Recitation 7

Combining BSTs

7.1 Announcements

- *FingerLab* is due **Friday afternoon**. It's worth 125 points.
- *RangeLab* will be released on **Friday**.

7.2 Generalized Combination

In lecture, we discussed union, and argued that it has $O\left(m \log\left(\frac{n}{m}+1\right)\right)$ work and $O(\log(n)\log(m))$ span. The latter bound can be improved to $O(\log n + \log m)$ using *futures*[1], but that is outside the scope of this course.

What about the functions intersection and difference? These can be implemented in a similar fashion as union, and as such have the same cost bounds. In this recitation, we'll establish this more concretely.

Task 7.1. *Implement all three functions* union, intersection, *and* difference *in terms of a single helper function* combine *which has* $O\left(m \log\left(\frac{n}{m}+1\right)\right)$ *work and* $O(\log(n)\log(m))$ *span for BSTs of size* n *and* m, $n \geq m$. *Conclude that all three of these functions have the same cost bounds.*

Let's begin by inspecting the code for union.

Algorithm 7.2. *BST union.*

```
fun union $(T_1, T_2)$ =
   case $(T_1, T_2)$ of
      $(\_, Leaf) \Rightarrow T_1$
   | $(Leaf, \_) \Rightarrow T_2$
   | $(Node\ (L_1, x, R_1), \_) \Rightarrow$
         let val $(L_2, \_, R_2)$ = split $(T_2, x)$
               val $(L, R)$ = (union $(L_1, L_2)$ || union $(R_1, R_2)$)
         in joinMid $(L, x, R)$
         end
```

What do we have to change to generalize this? Notice that, for example, intersection returns Leaf in both base cases, while difference only returns Leaf in the second case. Next, consider that intersection only keeps the key x if it is also present in T_2, and difference specifically removes x if it is present in T_2. We can account for all of these differences by introducing new arguments which specify what to do in the base cases, and whether or not we should keep x in the recursive case (based on whether or not it is present in T_2).

[1] http://dl.acm.org/citation.cfm?id=258517

Algorithm 7.3. *Generalized BST combine.*

```
fun combine $f_1$ $f_2$ $k$ =
  let
    fun combine′ $(T_1, T_2)$ =
      case $(T_1, T_2)$ of
        (_, Leaf) ⇒ $f_1(T_1)$
      | (Leaf, _) ⇒ $f_2(T_2)$
      | (Node $(L_1, x, R_1)$, _) ⇒
          let val $(L_2, y, R_2)$ = split $(T_2, x)$
              val $(L, R)$ = (combine′ $(L_1, L_2)$ || combine′ $(R_1, R_2)$)
          in if $k(y)$ then joinMid $(L, x, R)$ else join $(L, R)$
          end
  in
    combine′
  end

val union =
  combine (fn $T_1$ ⇒ $T_1$) (fn $T_2$ ⇒ $T_2$) (fn $y$ ⇒ true)

val intersection =
  combine (fn $T_1$ ⇒ Leaf) (fn $T_2$ ⇒ Leaf) (fn $y$ ⇒ isSome $y$)

val difference =
  combine (fn $T_1$ ⇒ $T_1$) (fn $T_2$ ⇒ Leaf) (fn $y$ ⇒ not isSome $y$)
```

Task 7.4. *Consider a function* symdiff *where* (symdiff (A, B)) *returns a BST containing all keys which are either in A or B, but not both. Implement* symdiff *in terms of* combine*.*

```
val symdiff = combine (fn $T_1$ ⇒ $T_1$) (fn $T_2$ ⇒ $T_2$) (fn $y$ ⇒ not isSome $y$)
```

.