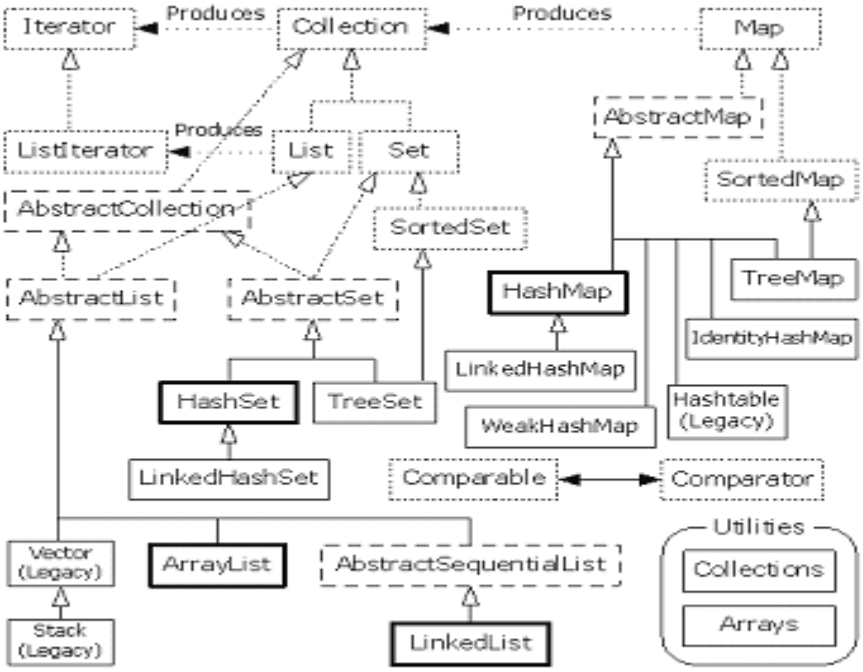
浅谈 Java 容器

杜鹏飞,2013/4



Full Container Taxonomy

The dotted boxes represents interfaces,and the solid boxes are regular(concrete)classes.The dotted lines with hollow arrows indicate that a particular class is implementating an interface.The solid arrows show that a class can produce objects of the class the arrow is pointing to.

点框代表接口,实框代表一般的(具体的)类.带有空心箭头的点线指明了一个类所模拟的接口(抽象类模拟部分接口).实箭头表示一个类可以产生其所指向的类的对象.

~Thinking in Java(Fourth edition)

上图虽然不是最新版的  ,但仍清晰地描述了 Java 中容器之间的关系.

关于容器的常规用法,本文不再赘述,请参阅 JDKdoc.

我在这里讨论如何选择容器.要解决这个问题,不仅要了解 Java 中容器之间的关系,更要对容器的实现做了解.

注意上图中的 Vector,Stack,HashTable 标记为 Legacy,说明这三者是过时的.(可是我对 Vector,Stack,HashTable 倒有不少亲切感).

Collection 只是提供了一组通用接口,故不作讨论.

虽然 Link,set,map 也是一组接口,但它们提供的信息比 Collection 多得多.

Link

Link,最常见的是 Linked List,也有 Double Linked List,均是通过结点内部的子域将一组结点串联起来.链表有许多变种,诸如块状链表,十字链表,跳跃表等等,值得一提的是,理论计算机

科学家 Donald Knuth 曾于千禧年发表了 *Dancing Links,*有兴趣的读者可以一看.

说了点题外话,现在接着讨论.可以看到,ArrayList 和 LinkedList 均是模拟了 AbstractList,LinkedList 顾名思意,并且从它支持迭代器可以发现其使用双向链表实现的,那 ArrayList 是什么呢?这倒让我疑惑了.

先看看 JDK 中对 ArrayList 的描述:

The size, isEmpty, get, set, iterator, and listIterator operations run in constant time. The add operation runs in amortized constant time, that is, adding n elements requires O(n) time. All of the other operations run in linear time (roughly speaking). The constant factor is low compared to that for the LinkedList implementation.

Size,isEmpty,get,set,iterator 和 ListIterator 操作的时间复杂度均为 O(1),而 add 操作的均摊时间复杂度为 O(1),这个神了!

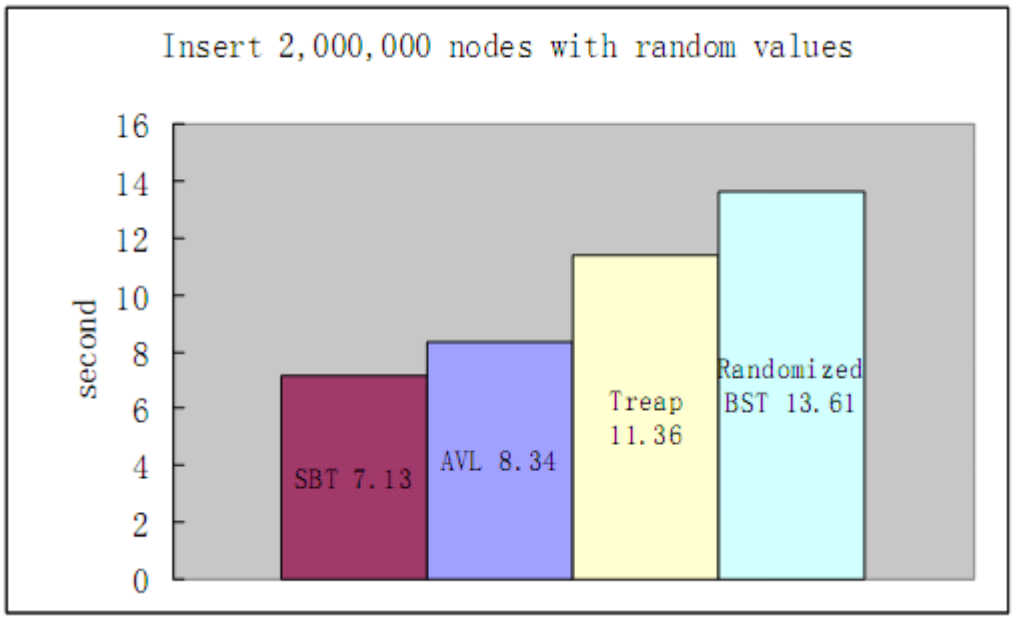
Each ArrayList instance has a capacity. The capacity is the size of the array used to store the elements in the list. It is always at least as large as the list size. As elements are added to an ArrayList, its capacity grows automatically. The details of the growth policy are not specified beyond the fact that adding an element has constant amortized time cost.

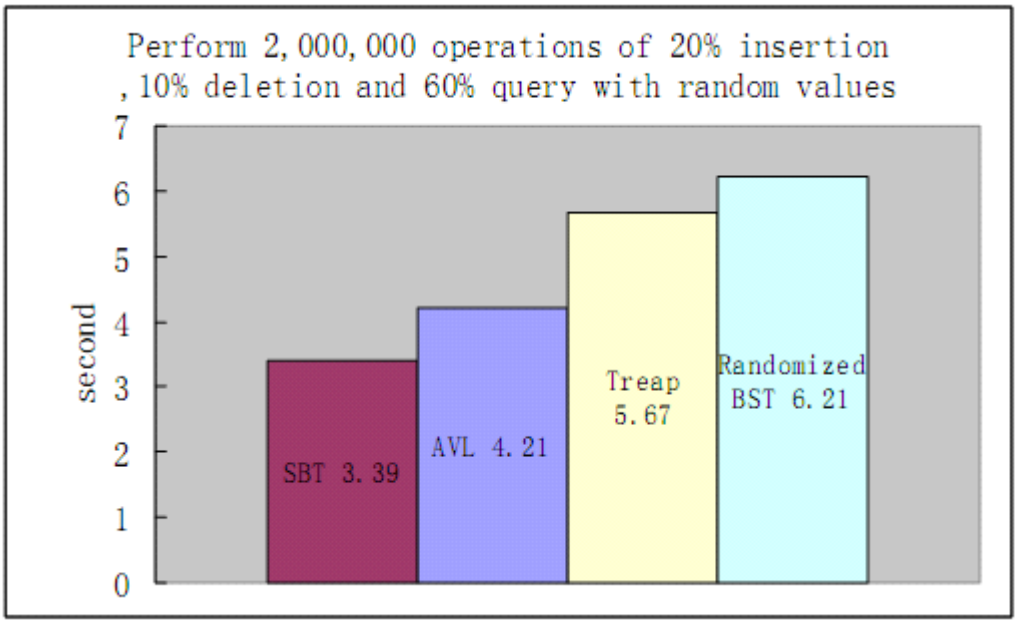
啊哈,这个就是动态表,却取了个 ArrayList 的名字来唬人.由此发现,Java 放弃了 Vector 也不是没有道理的.

Set

接下来讨论 Set,注意 TreeSet 继承了 SortedSet,这说明 TreeSet 中的元素是可比较的.根据官方资料,TreeSet 是用红黑树来实现的.红黑树是由 Bayer 在 1971 年发明的,详细内容见 *Symmetric Binary B-Trees:Data Structure and Algorithms for Random and Sequentical processing,* 一篇研究红黑树性质的论文 *A dichromatic framework for balanced trees,*作者是 Guibas 和 Sedgewick.

在这里我不打算讨论红黑树,这种数据结构相当复杂.下面用几张测试图来说明,感谢现在斯坦福大学的陈启峰!


Insert 2,000,000 nodes with random values
second
16
14
12
10
8
6
4
2
0
SBT 7.13
AVL 8.34
Treap
11.36
Randomized
BST 13.61


Perform 2,000,000 operations of 20% insertion
,10% deletion and 60% query with random values
second
7
6
5
4
3
2
1
0
SBT 3.39
AVL 4.21
Treap
5.67
Randomized
BST 6.21

Insert 2,000,000 nodes with random values

Type	SBT	AVL	Treap	Randomized BST	Splay	Perfect BST
Average Depth	19.2415	19.3285	26.5062	25.5303	37.1953	18.9514
Height	24	24	50	53	78	20
Times of Rotation	1568017	1395900	3993887	3997477	25151532	?

Insert 2,000,000 nodes with ordered values

Type	SBT	AVL	Treap	Randomized BST	Splay	Perfect BST
Average Depth	18.9514	18.9514	25.6528	26.2860	999999.5	18.9514
Height	20	20	51	53	1999999	20
Times of Rotation	1999979	1999979	1999985	1999991	0	?

SBT 并不是 Red-black tree,两者孰优孰劣呢?你猜呢?

看到这,你也许会说,HashXX 呢,不讨论了吗?X,当然要讨论,继续看吧!

Map

对 Map 的讨论涉及到一个”恐怖的”名词 Retrieval on the secondary keys,恐怖的不仅仅是这个不好翻译的名词,还有这个名词背后所涉及的技术.

想象如何去维护一个公司中雇员的有关信息.假设需要维护的是工资,生日,年龄,性别以及是否单身等等.

你会想,这四种信息肯定是数值!当我们要维护一条信息,可以用单维数轴,要维护两条信息,用二维数轴,以此类推.

对,基本思路就是这样!仅仅由于实现的不同,导致了 HashXX 和 TreeXX 两种技术的产生!

TreeXX:

可以将问题再次简化.

假设我们要查询年龄在 a(ge)_l~age_r, 工资在 p(ay)_l~pay_r, 生日月份在 m(onth)_l~month_r, 我们就是要在 3 维坐标轴上找到与正方体 [a_l,a_r]x[p_l,p_r]x[m_l,m_r]相交的点,而点代表着某个雇员.

这个问题就是计算几何中的经典问题~正交区域查找.

实际中,p(ay)等可能是很大的,所以我们不能用一条线段来表示解答空间.相反,我们

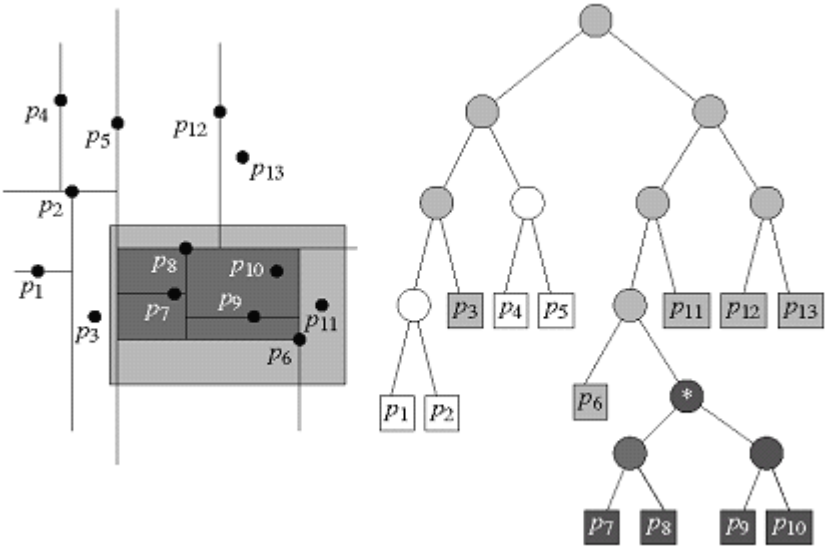
需要一种数据结构,来维护雇员的对应信息.

目前流行的数据结构是 k-d 树和高维区域树.

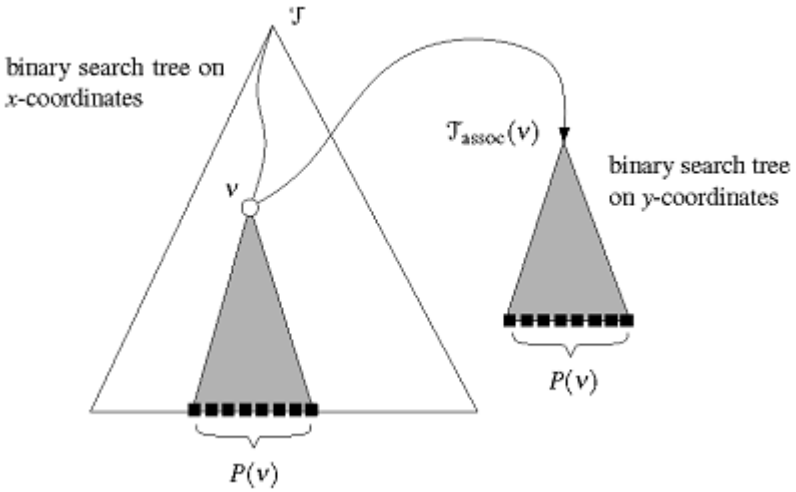
我从简单情况入手,来概要介绍下两种数据结构.假设我仅仅是要确定是否存在这样一个点(a,b).

K-d 树:考虑在二叉查找树上的查找过程,是通过比较左右子树来进行的.如何将其扩展呢?下面我们需要关注”点集”,首先将点集 A 根据 x 坐标二分,A 分裂出 A_1,A_2 两个子集(1),接下来依次对每个子集按 y 坐标二分(2),两者交替进行直到集合为空.

查找时,交替查找 a,b.



高维区域树:简单来说,就是将 K-d 树的第(2)个步骤改为将其作为一个以 y 坐标划分的辅助二分查找树以形成树套树.



不管怎样,我们都要维护高维树,也就是树套树.

Oh my god!

HashXX:

继续上面的例子.注意,是由于有时解答空间过大,我们才采用树结构.可是,就不能把解答空间缩小吗?

思路就是,我们要找到一个散列函数 H(n1,n2,...,nk),来将雇员散列到 k 维区域中的一点.

So easy?

如果无法采取相当优秀的散列函数,那么就需要利用链接法解决冲突,这意味着你不能维护 k 维数组,要维护 k 维数链!不仅如此,对参数的处理,散列函数的选择,插入,删除的设计,查询的方法都要仔细考虑.真的很复杂,不亚于 TreeXX.

故不再讨论了,向那些写作容器的人们(Java 的标准容器由 Joshua Bloch 设计)致敬!

到这里,本文就结束了.可是,本文所探讨的技术仍在蓬勃发展.