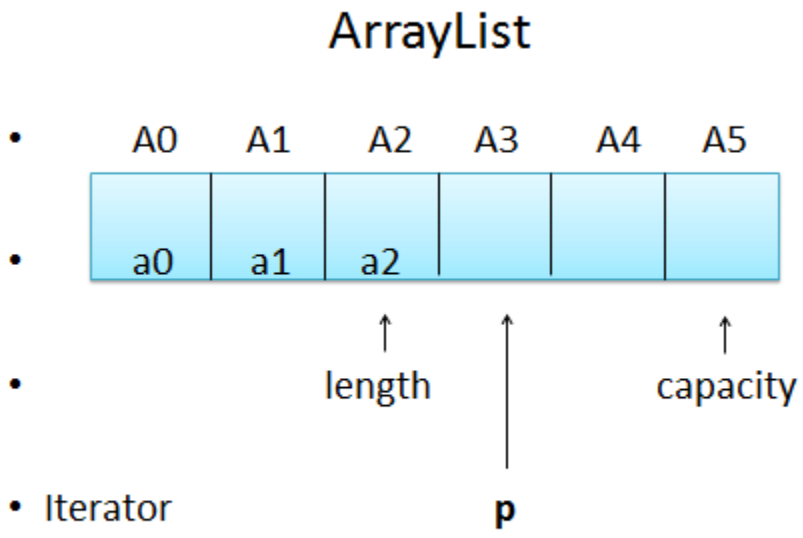
Bees Report

实现方式：

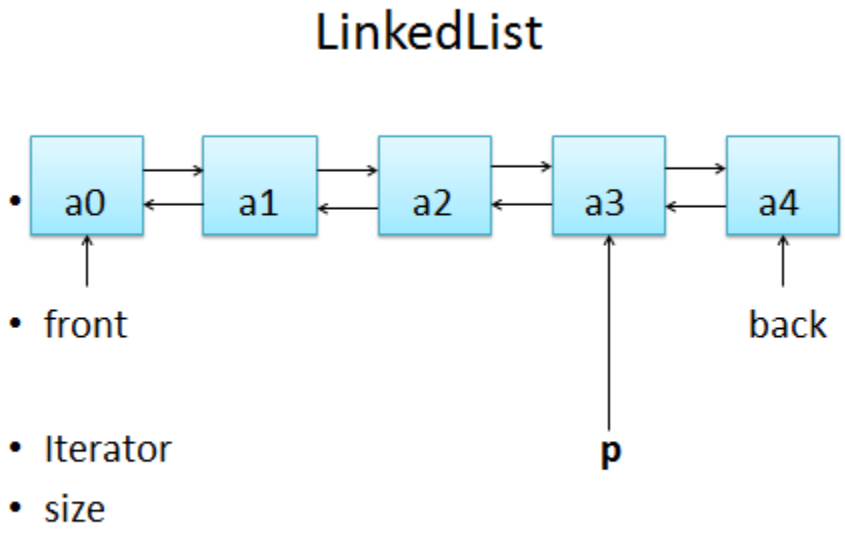
ArrayList：

这个类的实现应当说是最简单的，利用一个动态数组，再记录 capacity 和 length 即可很容易的实现. 迭代器的实现也是不难的. 在迭代器里只需保存一个指向ArrayList的指针还有一个int类型的变量表示迭代器下一次返回的元素在数组中的位置，每次遍历只需把 int 类型的变量一次++即可. add 操作需将length++，在数组最后一个位置存入该元素即可，查询的时候只需从头到尾遍历一次数组，判断有没有该元素，删除操作只需 length--，并把该位置后的所有元素一次前移一位，get 可直接返回数组相应下表对应的值，对于 subList 操作，只需生成一个新的ArrayList类型的变量，并依次add对应范围内的元素即可. 复制构造，operator= 的方法也类似. 值得注意的是，当 length == capacity 的时候，如果 length 还要增加，就要调用 doubleSpace()函数增加数组大小.



LinkedList：

实现 LinkedList 需要在 private 里定义一个节点类型 Node，存入指向前后节点的指针和 data，实现的时候应该也说是比较简单的，记录头尾指针 front 和 back，并且为了快速得到 List 的大小，记录一个 int 型的 size，在使用指针的时候应当尽量注意避免使用空指针导致内存错误即可．迭代器的实现也是不难的，在迭代器里只需保存一个指向 LinkedList 的指针和指向迭代器下一次 next()返回值所在的Node的指针，每次遍历只需把Node类型的指针依次->next 即可．add 操作只需 new 一个 Node，并加入到双链表的尾部，并令 size++．查询的时候只需从头到尾遍历一次数组，判断有没有该元素．删除操作只需从链表中删除该节点，回收空间，并 size--．对于 subList，复制构造等函数，方法与上面基本相同.

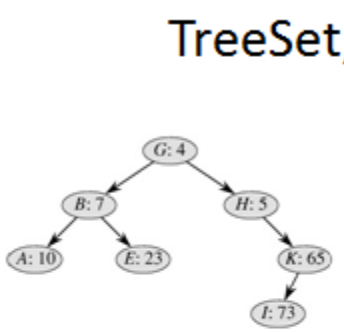


TreeSet：

实现 TreeSet 的时候我采用链接方式的 Treap 实现的. 同样需要在 private 里定义一个节点类型 Node, 存入指向左右子节点, 父节点的指针, data, 和一个额外的数据 priority(优先级). 对于迭代器的实现, 同样, 在迭代器里需保存的变量有指向 TreeSet 的指针和指向下一次 next()返回值所在的 Node 的指针, 查找元素很简单, 只需递归地寻找左子树或右子树. 对插入操作, 给节点随机分配一个优先级，先和二叉搜索树的插入一样，先把要插入的点插入到一个叶子上，然后跟维护堆一样，如果当前节点的优先级比根大就旋转，如果当前节点是根的左儿子就右旋如果当前节点是根的右儿子就左旋.对于删除, 因为 Treap 满足堆性质，所以我们只需要把要删除的节点旋转到叶节点上，然后直接删除就可以了。具体的方法就是每次找到优先级最大的儿子，向与其相反的方向旋转，直到那个节点被旋转到了叶节点，然后直接删除.

TreeMap：

TreeMap 和 TreeSet 的实现机制和方法大同小异, 只不过 data 换做了 entry, 在这里我就不再累述.

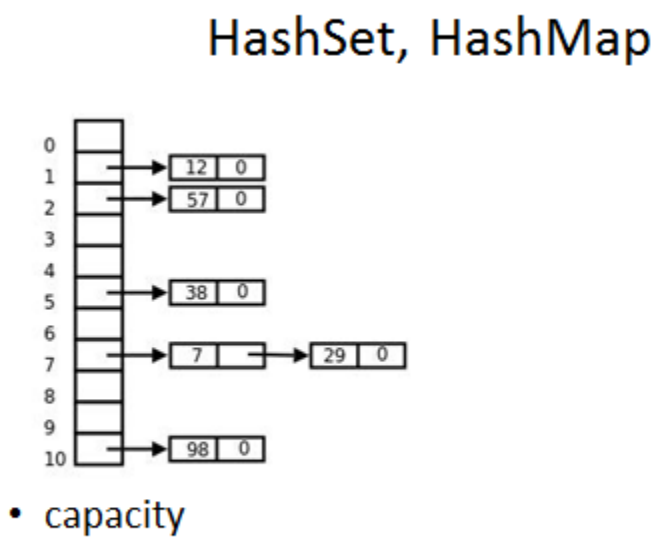


HashSet：

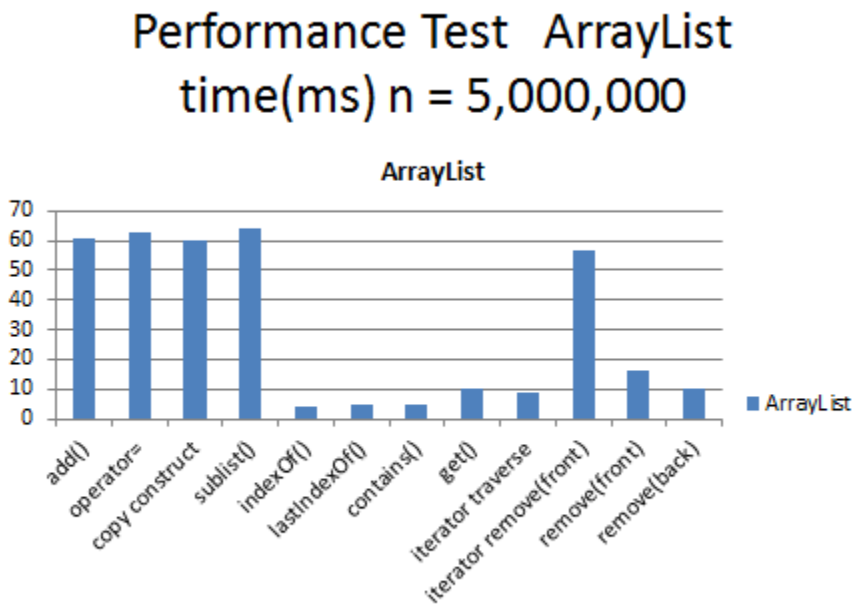
实现 HashSet 我采用的是开散列表的形式，比较特殊的是，我并没有用链表,而是用了之前的 TreeSet. 这样寻找，访问等操作的时间复杂度能小一些. 具体实现，储存一组 TreeSet 即可. 对于哈希表的迭代器，需保存一个指向哈希表的指针，一个TreeSet的迭代器，一个int类型的变量，它的作用是确定已经遍历到了哪个 TreeSet，遍历的时候只需一次从 0 到 capacity，分别遍历 TreeSet. 对于插入操作每次需要找到哈希表所在位置，然后调用 TreeSet 的插入操作，查询操作，删除操作也是如此.

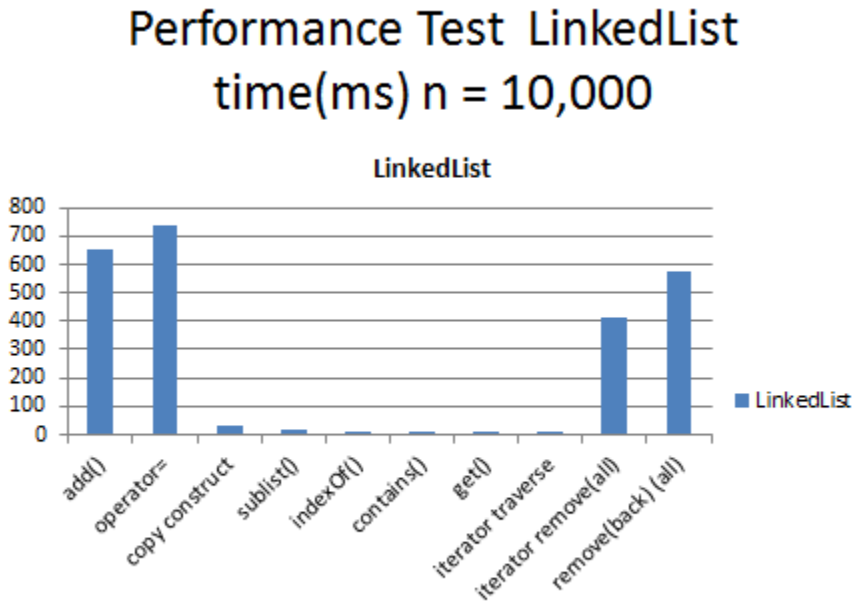
HashMap：

实现方式和 HashSet 更是大同小异，不再累述.

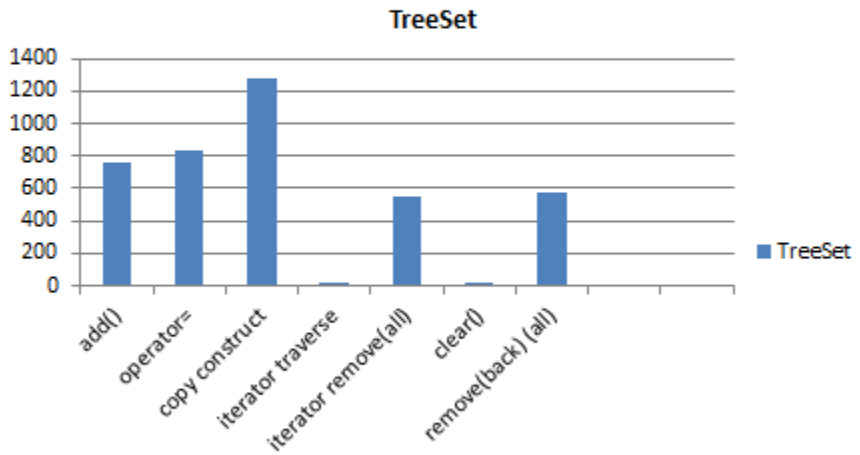


性能测试和比较:

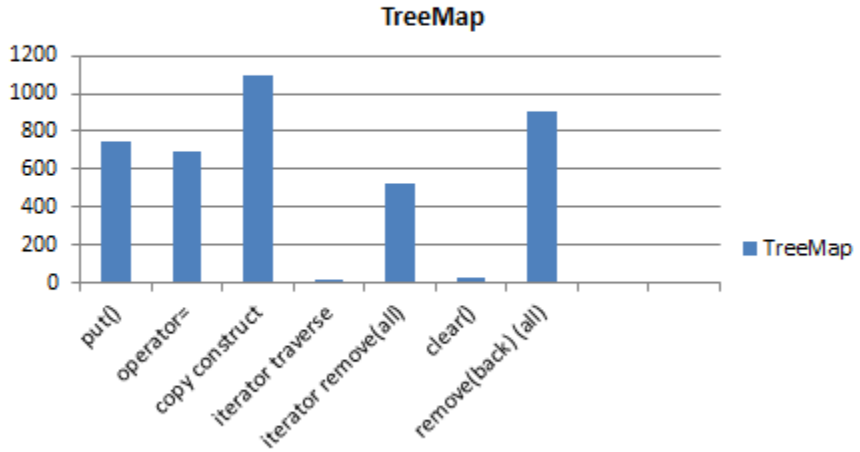

Performance Test ArrayList
time(ms) n = 5,000,000
ArrayList
70
60
50
40
30
20
10
0
add()
operator=
copy construct
sublist()
indexOf()
lastIndexOf()
contains()
get()
iterator traverse
iterator remove(front)
remove(front)
remove(back)
ArrayList


Performance Test LinkedList
time(ms) n = 10,000
LinkedList
800
700
600
500
400
300
200
100
0
add()
operator=
copy construct
sublist()
indexOf()
contains()
get()
iterator traverse
iterator remove(all)
remove(back) (all)
LinkedList

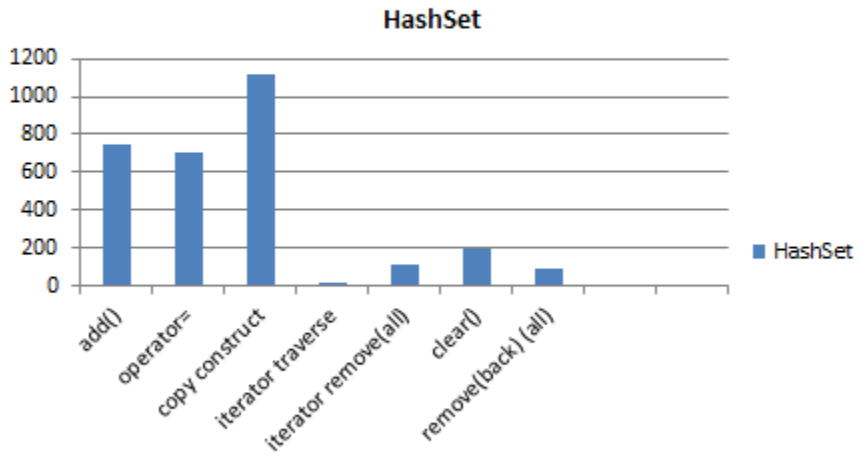
Performance Test TreeSet
time(ms) n = 10,000



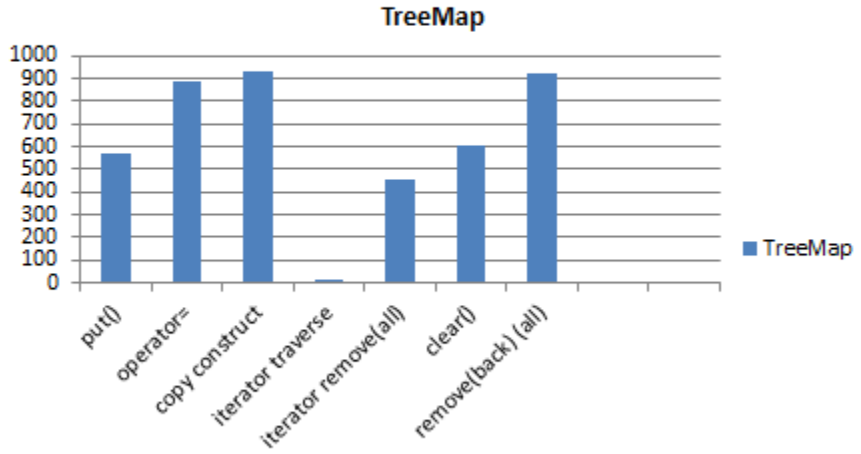
Performance Test TreeMap
time(ms) n = 10,000

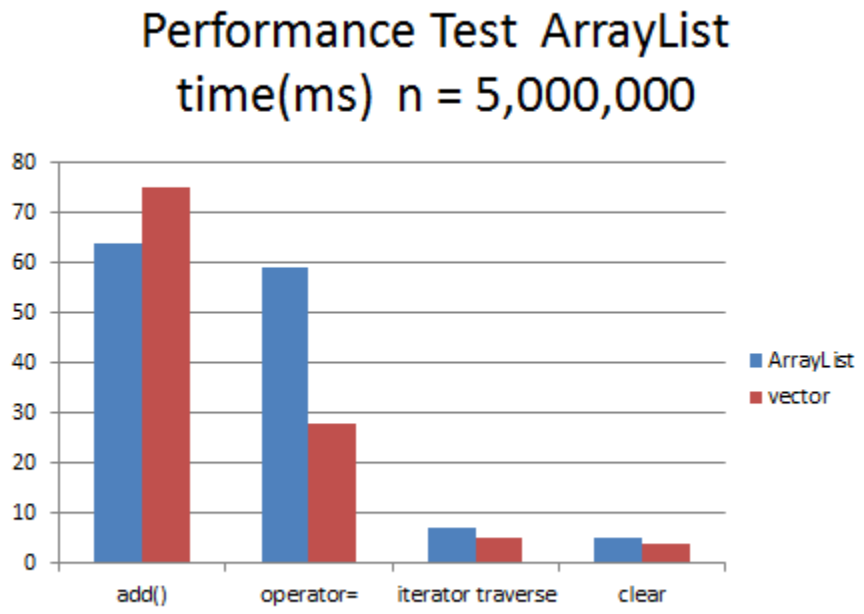


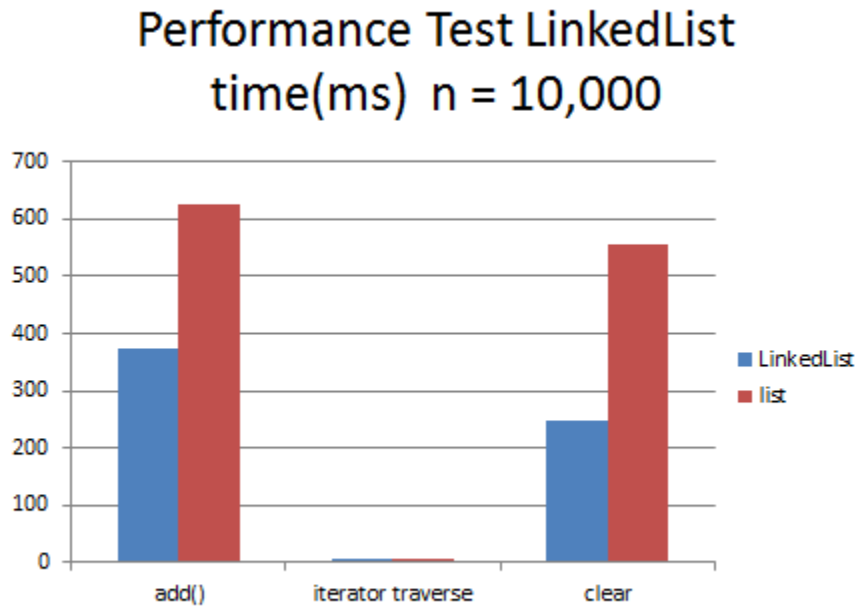
Performance Test HashSet
time(ms) n = 10,000

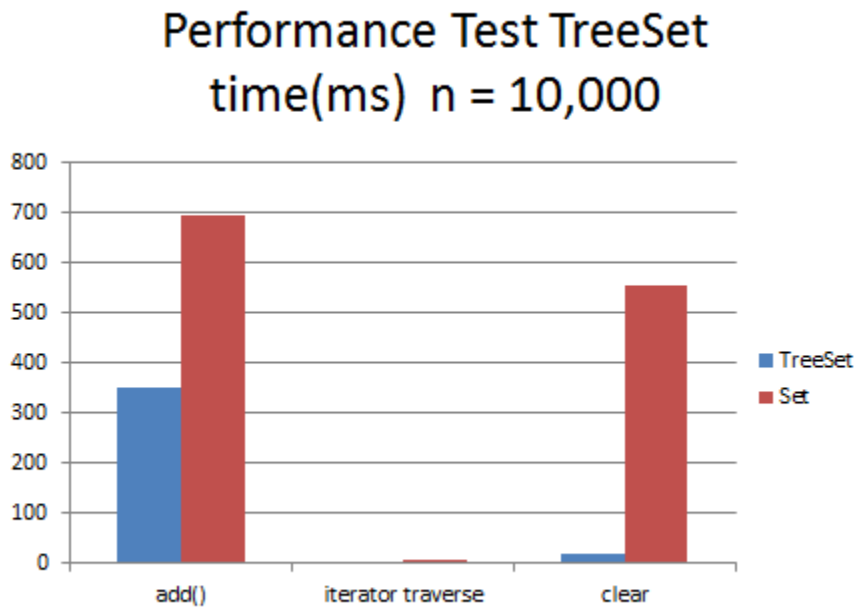


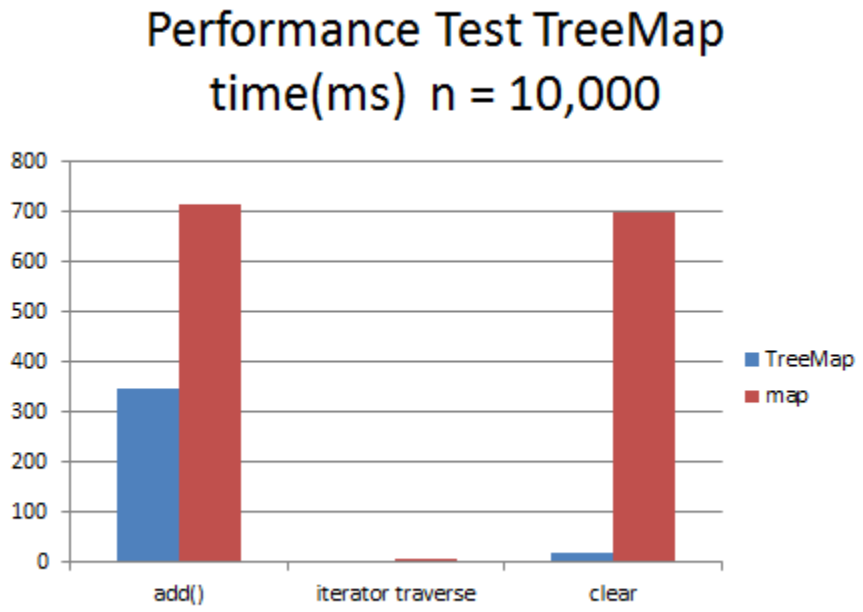
Performance Test HashMap
time(ms) n = 10,000

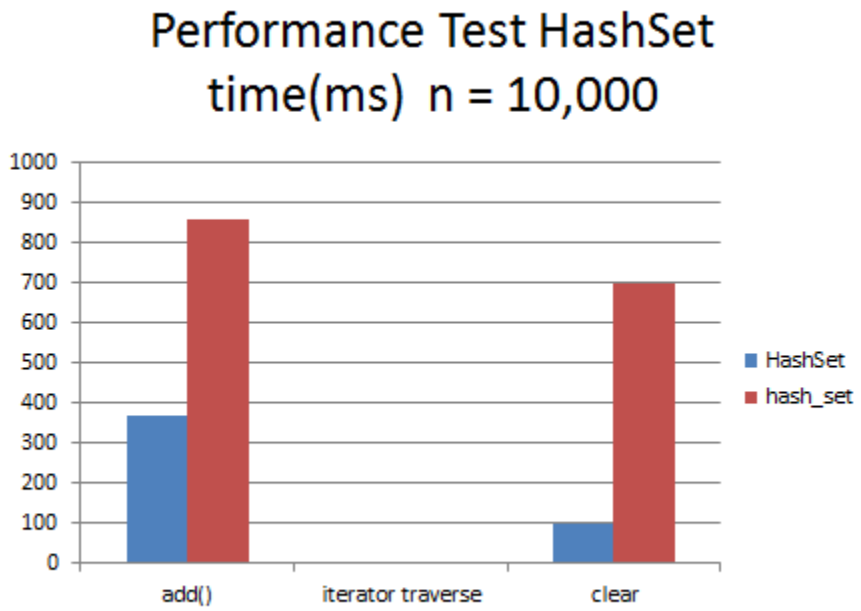


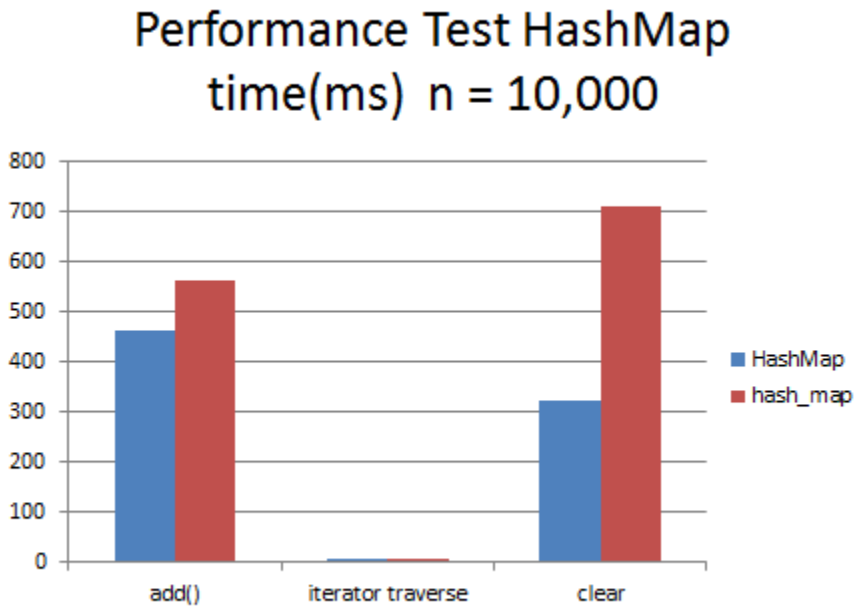

Performance Test ArrayList
time(ms) n = 5,000,000
80
70
60
50
40
30
20
10
0
add()
operator=
iterator traverse
clear
ArrayList
vector


Performance Test LinkedList
time(ms) n = 10,000
700
600
500
400
300
200
100
0
add()
iterator traverse
clear
LinkedList
list


Performance Test TreeSet
time(ms) n = 10,000
800
700
600
500
400
300
200
100
0
add()
iterator traverse
clear
TreeSet
Set


Performance Test TreeMap
time(ms) n = 10,000
800
700
600
500
400
300
200
100
0
add()
iterator traverse
clear
TreeMap
map





从以上测试结果可以看出，寻址，读取等操作的效率很高，可能由于 windows 内存分配的原因，凡是内存操作都很慢，尤其是释放内存. 可以尝试的改进方法有：可以从算法角度提升，不过提升空间不大. 可以用顺序存储，比如用动态数组存储这些结点，这样回收的时候更快一些.