Motivating Question

How can we create a time-efficient implementation of heaps?

1 Implementing Heaps

In last lecture, we talked about the algorithms to insert and delete items in a heap. We said that if the heap was balanced, then insertion and deletion would take log time in the number of items in the heap. Now, we have to figure out how to implement heaps to achieve that time.

Heaps are a form of binary tree. Therefore, we can imagine using a basic binary tree implementation to also implement heaps.

1.1 Reminder: Binary Tree Classes in Java

As a reminder, here are the Java classes and interface that we used for binary trees:

```java
interface IBinTree {}

class EmptyBT implements IBinTree {}

class NonEmptyBT implements IBinTree {
    int value;
    IBinTree left;
    IBinTree right;
}
```
1.2 Converting the Binary Tree Classes to Scala

Let’s map these over to Scala.

```scala
trait IBinTree {
}
case object EmptyBT extends IBinTree
case class NonEmptyBT(
  data: Int,
  left: IBinTree,
  right: IBinTree) extends IBinTree
```

1.2.1 Case Classes

This definition uses a new feature of Scala called *case classes*. Case classes are useful for implementing recursive datatypes with different cases (like binary trees, or programming language syntax trees). In particular, they have several features:

- constructor params are val by default (contents are immutable)
- constructors can be used without `new`
- equals methods are provided automatically that compare the internal fields, rather than just the outermost memory addresses

Here’s an example that illustrates the use of case classes and the automatically-created `equals` method:

```scala
object BTExamples {
  val bt1 = NonEmptyBT(5,
    NonEmptyBT(4, EmptyBT, EmptyBT),
    EmptyBT)
  val bt2 = NonEmptyBT(5,
    NonEmptyBT(4, EmptyBT, EmptyBT),
    EmptyBT)
}
```

At the console, if we run `BTExamples.bt1.equals(BTExamples.bt2)`, we get `true`. In contrast, if we tried this with a non-case class and we had not written a custom `equals` method, we would get `false`.

1.3 A Heap Using Case-Class Trees

In lecture, we will discuss implementing a heap using a binary tree. Here is the interface we eventually want to support for heaps:
trait IHeap {
    def getMax: Option[Int]
    def insert(newElt: Int): Unit
    def deleteMax(): Option[Int]
}

Here are three proposals of how to connect the IHeap interface to the binary tree classes. Think about which you prefer and why.

// OPTION 1: put the heap/tree relationship in the interface
trait IHeap extends IBinTree {
    def getMax: Option[Int]
    def insert(newElt: Int): Unit
    def deleteMax(): Option[Int]
}

// OPTION 2: put the heap/tree relationship on the (eventual) Heap class
class Heap extends IBinTree with IHeap { ... }

// OPTION 3: put the tree within the (eventual) Heap class
class Heap extends IHeap {
    var theHeap: IBinTree = EmptyBT
    ...
}

2 Lecture Preparation Tasks

Prior to lecture, review the provided code and consider the following questions. Record your answers to all but the first in the Canvas setup survey for this lecture.

- Post questions about the Scala code to Piazza.
- Which of the three proposals for relating heaps and binary trees do you prefer?
- The class outlines in these notes don’t say anything about priorities, which are central to heaps. Where should heap-specific details, such as priorities, show up in the class definitions (not the methods, just in the classes and fields)?
- Imagine that you were asked to build a heap of \( N \) numbers, using the above classes for the implementation. How many slots in memory would you use, if each number and reference to another item in memory takes one slot?
- Imagine that you were asked to build a heap of \( N \) security alerts, using the above classes for the implementation. How many slots in memory would you use, if each number and reference to another item in memory takes one slot?
Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS18 document by filling out the anonymous feedback form: https://cs.brown.edu/courses/cs018/feedback.