Exercise 100

Describe how to modify each of the linked list algorithms

- if object hash codes are not guaranteed to be unique.

- key = x.hashCode()
- x.hashCode() == y.hashCode()

Linked List L

- Set S(L)

- L.add(x) = $S(L) \cup \{x\}$
- L.remove(x) = $S(L) \setminus \{x\}$
- L.contains(x) = $x \in S(L)$

Add

- “**if (cur.key == key) return false**”修改为在**pred.key <= key and key < curr.key**范围内确定插入位置
- 不需要判断新结点hash值是否等于cur.key，只需判断新结点hash值是否小于或大于等于cur.key

- L.add(x) = S(L) if x∈S(L)
  - S(L)不是多集（multi-set）

Remove/Contains

- **remove() / constains()** 需要遍历链表直至**key**相等时结束
- remove() 应在新结点 hash值小于cur.key时pre、 cur节点后移，新结点hash值等于cur.key时cur节点后移，新结点hash值大于cur.key时令pre.next=cur
  - L.remove(x) = S(L) \ { y | y.hashCode() == x.hashCode() }
- contains() 不需要修改，因为即使有多个hash值相同的结点，该方法仍应返回true
  - L.contains(x) = ∃y∈S(L), y.hashCode() == x.hashCode()

Solution

- while (curr .key < key) 修改为 while (curr .key < key || (curr .key == key && !curr.item.equals(item)))

- 直接比较items，如果items可相互比较（即存在全序）

Exercise 118

- Explain why this cannot happen in the LockFreeList algorithm.
- A node with item x is logically but not yet physically removed by some thread,
  - T1: remove(x)
- then the same item x is added into the list by another thread,
  - T2: add(x)
- and finally a contains() call by a third thread traverses the list, finding the logically removed node, and returning false,
  - T3: contains(x) = false
- even though the linearization order of the remove() and add() implies that x is in the set.

Solution

- LockFreeList
- T1: |----r----| remove(x)
- T2: |----a----| add(x)
- T3: |----c----| contains(x) = false
- T3': |----c----| contains(x) = true
- The thread adding x the second time would physically remove the marked node containing x.

Not A Solution

☹因为contains() --> find() 会忽略所有marked的节点。

☹如果线性化的顺序是remove-->add-->contains，contains().find()会发现curr.key == key而不是停止在marked removed node with key == key。