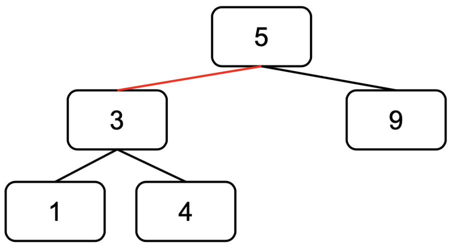
CS 61B
Fall 2020

LLRBs and Hashing

Exam Prep Discussion 8: October 12, 2020

1 LLRB Insertions

Given the LLRB below, perform the following insertions and draw the final state of the LLRB. In addition, for each insertion, write the fixups needed in the correct order. A fixup can either be a rotate right, a rotate left, or a color flip. If no fixups are needed, write "Nothing".



1. Insert 7

2. Insert 6

3. Insert 2

4. Insert 8

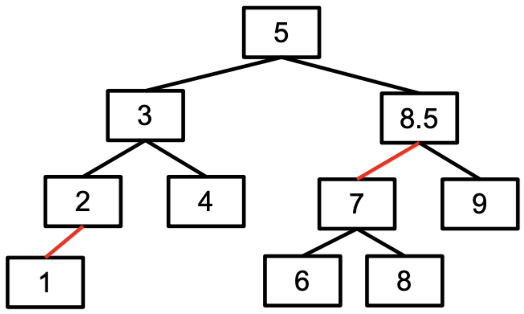
5. Insert 8.5

Final state:

Solution: For a visualization of the process, see **here**

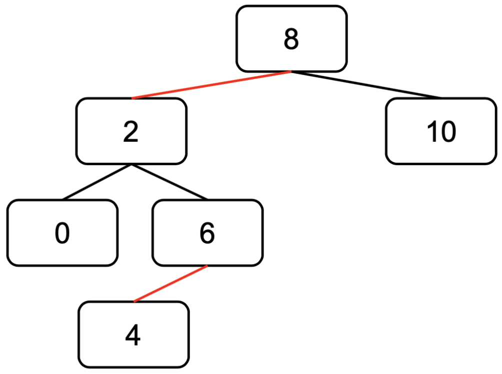
1. Insert 7
   - Nothing
2. Insert 6
   - rotateRight(9)
   - colorFlip(7)
   - colorFlip(5)
3. Insert 2
   - rotateLeft(1)
4. Insert 8
   - Nothing
5. Insert 8.5
   - rotateLeft(8)
   - rotateRight(9)
   - colorFlip(8.5)
   - rotateLeft(7)

Final state:



2 LLRB Maximization

For this problem, we are working with an LLRB containing the integers 0, 2, 4, 6, 8, 10. Choose an integer x for the below configuration of the LLRB such that the insertion of x into the LLRB requires exactly 6 fixups. A fixup can either be a rotate right, a rotate left, or a color flip. The solution must include the integer x and an enumeration of the fixups in the proper order.



Solution:

We insert integer 5 into the configuration below and perform the following steps:

1. Rotate 4 left
2. Rotate 6 right
3. Colorflip 5
4. Rotate 2 left
5. Rotate 8 right
6. Colorflip 5
7. Adjust the color of the root node

3 Hashing Gone Crazy

For this question, use the following TA class for reference -

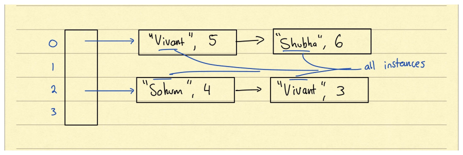
```java
public class TA {
    int charisma;
    String name;
    TA(String name, int charisma) {
        this.name = name;
        this.charisma = charisma;
    }
    @Override
    public boolean equals(Object o) {
        TA other = (TA) o;
        return other.name.charAt(0) == this.name.charAt(0);
    }
    @Override
    public int hashCode() {
        return charisma;
    }
}
```

Assume that the hashCode of a TA object returns charisma, and the equals method returns true if and only if two TA objects have the same first letter in their name.

Assume that the ECHashMap is a HashMap implemented with external chaining as depicted in lecture. The ECHashMap instance begins at size 4 and, for simplicity, does not resize. Draw the contents of map after the executing the insertions below:

```java
ECHashMap<TA, Integer> map = new ECHashMap<>();
TA sohum = new TA("Sohum", 10);
TA vivant = new TA("Vivant", 20);
map.put(sohum, 1);
map.put(vivant, 2);

vivant.charisma += 2;
map.put(vivant, 3);

sohum.name = "Vohum";
map.put(vivant, 4);

sohum.charisma += 2;
map.put(sohum, 5);

sohum.name = "Sohum";
TA shubha = new TA("Shubha", 24);
map.put(shubha, 6);
```

Solution:



4 Buggy Hash

The following classes may contain a bug in one of its methods. Identify those errors and briefly explain why they are incorrect and in which situations would the bug cause problems.

```java
class Timezone {
    String timeZone; // "PST", "EST" etc.
    boolean dayLight;
    String location;
    ...
    public  int currentTime() {
        // return the current time in that time zone
    }
    public int hashCode() {
        return currentTime();
    }
    public boolean equals(Object o) {
        Timezone tz = (Timezone) o;
        return tz.timeZone.equals(timeZone);
    }
}
```

Solution:

Although equal objects will have the same hashcode, but the problem here is that `hashCode()` is not deterministic. This will result in weird behaviors (e.g. the element getting lost) when we try to put or access elements in our hashing data structures.

```java
class Course {
    int courseCode;
    int yearOffered;
    String[] staff;
    ...
    public int hashCode() {
        return yearOffered + courseCode;
    }
    public boolean equals(Object o) {
        Course c = (Course) o;
        return c.courseCode == courseCode;
    }
}
```

Solution: The problem with this `hashCode()` is that not all equal objects have the same hashcode. One key thing to remember is that when we override the `equals()` method, we have to also override the `hashCode()` method to ensure equal objects have the same hashcode.