Lecture 9: Iterators & Linked Lists

CS 62
Spring 2015
Kim Bruce & America Chambers

Assignment 3

- On-disk sorting: What to do when more data than can fit in memory of computer?
- Read info on File I/O in Java and file systems in appendix to assignment. See on-line Streams cheat sheet!
- Lab 3: More complexity/timing

Induction Proofs

- To prove for all n ≥ m, P(n), using
  - Simple induction:
    - Prove P(m)
    - Assuming P(k), prove P(k+1)
  - Strong induction:
    - Assuming P(i) for all m ≤ i < k, prove P(k)
    - May or may not do P(m) as a special case
    - Strong induction useful for divide and conquer algorithms (where algorithm dependent on more than just k-1 case)

Sort Review

- Mergesort:
  - Algorithm: Divide in half, sort each half, then merge them in order
  - Complexity: O(n log n)
    - Proved f(n) ≤ n log n + n by strong induction
    - Why is that O(n log n)?
- Quicksort: New divide & conquer:
  - Algorithm: Move small elts to left, large to right
    Sort left elts, sort right elts, done!
  - Complexity: O(n log n) on average, $O(n^2)$ in worst case

When we write log n in CS, we mean $\log_2 n$

Iterators

- Provide elements of data structure one at a time so can iterate through elts performing operations.
- Interface in standard Java

Iterator in Java

```
public interface Iterator<E> {
  // Returns true if the iteration has more elements.
  boolean hasNext()

  // Returns the next element in the iteration.
  E next()

  /**
   * Removes from the underlying collection the last element
   * returned by this iterator (optional operation).  */
  void remove()
}
```

Iterator Rules

- Remove is optional (we won't use it)
- Only allowed to call it once and then must terminate iteration.
- Never change a collection in middle of an iteration
  - Behavior is officially undefined if do!
  - Iterator often copies data structure before iterating, so changes may not appear to original!

Iterable

- Data structures with an iterator, satisfy interface Iterable:
  - Has method Iterator<T> iterator()
- Example: ArrayList<E> has method
  - Iterator<E> iterator()
- See definition and use of of iterator in ArrayIndexList<E>.

Code using Iterator

```
Iterator<String> listIterator = myList.iterator();

while(listIterator.hasNext()){
      System.out.println(listIterator.next());
}

while(listIterator.hasNext()){
      String elt = listIterator.next()
      System.out.println(elt);
}
```

Can make it even easier!

Iterators and For loops

```
for(String elt: myList){
      System.out.println(elt);
}
```

- Abbreviates previous code!
- Fine as long as myList has an iterator method

List Iterator

- Notice can have two iterators going through list independently!
- Never modify a data structure when iterating through elements as may get unpredictable results.
  - Most classes in Java collection classes have iterators which are designed to "fail fast". Throw an exception if continue with iterator (e.g., next()) after add or delete.

Java 8

- See Iterating over collections in Java 8
- forEach method now in collection classes

```
public void forEach(Consumer<? super E> action)
```

Description copied from interface: Iterable

Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception. Unless otherwise specified by the implementing class, actions are performed in the order of iteration (if an iteration order is specified). Exceptions thrown by the action are relayed to the caller.

Code using forEach

myList.forEach(elt -> {System.out.println(elt);});

lambda expression

- No explicit control over iterator
- Similar to Java 5 built-in for loop
  - but it is a method added by programmer!!
  - Consumer is an interface with method
    void accept (T t)
  - accept method has code to be executed
  - Most valuable when more than one way to traverse
  - May only access effectively final variables from scope

Code

- I added to ArrayIndexList:

```
public void forEach(Consumer <? super E>action) {
   for (E elt: this) {
     action.accept(elt);
   }
}
```

- forEach is "default method" of Iterable interface.
  - Automatically inherited in all classes implementing it.
  - See article for restrictions on default methods — can't access instance variables!

Linked Lists

- Alternate implementation of lists
- Trade-offs in complexity
  - With ArrayList expensive to add at beginning of list
  - Linked lists inexpensive to add early
  - However, slow to access ith element.

Linked List

- Composed of Nodes
  - Think of as pop-beads
  - See code in structure5 library
    - From documentation page!
- SinglyLinkedList (not std Java!)
  - keep track of head and size
  - Extends AbstractList -- look at on own!
    - Vector also extends AbstractList
  - Do algorithms on board!