继续查找 j=15 或 53 时的两种不同情况。

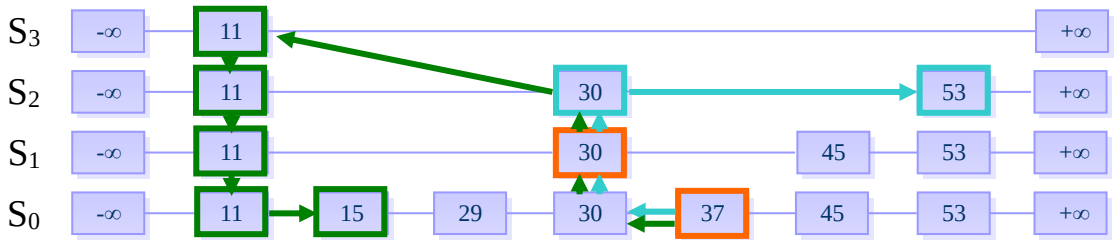

图 4.2　新一轮查找元素为 **15**（**53**）的步骤流程

记忆化查找（Search with fingers）技术对于那些前后相关性较强的数据效率极高，这点可以在后文中的实际测试报告中略见一斑。

【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。

下面我们来一项一项分析。

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（S_i）的概率为 p^{i-1}，则在第 i 层插入的期望元素个数为 np^{i-1}，跳跃表的元素期望个数为 $\sum_{i=0}^{h-1} np^i$，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（S_i）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 $1+3\log_{1/p} n$ 层。

$S_{1+3\log_{1/p} n}$ 层的元素期望个数为 $np^{3\log_{1/p} n} = 1/n^2$，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)

五、实际测试效果

（1）不同的 p 对算法复杂度的影响

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

表 1　进行 10^6 次随机操作后的统计结果

从表 1 中可见，当 p 取 1/2 和 1/e 的时候，时间效率比较高（为什么？）。而如果在实际应用中空间要求很严格的话，那就可以考虑取稍小一些的 p，如 1/4。

（2）运用“记忆化”查找（Search with fingers）的效果分析

所谓“记忆化”查找，就是在前一次查找的基础上进行进一步的查找。它可以利用前一次查找所得到的信息，取其中可以被当前查找所利用的部分。利用“记忆化”查找可以将一次查找的复杂度变为 O(logk)，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。

P	数据类型	平均操作时间（不运用记忆化查找）	平均操作时间（运用记忆化查找）	平均每次查找跳跃次数（不运用记忆化查找）	平均每次查找跳跃次数（运用记忆化查找）
0.5	随机（相邻被查找元素键值差的绝对值较大）	0.0020150 ms	0.0020790 ms	23.262	26.509
0.5	前后具备相关性（相邻被查找元素键值差的绝对值较小）	0.0008440 ms	**0.0006880 ms**	26.157	**4.932**

表 1　进行 10^6 次相关操作后的统计结果

从表 2 中可见，当数据相邻被查找元素键值差绝对值较小的时候，我们运用“记忆化”查找的优势是很明显的，不过当数据随机化程度比较高的时候，“记忆化”查找不但不能提高效率，反而会因为跳跃次数过多而成为算法的瓶颈。

合理地利用此项优化，可以在特定的情况下将算法效率提升一个层次。

【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。

能运用到跳跃表的地方很多，与其去翻陈年老题，不如来个趁热打铁，拿 NOI2004 第一试的第一题——郁闷的出纳员(Cashier)来“小试牛刀”吧。

例题一：NOI2004　Day1 郁闷的出纳员(Cashier)

[点击查看附录中的原题]

这道题解法的多样性给了我们一次对比的机会。用不同的算法和数据结构，在效率上会有怎样的差异呢？

首先定义几个变量

R － 工资的范围

N － 员工总数

我们来看一下每一种适用的算法和数据结构的简要描述和理论复杂度：

（1） 线段树

简要描述：以工资为关键字构造线段树，并完成相关操作。

I 命令时间复杂度：O(logR)

A 命令时间复杂度：O(1)

S 命令时间复杂度：O(logR)

F 命令时间复杂度：O(logR)

（2） 伸展树(Splay tree)

简要描述：以工资为关键字构造伸展树，并通过“旋转”完成相关操作。

I 命令时间复杂度：O(logN)

A 命令时间复杂度：O(1)

S 命令时间复杂度：O(logN)

F 命令时间复杂度：O(logN)

（3） 跳跃表(Skip List)

简要描述：运用跳跃表数据结构完成相关操作。

I 命令时间复杂度：O(logN)

A 命令时间复杂度：O(1)

S 命令时间复杂度：O(logN)

F 命令时间复杂度：O(logN)

实际效果评测：　（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

从结果来看，线段树这种经典的数据结构似乎占据着很大的优势。可有一点万万不能忽略，那就是线段树是基于键值构造的，它受到键值范围的约束。在本题中 R 的范围只有 10^5 级别，这在内存较宽裕的情况下还是可以接受的。但是如果问题要求的键值范围较大，或者根本就不是整数时，线段树可就很难适应了。这时候我们就不得不考虑伸展树、跳跃表这类基于元素构造的数据结构。而从实际测试结果看，跳跃表的效率并不比伸展树差。加上编程复杂度上的优势，跳跃表尽显出其简单高效的特点。

参考程序：


cashier_skiplist.txt

例题二：HNOI2004　Day1　宠物收养所(pet)

[点击查看附录中的原题]

此题与《郁闷的出纳员》最大的不同，就在于它的键值范围达到了 2^{31} 级别。这对线段树来说可是一大考验。虽然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(NlogN)!何况如果稍微拓展一下，**插入的元素不是整数而是实数呢？**

而这道题对于跳跃表来说,可真是再适合不过了。几乎对标准的算法不需要做修改，如果熟练的话，从思考到编写完成也就 20 分钟左右的时间，最终的算法效率也很高。更加重要的一点，跳跃表绝不会因键值类型的变化而失效，推广性很强。

参考程序：


pet.txt

【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（见附表 1）。

人们在思考一类问题的时候，往往会无意中被局限在一个小范围当中。就拿和平衡树相关的问题来说，人们凭借自己的智慧，创造出了红黑树，AVL 树等一些很复杂的数据结构。可是千变万变，却一直走不出“树”这个范围。过高的编程复杂度使得这些成果很难被人们所接受。而跳跃表的出现，使得人们眼前顿时豁然开朗。原来用与树完全不相关的数据结构也能够实现树的功能！

“跳跃表”这个名字有着其深远的意义。不仅是因为它形象地描述了自身的结构，更有一点，它象征着一种思考方法，一种“跳出定式”的思考方法。在你面临一个困难却山穷水复疑无路的时候，不妨找到问题的原点，“跳”出思维的定式，说不定在另一条全新的路上，你将会看到胜利的曙光。

【参考文献】

[1] William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees
[2] William Pugh. A Skip List Cookbook

【附录】

• 附表：跳跃表与 AVL 树、2-3 树、伸展树在时间效率上的对比（摘自 *[1] William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees*）

Implementation	Search Time	Insertion Time	Deletion Time
Skip lists	0.051 msec (1.0)	0.065 msec (1.0)	0.059 msec (1.0)
non-recursive AVL trees	0.046 msec (0.91)	0.10 msec (1.55)	0.085 msec (1.46)
recursive 2-3 trees	0.054 msec (1.05)	0.21 msec (3.2)	0.21 msec (3.65)
Self-adjusting trees:			
top-down splaying	0.15 msec (3.0)	0.16 msec (2.5)	0.18 msec (3.1)
bottom-up splaying	0.49 msec (9.6)	0.51 msec (7.8)	0.53 msec (9.0)

Table 2 - Timings of implementations of different algorithms

• 程序：“跳跃表”的程序(Pascal 语言实现)：

skiplist.txt

附解：为什么 p=1/e 的时候时间效率最高？

解：由复杂度分析中得出，跳跃表的时间效率取决于跳跃的次数，也就是 k/p（k 为跳跃表高度），而 k 是 $\log_{1/p} n$ 级别。故有：

$$f(x) = k/p = \frac{\log_{1/p} n}{p} = \ln n \frac{1/p}{\ln 1/p}$$

令 x = 1/p ，则有：

$$f(x) = \ln n \frac{x}{\ln x}$$

对 $g(x) = \frac{x}{\ln x}$ 求导，有 $g'(x) = \frac{\ln x - 1}{(\ln x)^2}$

当 x=e 时，g’(x)=0，即 f(x)到达极值点。
此时 p = 1/e

[返回“实际测试效果部分”]

附题：NOI2004 Day1　郁闷的出纳员

[返回“跳跃表的应用”部分]

郁闷的出纳员

【问题描述】

OIER 公司是一家大型专业化软件公司，有着数以万计的员工。作为一名出纳员，我的任务之一便是统计每位员工的工资。这本来是一份不错的工作，但是令人郁闷的是，我们的老板反复无常，经常调整员工的工资。如果他心情好，就可能把每位员工的工资加上一个相同的量。反之，如果心情不好，就可能把他们的工资扣除一个相同的量。我真不知道除了调工资他还做什么其它事情。

工资的频繁调整很让员工反感，尤其是集体扣除工资的时候，一旦某位员工发现自己的工资已经低于了合同规定的工资下界，他就会立刻气愤地离开公司，并且再也不会回来了。每位员工的工资下界都是统一规定的。每当一个人离开公司，我就要从电脑中把他的工资档案删去，同样，每当公司招聘了一位新员工，我就得为他新建一个工资档案。

老板经常到我这边来询问工资情况，他并不问具体某位员工的工资情况，而是问现在工资第 k 多的员工拿多少工资。每当这时，我就不得不对数万个员工进行一次漫长的排序，然后告诉他答案。

好了，现在你已经对我的工作了解不少了。正如你猜的那样，我想请你编一个工资统计程序。怎么样，不是很困难吧？

【输入文件】

第一行有两个非负整数 n 和 min。n 表示下面有多少条命令，min 表示工资下界。

接下来的 n 行，每行表示一条命令。命令可以是以下四种之一：

名称	格式	作用
I 命令	I_k	新建一个工资档案，初始工资为 k。如果某员工的初始工资低于工资下界，他将立刻离开公司。
A 命令	A_k	把每位员工的工资加上 k
S 命令	S_k	把每位员工的工资扣除 k
F 命令	F_k	查询第 k 多的工资

_（下划线）表示一个空格，I 命令、A 命令、S 命令中的 k 是一个非负整数，F 命令中的 k 是一个正整数。

在初始时，可以认为公司里一个员工也没有。

【输出文件】

输出文件的行数为 F 命令的条数加一。

对于每条 F 命令，你的程序要输出一行，仅包含一个整数，为当前工资第 k 多的员工所拿的工资数，如果 k 大于目前员工的数目，则输出-1。

输出文件的最后一行包含一个整数，为离开公司的员工的总数。

【样例输入】

```
9 10
I 60
```

I 70
S 50
F 2
I 30
S 15
A 5
F 1
F 2

【样例输出】
10
20
-1
2

【约定】
I 命令的条数不超过 100000
A 命令和 S 命令的总条数不超过 100
F 命令的条数不超过 100000
每次工资调整的调整量不超过 1000
新员工的工资不超过 100000

【评分方法】
对于每个测试点，如果你输出文件的行数不正确，或者输出文件中含有非法字符，得分为 0。否则你的得分按如下方法计算：如果对于所有的 F 命令，你都输出了正确的答案，并且最后输出的离开公司的人数也是正确的，你将得到 10 分；如果你只对所有的 F 命令输出了正确答案，得 6 分；如果只有离开公司的人数是正确的，得 4 分；否则得 0 分。

附题：HNOI2004 Day1　宠物收养所
[返回“跳跃表的应用”部分]

宠物收养所

(pet)

【背景描述】

最近，阿 Q 开了一间宠物收养所。收养所提供两种服务：收养被主人遗弃的宠物和让新的主人领养这些宠物。

每个领养者都希望领养到自己满意的宠物，阿 Q 根据领养者的要求通过他自己发明的一个特殊的公式，得出该领养者希望领养的宠物的特点值 a（a 是一个正整数，a<2^31），而他也给每个处在收养所的宠物一个特点值。这样他就能够很方便的处理整个领养宠物的过程了，宠物收养所总是会有两种情况发生：被遗弃的宠物过多或者是想要收养宠物的人太多，而宠物太少。

1. 被遗弃的宠物过多时，假若到来一个领养者，这个领养者希望领养的宠物的特点值为 a，那么它将会领养一只目前未被领养的宠物中特点值最接近 a 的一只宠物。（任何两只宠物的特点值都不可能是相同的，任何两个领养者的希望领养宠物的特点值也不可能是一样的）如果有两只满足要求的宠物，即存在两只宠物他们的特点值分别为 a-b 和 a+b，那么领养者将会领养特点值为 a-b 的那只宠物。
2. 收养宠物的人过多，假若到来一只被收养的宠物，那么哪个领养者能够领养它呢？能够领养它的领养者，是那个希望被领养宠物的特点值最接近该宠物特点值的领养者，如果该宠物的特点值为 a，存在两个领养者他们希望领养宠物的特点值分别为 a-b 和 a+b，那么特点值为 a-b 的那个领养者将成功领养该宠物。

一个领养者领养了一个特点值为 a 的宠物，而它本身希望领养的宠物的特点值为 b，那么这个领养者的不满意程度为 abs(a-b)。

【任务描述】

你得到了一年当中，领养者和被收养宠物到来收养所的情况，希望你计算所有收养了宠物的领养者的不满意程度的总和。这一年初始时，收养所里面既没有宠物，也没有领养者。

【输入格式】：(input.txt)

你将从文件 input.txt 当中读入数据。文件的第一行为一个正整数 n，n<=80000，表示一年当中来到收养所的宠物和领养者的总数。接下来的 n 行，按到来时间的先后顺序描述了一年当中来到收养所的宠物和领养者的情况。每行有两个正整数 a, b，其中 a=0 表示宠物，a=1 表示领养者，b 表示宠物的特点值或是领养者希望领养宠物的特点值。（同一时间呆在收养所中的，要么全是宠物，要么全是领养者，这些宠物和领养者的个数不会超过 10000 个）

【输入样例】

```
5
0 2
0 4
1 3
1 2
1 5
```

【输出格式】：(cut.out)

输出文件 output.txt 中仅有一个正整数，表示一年当中所有收养了宠物的领养者的不

满意程度的总和 mod 1000000 以后的结果。

【输出样例】

```
3
```

(abs(3-2) + abs(2-4)=3，最后一个领养者没有宠物可以领养)

【运行限制】

运行时限：1 秒钟

【评分方法】

本题目一共有十个测试点，每个测试点的分数为总分数的 10%。对于每个测试点来说，如果你给出的答案正确，那么你将得到该测试点全部的分数，否则得 0 分。