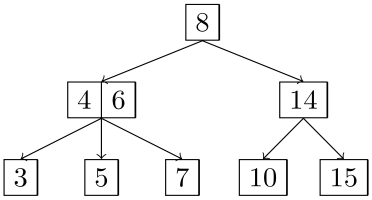
CS 61B
Fall 2020

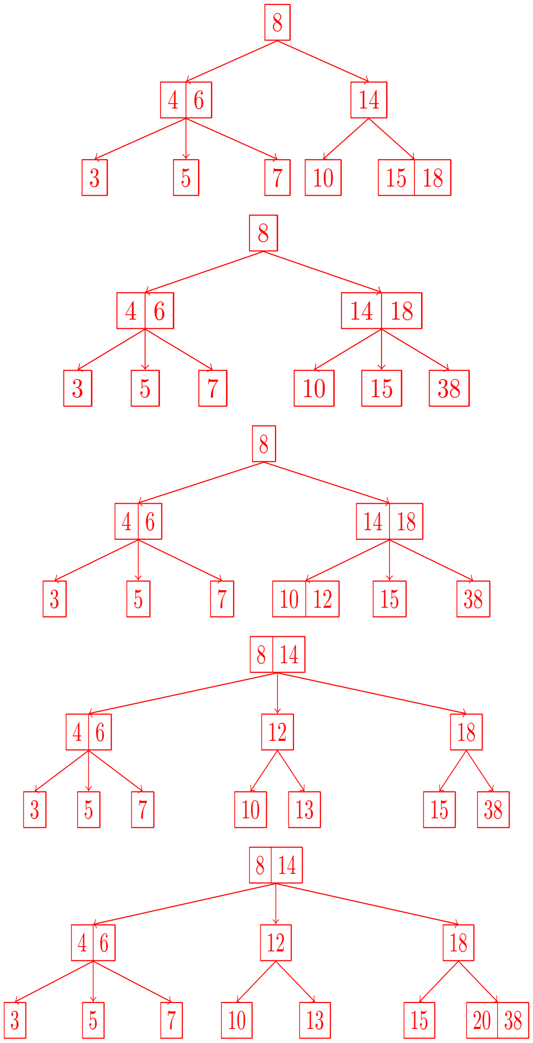
LLRBs, Hashing

Discussion 8: October 12, 2020

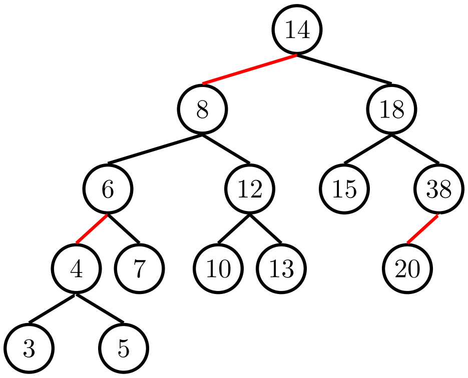
1 2-3 Trees and LLRB's

(a) Draw what the following 2-3 tree would look like after inserting 18, 38, 12, 13, and 20.





(b) Now, convert the resulting 2-3 tree to a left-leaning red-black tree.



(c) If a 2-3 tree has depth H (that is, the leaves are at distance H from the root), what is the maximum number of comparisons done in the corresponding red-black tree to find whether a certain key is present in the tree?

$2H + 2$ comparisons.

The maximum number of comparisons occur from a root to leaf path with the most nodes. Because the height of the tree is H, we know that there is a path down the leaf-leaning red-black tree that consists of at most H black links, for black links in the left-leaning red-black tree are the links that add to the height of the corresponding 2-3 tree. This means that there are $H + 1$ nodes on the path from the root to the leaf, since there is one less link than nodes,

In the worst case, in the 2-3 tree representation, this path can consist entirely of nodes with two items, meaning in the left-leaning red-black tree representation, each blank link is followed by a red link. This doubles the amount of nodes on this path from the root to the leaf.

This example will represent our longest path, which is $2H + 2$ nodes long, meaning we make at most $2H + 2$ comparisons in the left-leaning red-black tree.

2 Hashing

(a) Here are three potential implementations of the `Integer`'s `hashCode()` function. Categorize each as either a valid or an invalid hash function. If it is invalid, explain why. If it is valid, point out a flaw or disadvantage.

```
public int hashCode() {
    return -1;
}
```

Valid. As required, this hash function returns the same `hashCode` for Integers that are `equals()` to each other. However, this is a terrible hash code because collisions are extremely frequent (collisions occur 100% of the time).

```
public int hashCode() {
    return intValue() * intValue();
}
```

Valid. Similar to (a), this hash function returns the same `hashCode` for integers that are `equals()`. However, integers that share the same absolute values will collide (for example, $x = 5$ and $x = -5$ will have the same hash code). A better hash function would be to just return the `intValue()` itself.

```
public int hashCode() {
    return super.hashCode();
}
```

Invalid. This is not a valid hash function because integers that are `equals()` to each other will not have the same hash code. Instead, this hash function returns some integer corresponding to the integer object's location in memory.

(b) For each of the following questions, answer **Always**, **Sometimes**, or **Never**.

1. When you modify a key that has been inserted into a `HashMap` will you be able to retrieve that entry again? Explain.

   Sometimes. If the `hashCode` for the key happens to change as a result of the modification, then we won't be able to reliably retrieve the key.

2. When you modify a value that has been inserted into a HashMap will you be able to retrieve that entry again? Explain.

   Always. The bucket index for an entry in a `HashMap` is decided by the key, not the value. Mutating the value does not affect the lookup procedure.

3 Even More Asymptotics *Extra*

Give the runtime of the following functions in theta notation.

(a) $\Theta(\log\log N)$

```java
public static void f1(int N) {
    for (int i = 2; i < N; i *= i) { }
    System.out.println("Hi");
}
```

You can see that i follows the pattern of $2^{2^0}, 2^{2^1}, 2^{2^2}, ..., 2^{2^{\log\log N}}$ and 1 work is done at each step (due to the multiplication).

$\Theta(\sum_{i=0}^{\log\log N} 1) = \Theta\log\log n$

(b) $\Theta(2^N)$

```java
public static void f2(int N) {
    for (int i = 0; i < N; i++) {
        int jLimit = Math.pow(2, i + 1) - 1;
        for (int j = 0; j < jLimit; j += 2) {
            System.out.println("Hi");
        }
    }
}
```

$\Theta(\sum_{i=0}^{N}(0.5)2^i) = \Theta((0.5)2^{N+1}) = \Theta(2^N)$

(c) *This problem is really hard and not in scope but its fun.*
$\Theta(N^2 logN)$

```java
public static void f1(int N) {
    for (int i = 0; i < N * N; i++) {
        for (int j = 1; j < i; j *= 2) {
            System.out.println("Hi");
        }
    }
}
```

$\Theta(\sum_{i=0}^{N*N}\log(i)) = \Theta(\log((N*N)!)) = \Theta(N^2\log(N^2)) = \Theta(N^2\log(N))$

How do we know that $\Theta(\log((N*N)!) == \Theta(N*N\log(N*N))$? See this proof: http://www.mcs.sdsmt.edu/ecorwin/cs372/handouts/theta_n_factorial.htm