Lecture 17

TreeMap and TreeSet

Last time:

In the previous two lecture notes, we looked at two different usages of binary trees:

- Binary Search Tree
- Huffman Coding (Data Compression)

Having a frequency table of characters and the algorithm, we can build a Huffman tree.

And, with the tree, we can find corresponding codes for each character.

In comparison to the fixed width coding, it is possible to have smaller number of bits for the same length of characters with the Huffman Coding.

Today, let's look at the Java Collections Framework to see two classes of tree data structure: TreeMap and TreeSet.

TreeMap

TreeMap is one of the general-purpose Map implementations. And, it is based on Red-Black tree, which means TreeMap is guaranteed to be balanced. It implements the NavigableMap interface.

As we discussed in previous lectures, HashMap offers the best alternative on average for inserting, deleting, and searching for elements.

But, HashMap has some limits, one of which is that it does not allow us to traverse the elements in any sorted order.

That's where TreeMap comes in handy!

Let's look at its usages.
Remember the following code from Hash in Java lecture?

```
HashMap<String, Integer> freqOfWords = new HashMap<String,
Integer>();
String[] words = "coming together is a beginning keeping together
is progress working together is success".split(" ");

for(String word : words) {
      Integer frequency = freqOfWords.get(word);
      if(frequency == null)
            frequency = 1;
      else
            frequency++;

      freqOfWords.put(word, frequency);
}
```

How do we print all of the keys in a sorted order?

```
TreeMap<String, Integer> sortedWords = new TreeMap<String,
Integer>(freqOfWords);
System.out.println(sortedWords);
```

How about descending order?

```

System.out.println(_____________________________________);
```

In comparison to HashMap, TreeMap provides many other methods. The following is the list of some important methods:

- ceilingKey(key)
- floorKey(key)
- higherKey(key)
- lowerKey(key)
- pollFirstEntry()
- pollLastEntry()
- firstEntry()
- lastEntry()
- firstKey()
- lastKey()
- descendingMap()

TreeSet

TreeSet is one of the general-purpose Set implementations. It implements NavigableSet interface and is based on TreeMap.

Note!

It is generally faster to add elements to HashSet. And, if you need to traverse the elements in sorted order, you can use TreeSet.

Once again, a TreeSet is always balanced which means it guarantees _____ running time complexity for the major operations such as insertion, deletion, and search.

```
HashSet<String> distinctWords = new HashSet<String>();

String[] words = "coming together is a beginning keeping together
is progress working together is success".split(" ");

for(String word : words) distinctWords.add(word);

System.out.println(distinctWords.size()+" distinct words.");
System.out.println("They are: "+distinctWords);

TreeSet<String> sortedWords = new TreeSet<String>(distinctWords);
System.out.println(sortedWords);
```

TreeSet also provides some useful methods:

- ceiling(e)
- floor(e)
- higher(e)
- lower(e)
- pollFirst()
- pollLast()
- first()
- last()
- headSet(e)
- tailSet(e)
- descendingSet()

You are encouraged to take a moment to look at the API documents of TreeMap and TreeSet to check other available methods.

The other thing you need to remember is that **all of the elements in TreeSet and TreeMap must be sortable.**

That means, when you create your own class, you need to properly implement Comparable interface or Comparator interface.

Example application using TreeSet

Suppose you own a bus transportation company and one major route that your company operates is between Pittsburgh and New York City.

The company has its own website where customers can find the schedule.

You found out that customers want to have the following lookups from the schedule.

- What is the last bus that leaves Pittsburgh before 4 pm?
- What is the first bus that leaves Pittsburgh after 10 pm?

What data structure would you use and how would you implement?

For now, to make it simpler, we assume that all of the times are in military time. (For example, 4 pm is 1600)

```
// set up the daily bus schedule
TreeSet<Integer> schedule = new TreeSet<Integer>();
schedule.add(1223);
schedule.add(1430);
schedule.add(1545);
schedule.add(1610);
schedule.add(1705);
schedule.add(2010);
schedule.add(2215);
schedule.add(2320);
schedule.add(2430);

// show the last bus that leaves PGH before 4 pm
System.out.println(schedule.headSet(1600).last());

// show the first bus that leaves PGH after 10 pm
System.out.println(schedule.tailSet(2200).first());
```