Recitation 8

Iterable, Iterator, and Examples

Iterable and Iterator

Iterable<E>
iterator();
Returns Iterator<E>

Collection<E>
add(E);
contains(Object);
isEmpty();
remove(Object);
size();
...

Iterator<E>
hasNext();
next();
remove(); (optional)

Returns Iterator<E>

Iterator<Integer> iter = linkedList.iterator();
while (iter.hasNext()) {
    Integer i = iter.next();
    if (i < 0)
        iter.remove();
}

We will not implement remove, but it's still useful to know about.

Example of filtering all negative integers from a LinkedList collection:

Iterator<Integer> iter = linkedList.iterator();
while (iter.hasNext()) {
    Integer i = iter.next();
    if (i < 0)
        iter.remove();
}

Note: Don’t use for data structures that have O(n) remove like ArrayLists!

Syntactic sugar!

LinkedList<String> list;
Iterator<String> iter = list.iterator();
while (iter.hasNext()) {
    String s = iter.next();
    // use s
}

Any object that implements Iterable<E> will be able to use the for-each loop.

Last week: Linear probing

Three possible entries:
1. null
2. A HashEntry object with isInSet false
3. A HashEntry object with isInSet true

Iterate over

HashEntry<String>[] b;

HashSet<T>

public class HashSet<T> { ...
    private HashEntry<T>[] b;
    private int size = 0; //number of active entries in set
    public boolean add(T x) { ... } ...
    public boolean contains(Object x) { ... }
    public boolean remove(Object x) { ... }
    private class HashSetIterator implements Iterator<T> {
        public boolean hasNext() { ... } // Has access to field size
        public T next() { ... }
        public void remove() { ... }
    }
}
HashSetIterator

```java
/** An instance is an Iterator of this HashSet */
private class HashSetIterator implements Iterator<T> {
    // all elements in b[0..pos] have been enumerated
    private int pos = -1;
    // number of elements that have been enumerated
    private int enumerated = 0;
    /** = "there is another element to enumerate". */
    public boolean hasNext() {
        return enumerated != size;
    }
    // continued on next slide
}
```

HashSet<T>

```java
public class HashSet<T> implements Iterable<T> {
    private HashEntry<T>[] b;
    private int size = 0;
    public boolean add(T x) { ... }
    public boolean contains(Object x) { ... }
    public boolean remove(Object x) { ... }
    public Iterator<T> iterator() {
        return new HashSetIterator();
    }
    private class HashSetIterator implements Iterator<T> { ... }
}
```

Don't change the set while iterating!

```java
HashSet<Integer> hs = new HashSet<Integer>();
// Add a bunch of strings to hs;
for (Integer k : hs) {
    hs.add(-k);
}  // This may change array b and int field size. May
   // cause rehash. hs's class invariant (meanings of
   // hs.pos and iter.enumerated) no longer holds.
   // Will get ConcurrentModificationException.
```

Nested Classes: class within class

```java
Nested Static Class:
- Do not live within outer class object
- HashEntry objects do not reference HashSet fields or instance methods.
```

iter.next() vs. list.get(i)

```java
for (Integer i : linkedList) { System.out.println(i); }
```

O(n)

```java
for (int i = 0; i < linkedList.size(); i++) { System.out.println(linkedList.get(i)); }
```

O(n^2)
Data Structure Problems

Review: HashMap

- Key-Value pair relationships
- Uses hashing like HashSet
- Expected: O(1) insert and lookup
- Worst case: O(n) insert and lookup

Example of HashMap:

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>to</td>
<td>2</td>
</tr>
<tr>
<td>be</td>
<td>2</td>
</tr>
<tr>
<td>or</td>
<td>1</td>
</tr>
<tr>
<td>not</td>
<td>1</td>
</tr>
<tr>
<td>that</td>
<td>1</td>
</tr>
<tr>
<td>is</td>
<td>1</td>
</tr>
<tr>
<td>the</td>
<td>1</td>
</tr>
<tr>
<td>question</td>
<td>1</td>
</tr>
</tbody>
</table>

"To be or not to be that is the question"

Review: Binary heap

- Maintains max or min of collection
- Follows heap order invariant at every level
- Always balanced!
- Worst case: O(log n) insert/O(log n) update/O(1) peek/O(log n) removal

Least Recently Used (LRU) Cache

Supported Operations all in O(1) time:
- add(E element)
- update(E element)

A cache (container) should contain at most N elements. When the N+1 element tries to get added, the element that was least recently used gets evicted.

The elements are immutable and distinct.

Finding top k elements in array

Given an array b, print the top k elements.
(Order not important)

<table>
<thead>
<tr>
<th>100</th>
<th>7</th>
<th>3</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

Where k = 3: 100, 7, 10