Sets, Searching, and Balancing

Gimme, gimme yer homework!

Interface for a Set

Searching for Items

Optimal Searching for Items
Sets

Basic operation is membership: requires only Eq

Implies $O(n)$ membership test

Check all members for equality to item being tested

By assuming Ord, can change $O(n)$ to $O(\log n)$

```java
public interface Set {
    boolean member(Comparable i); // ....
}
```

Other useful operations: intersect, union, difference, negation

```java
Set intersect(Set s);
Set union(Set s);
Set difference(Set s); } // negation different
```
Meaning of Operations

Intersection

\[ A \cap B \rightarrow \{ \forall e :: (e \in A, e \in B) \} \]

intersect a b = [e | member a e, member b e]

Union

\[ A \cup B \rightarrow \{ \forall e :: (e \in A \parallel B) \} \]

union a b = [e | e<-a++b]

Difference

difference a b = [e | e<-a, not member b e]
Implementing a Set

Binary search, like a phone book!

Open to middle
Is middle the item? If not,
Item less than middle, recursively look in front
Item greater than middle, recursively look in back

```java
boolean member(Comparable i) {
    int order = i.compareTo(this.median());
    if (order == 0) return (length > 0 ? true : false)
    else if (order < 0) return this.frontHalf().member(i)
    else return this.backHalf().member(i); }
```
Binary Trees

Binary tree is empty or is a pair of binary trees

Choice:

a. Items at “empty” nodes (leaves)?
b. Or at internal nodes?
c. Or both?

a => O(n log n) space
b => O(2n) => O(n) space
c => O(n) space
Binary Search in Binary Trees

For searching, only b or c make sense

```
data binaryTree a = E a
                 | Node (binaryTree a) a (binaryTree a)
member :: binaryTree a -> a -> bool
member (E b) a = a==b
member (Node left b right) a =
  if (a==b)
    then true
    else if (a<b)
      then member left a
    else member right a

What's the worst case performance?
```
Hence, Balanced Trees

A tree with $n$ items is balanced if the longest route is $O(\log n)$

Balancing a binary tree requires non-local information — Yucky!

So, let’s be clever....

Construct tree s.t. it “stays flat”
2-3-4 Trees

Use “broader” nodes to flatten tree

Instead of looking just left or right, try

Left, center, or right

Left, center left, center right, or right

I.e., 2, 3, or 4 subtrees to choose from

At most, twice as many subtrees to choose from

a constant factor

but must make sure to broaden nodes when we can
A Definition

data T234 a =
    L
    | N2 (T234 a) a (T234 a)
    | N3 (T234 a) a (T234 a) a (T234 a)
    | N4 (T234 a) a (T234 a) a (T234 a) a (T234 a)

Leaf with value

or 2,3,4 subtrees with 1,2,3 values
Membership

member L i = false
member (N2 left a right) i
  | a==i = true
  | i<a = member left i
  otherwise = member right i
member (N3 left a middle b right) i
  | a==i || b==i = true
  | i<a = member left i
  | i>b = member right i
  otherwise = member middle i
member (N4 left a midl b midr c right) i
  | i==a || i==b || i== c = true
  | i<a = member left i
  | i>c = member right i
  | i<b = member midl i
  otherwise = member midr i
**Insertion**

Using for sets, so if i is a member, we’re done

i.e., we *will* reach a L

If we recurse to a 4 when we insert, subdivide the four into three twos, then continue

At the end, replace a L with a 2

If its parent is a 2, upgrade to a 3

If its parent is a 3, upgrade to a 4
**Insertion in 4-node**

\[
\text{insert } t@(N4 \text{ left } a \text{ midl } b \text{ midr } c \text{ right}) \ i \\
| \ i=a \ || \ i=b \ || \ i=c = t \\
\text{default} = \text{insert splitTree} \ i \\
\text{where splitTree} = \\
N2 \ (N2 \text{ left } a \text{ midl}) \ b \ (N2 \text{ midr } c \text{ right})
\]
Insertion into Leaf

\[
\text{insert } L \text{ a } i \\
\quad | \ i==a = L \ a \\
\quad | \ i<a = N2 (L \ i) \ a \ (L \ a) \\
\text{default } = N2 (L \ a) \ i \ (L \ i) \\
\]

-- alternate: empty leaf
insert \ L \ i = N2 \ L \ i \ L
Insertion into 2-Node

\[
\text{insert } t@(N2 \text{ left } a \text{ right}) \ i \\
\begin{align*}
| \ i= & a = t \\
| \ i<& a = \text{let } l = \text{insert left } i \text{ in} \\
& \quad \text{case } (l) \text{ of} \\
& \quad \quad N2 \ 12 \ b \ r2 = N3 \ 12 \ b \ r2 \ a \ right \\
& \quad \quad N3 \ 12 \ b \ m2 \ c \ r2 = N4 \ 12 \ b \ m2 \ c \ r2 \ a \ right \\
& \quad \quad _ = N2 \ \text{left} \ a \ \text{right} \\
| \ i>& a = \text{let } l = \text{insert right } i \text{ in} \\
& \quad \text{case } (l) \text{ of} \\
& \quad \quad N2 \ 12 \ b \ r2 = N3 \ \text{left} \ a \ 12 \ b \ r2 \\
& \quad \quad N3 \ 12 \ b \ m2 \ c \ r2 = N4 \ \text{left} \ a \ 12 \ b \ m2 \ c \ r2 \\
& \quad \quad _ = N2 \ \text{left} \ a \ \text{right}
\end{align*}
\]
Insertion into 3-Node

\[
\text{insert } t@(N3 \text{ left } a \text{ mid } b \text{ right}) \text{ i}
\]

| i==a | i==b = t
| i<a = let l = insert left i in case (l) of
  \[N2 \text{ lt } c \text{ gt } = N4 \text{ lt } c \text{ gt } a \text{ mid } b \text{ right}\]
  \[N3 \text{ lt } c \text{ md } d \text{ gt } =\]
  \[N4 \text{ (N2 } \text{ lt } c \text{ md } d \text{ gt } a \text{ mid } b \text{ right}\]
  _ = t
| i>b = let l = insert right i in case (l) of
  \[N2 \text{ lt } c \text{ gt } = N4 \text{ left } a \text{ mid } b \text{ lt } c \text{ gt}\]
  \[N3 \text{ lt } c \text{ md } d \text{ gt } =\]
  \[N4 \text{ left } a \text{ mid } b \text{ lt } c \text{ (N2 } \text{ md } d \text{ gt )}\]
  _ = t
| default = let l = insert mid i in case (l) of
  \[N2 \text{ lt } c \text{ gt } = N4 \text{ left } a \text{ lt } c \text{ gt } b \text{ right}\]
  \[N3 \text{ lt } c \text{ md } d \text{ gt } =\]
  \[N4 \text{ left } a \text{ (N2 } \text{ lt } c \text{ md } d \text{ gt } b \text{ right}\]
  _ = t
Next Week

Reading: Ch 22 in CLRS