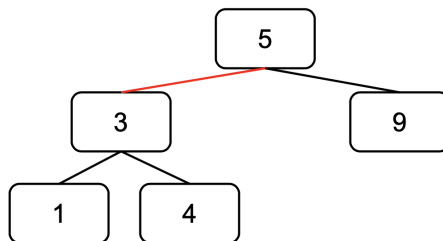


1 LLRB Insertions

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)

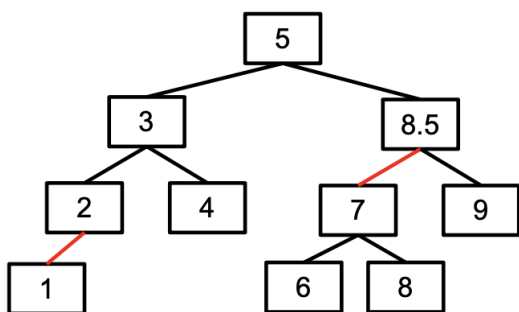
4. Insert 8

- Nothing

5. Insert 8.5

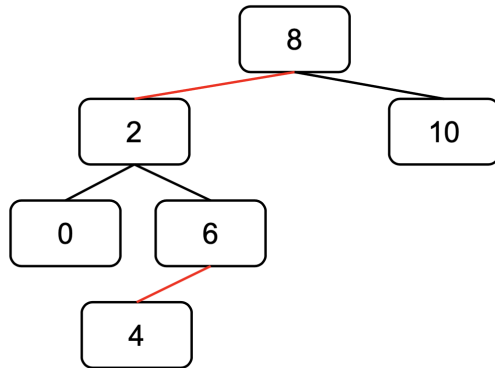
- rotateLeft(8)
- rotateRight(9)
- colorFlip(8.5)
- rotateLeft(7)

Final state:



2 LLRB Maximization

For this problem, we are working with the LLRB below. Determine an integer x 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

3 Hashing Gone Crazy

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

```

1  public class TA {
2      int charisma;
3      String name;
4      TA(String name, int charisma) {
5          this.name = name;
6          this.charisma = charisma;
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 charisma;
16     }
17 }

```

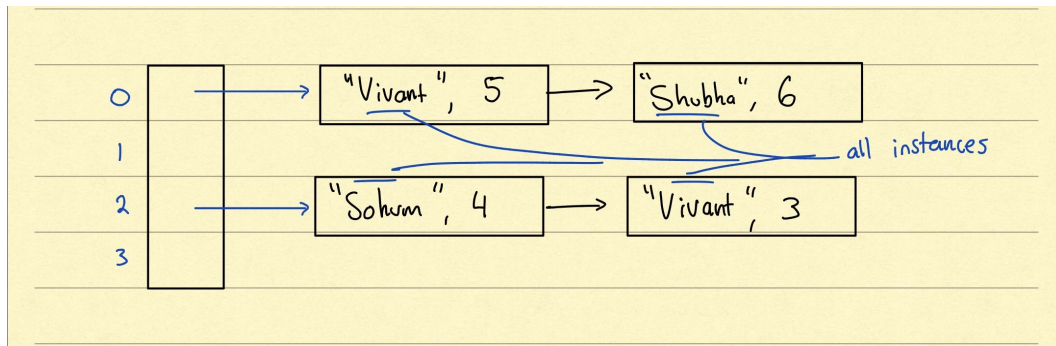
Assume that the hashCode of a TA object returns `charisma`, and the `equals` method returns `true` if and only if two TA objects have the same first letter in their name.

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:

```

1  ECHashMap<TA, Integer> map = new ECHashMap<>();
2  TA sohum = new TA("Sohum", 10);
3  TA vivant = new TA("Vivant", 20);
4  map.put(sohum, 1);
5  map.put(vivant, 2);
6
7  vivant.charisma += 2;
8  map.put(vivant, 3);
9
10 sohum.name = "Vohum";
11 map.put(vivant, 4);
12
13 sohum.charisma += 2;
14 map.put(sohum, 5);
15
16 sohum.name = "Sohum";
17 TA shubha = new TA("Shubha", 24);
18 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.

```

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 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.