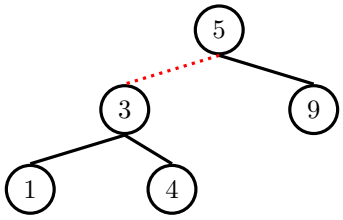
CS 61B Spring 2024

B-Trees, LLRBs, Hashing

Exam-Level 07: March 4, 2024

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 balancing operations needed in the correct order (rotate right, rotate left, or color flip). If no balancing operations are needed, write "Nothing". Assume that the link between 5 and 3 is red and all other links are black at the start.



(a)

1. Insert 7

2. Insert 6

3. Insert 2

4. Insert 8

5. Insert 8.5

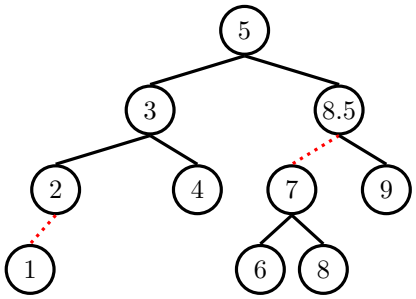
6. Final state

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

1. Insert 7
   - Nothing
2. Insert 6
   - rotateRight(9)

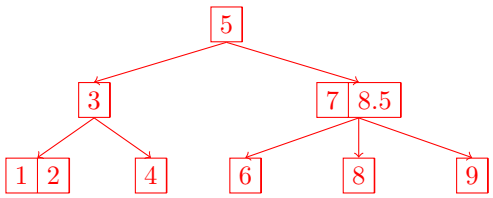
- colorFlip(7)
- colorFlip(5)

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



(b) Convert the final LLRB to its corresponding 2-3 Tree.

Solution:



2 Hashing Gone Crazy

For this question, use the following `TA` class for reference.

```
1   public class TA {
2       int semester;
3       String name;
4       TA(String name, int semester) {
5           this.name = name;
6           this.semester = semester;
7       }
8       @Override
9       public boolean equals(Object o) {
10          TA other = (TA) o;
11          return other.name.charAt(0) == this.name.charAt(0);
12      }
13      @Override
14      public int hashCode() {
15          return semester;
16      }
17  }
```

Assume that the `hashCode` of a `TA` object returns `semester`, 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:

```
1   ECHashMap<TA, Integer> map = new ECHashMap<>();
2   TA jasmine = new TA("Jasmine the GOAT", 10);
3   TA noah = new TA("Noah", 20);
4   map.put(jasmine, 1);
5   map.put(noah, 2);
6
7   noah.semester += 2;
8   map.put(noah, 3);
9
10  jasmine.name = "Nasmine";
11  map.put(noah, 4);
12
13  jasmine.semester += 2;
14  map.put(jasmine, 5);
15
16  jasmine.name = "Jasmine";
17  TA cheeseguy = new TA("Sam", 24);
18  map.put(cheeseguy, 6);
```

Solution:

2 Hashing Gone Crazy

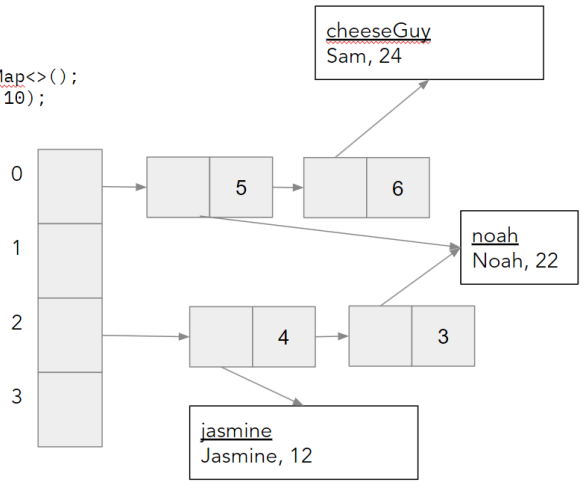
```
ECHashMap<TA, Integer> map = new ECHashMap<>();
TA jasmine = new TA("Jasmine the GOAT", 10);
TA noah = new TA("Noah", 20);
map.put(jasmine, 1);
map.put(noah, 2);

noah.semester += 2;
map.put(noah, 3);

jasmine.name = "Nasmine";
map.put(noah, 4);

jasmine.semester += 2;
map.put(jasmine, 5);

jasmine.name = "Jasmine";
TA cheeseGuy = new TA("Sam", 24);
map.put(cheeseGuy, 6);
```



Explanation:

Line 4: `jasmine` has `semester` value `10`. `10 % 4 = 2`, so `jasmine` is placed in bucket 2 with value `1`.
`0: [], 1: [], 2: [(jasmine, 1)], 3: []`

Line 5: `noah` is placed in bucket 0 with value `2`.
`0: [(noah, 2)], 1: [], 2: [(jasmine, 1)], 3: []`

Line 7: Increasing the `semester` value of `noah` does *not* cause it to be rehashed! (This is why modifying objects in a Hashmap is dangerous–it can change the hashcode of your object and make it impossible to find which bucket it belongs to).

Line 8: `noah` now has `semester` 4, so bucket 2 also has a node pointing to `noah`, with value `3`. (Note that the two `noah`s refer to the same object).
`0: [(noah, 2)], 1: [], 2: [(jasmine, 1), (noah, 3)], 3: []`

Line 11, 12: `noah` with `semester` 22 hashes to bucket 2. However, since we have changed `jasmine`'s name to be `"Nasmine"`, `noah.equals(jasmine)` returns true. Since we are hashing a key that is already present in the dictionary according to `.equals`, we replace `jasmine`'s old value with the new value, `4`.
`0: [(noah, 2)], 1: [], 2: [(jasmine, 4), (noah, 3)], 3: []`

Line 13, 14: `jasmine` with `semester` 12 hashes to bucket 0. However, since we have changed `jasmine`'s name to be `"Nasmine"`, `jasmine.equals(noah)` returns true. Since we are hashing a key that is already present in the dictionary according to `.equals`, we replace `noah`'s old value with the new value, `5`.
`0: [(noah, 5)], 1: [], 2: [(jasmine, 4), (noah, 3)], 3: []`

Line 16, 17, 18: `cheeseGuy` hashes to bucket 0. `cheeseGuy.equals(noah)` returns `false`, so we add a new node after `noah` with value 6.
`0: [(noah, 5), (cheeseGuy, 6)], 1: [], 2: [(jasmine, 4), (noah, 3)], 3: []`

3 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.

```
1   class Timezone {
2       String timeZone; // "PST", "EST" etc.
3       boolean dayLight;
4       String location;
5       ...
6       public  int currentTime() {
7           // return the current time in that time zone
8       }
9       public int hashCode() {
10          return currentTime();
11      }
12      public boolean equals(Object o) {
13          Timezone tz = (Timezone) o;
14          return tz.timeZone.equals(timeZone);
15      }
16  }
```

Solution:

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

```
1   class Course {
2       int courseCode;
3       int yearOffered;
4       String[] staff;
5       ...
6       public int hashCode() {
7           return yearOffered + courseCode;
8       }
9       public boolean equals(Object o) {
10          Course c = (Course) o;
11          return c.courseCode == courseCode;
12      }
13  }
```

Solution: The problem with this `hashCode()` is that not all equal objects have the same hashcode. This may produce unexpected behavior, e.g. multiple "equal" objects may be exist in different buckets in the `HashMap`, the `containsKey` operation may return false, etc. 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`.