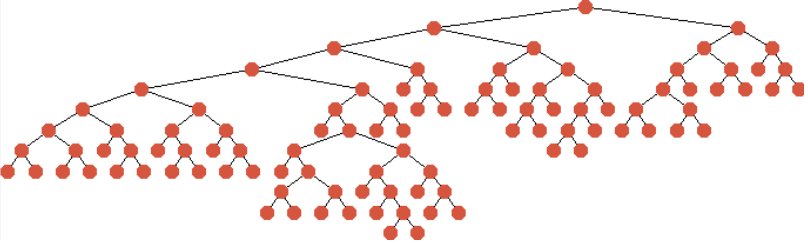

CSSE 230 Day 9

Binary Tree Iterators

After today, you should be able to…
… implement a simple iterator for trees
… implement _lazy_ iterators for trees

Announcements

- Partner Evaluation done?
- Doublets progress?
  - Time for Q/A now.

Binary Tree Iterators

What if we want to iterate over the elements in the nodes of the tree one-at-a-time instead of just printing all of them?

What's an iterator?

- In Java, specified by `java.util.Iterator<E>`

boolean	**hasNext**() Returns `true` if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
void	**remove**() Removes from the underlying collection the last element returned by the iterator (optional operation).

Implement an iterator using our toArrayList().

- Pros: easy to write.
- So let's recall or write toArrayList() now and use it.
- Cons? We'll see shortly!

What's an iterator?

- In Java, specified by **`java.util.Iterator<E>`**

boolean	**hasNext**() Returns `true` if the iteration has more elements.
E	**next**() Returns the next element in the iteration.
void	**remove**() Removes from the underlying collection the last element returned by the iterator (optional operation).

- **`ListIterator<E>`** adds:

boolean	**hasPrevious**() Returns `true` if this list iterator has more elements when traversing the list in the reverse direction.
int	**nextIndex**() Returns the index of the element that would be returned by a subsequent call to `next`.
Object	**previous**() Returns the previous element in the list.
int	**previousIndex**() Returns the index of the element that would be returned by a subsequent call to `previous`.
void	**set**(Object o) Replaces the last element returned by `next` or `previous` with the specified element (optional operation).

Using an Iterator

ag can be any `Collection` of `Integers`, since all Collections implement Iterable

```
for (Integer val : ag)
    sum += val;
System.out.println(sum);
```

The Java compiler translates the above into this:

```
for (Iterator<Integer> itr = ag.iterator(); itr.hasNext(); )
 sum += itr.next();
System.out.println(sum);
```

Why is the ArrayListIterator an inefficient iterator?

- Consider a tree with 1 million elements.
- What is the runtime of iterating over only the first 100 elements?

- (example on board)

- To improve efficiency, the iterator should only get as few elements as possible
  - The one time where being lazy has a reward!

Recall the four types of traversals

- What are they?
- How would you make a lazy **pre-order** iterator? (brainstorm an algorithm now)
- What do you need to add to create the other recursive iterators?
- What about the last iterator?
  - A quick change. Magic? Not really…

Work time

A good goal would be to complete Milestone 1 of BinarySearchTrees by next class