Red Black Trees Examples

datatype Colour = B | R
datatype Tree = E | T of Colour * int * Tree * Tree
val member = fn : int -> Tree -> bool
val balance = fn : Colour * int * Tree * Tree -> Tree
val insert = fn : int -> Tree -> Tree
val it = () : unit
  - insert 4 E;
val it = T (B,4,E,E) : Tree
  -

Why?
Red Black Trees Examples

insert 4 E =
let
 val ins = T(R,4,E,E)
 val T(_,y,a,b) = ins s
 in T(B,y,a,b)
 = T(B,4,E,E)
Red Black Trees Examples

Continued from previous slide:

- insert 5 it;
val it = T (B,4,E,T (R,5,E,E)) : Tree
-
Red Black Trees Examples

\[
\text{insert } 5 \ T(B,4,E,E) = \\
\text{let } \\
\quad \text{val ins}(s \text{ as } T(B,4,E,E)) = \\
\quad \quad \text{balance}(B,4,E,\text{ins } E) \\
\quad \text{val } T(_,y,a,b) = \text{ins } s \\
\quad \text{in } T(B,y,a,b)
\]

So we have to find out what \text{ins } E is.
Red Black Trees Examples

ins E = T(R,5,E,E)

This means the call to balance is :-

balance(B,4,E,T(R,5,E,E))

But since we don’t have two reds in a row we simply get the tree returned. So we get the result.
Red Black Trees Examples

Now things get more complicated:

- insert 6 it;
val it = T (B,5,T (B,4,E,E),T (B,6,E,E)) : Tree
-

Notice the tree has been re-balanced in the ordinary search tree we would have got something different.
Red Black Trees Examples

insert 6 \(T(B,4,E, T(R,5,E,E)) =\)
let val ins(s as T(B,4,E, T(R,5,E,E)) =
  balance(B,4,E,ins T(R,5,E,E))
val T(_,y,a,b) = ins s
in T(B,y,a,b)
end

So we have to work out ins T(R,5,E,E)
Red Black Trees Examples

\[
\text{ins } T(R,5,E,E) = \\
\text{balance}(R,5,E,\text{ins } E) = \text{balance}(R,5,E,T(R,6,E,E)) = \\
T(R,5,E,T(R,6,E,E))
\]

Again we don’t match any of the clauses for balance since we don’t have a black followed by two reds.
But we know now that :-

\[
\text{balance}(B,4,E,\text{ins } T(R,5,E,E)) = \\
\text{balance}(B,4,E, T(R,5,E,T(R,6,E,E))
\]

Black followed by two reds.
Red Black Trees Examples

Matching balance clause.

balance(B,4,E, T(R,5,E,T(R,6,E,E))

Matches :-

balance(B,x,a,T(R,y,b,T(R,z,c,d))

So we get :-

T(R,5,T(B,4,E,E),T(B,6,E,E)

Then by relabelling the top node black we get the answer.
Red Black Trees Summary

- Insert a new node labelled red with the value in the correct place as you would with a binary search tree.

- Remove Black, Red, Red violations replacing them with a red node with two black children.

The net effect is that the tree remains balanced.
Implementation Approaches - Persistence versus updating

val a = [1,2,3] : int list
- val b = map (fn x=>x+1) a;
GC #0.0.0.1.3.45: (20 ms)
val b = [2,3,4] : int list
-

The list a stills stays around. ML works by creating a copy of the list a.

ML only copies the bits that you need to copy, sharing the bits that don’t need to be copied.

Garbage collection gets rid of the bits you don’t use.
Implementation Approaches - Persistence versus updating

In C, the natural style is to create data-structures and modify them in place.

For example (not tested)

```c
int add_one(int a[], int size) {
    int i;
    for (i = 0; i < size; i++) {
        a[i] = a[i] + 1
    }
}
```

In C it is often hard to make persistent data-structures. In ML it is more natural to make data-structures persistent and sometimes make the code easier to write. Compare the red-black tree in the book.