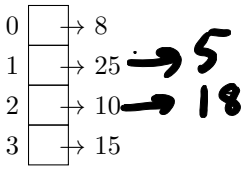
CS 61BL Hashing

Summer 2021 Recurring Section 9: Thursday July 29, 2021

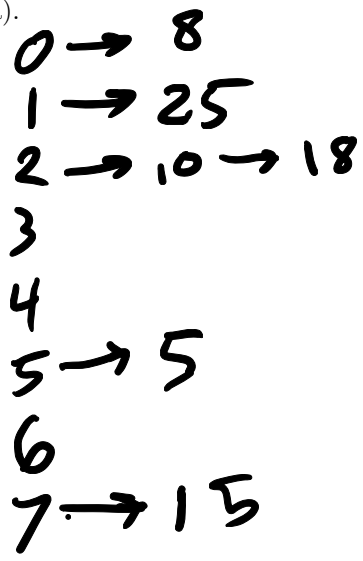
1 External Chaining

Consider the following External Chaining Hash Set below, which doubles in size when the load factor reaches 1.5. Assume that we're using the default hashCode for integers, which simply returns the integer itself.



(a) Draw the External Chaining Hash Set that results if we insert 18.

(b) Draw the External Chaining Hash Set that results if we insert 5 after the insertion done in part (a).



mod 4 ← len

mod 8

2 Invalid Hashes

For both parts below, suppose we are trying to hash the following class:

```
import java.util.Random;
class Point {
    private int x;
    private int y;
    private static count = 0;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
        count += 1;
    }
}
```

(a) Which of the hashCodes are invalid?

(i)
```
public void hashCode() {
    System.out.print(this.x + this.y);
}
```
int ← void — invalid

(ii)
```
public int hashCode() {
    Random randomGenerator = new Random();
    return randomGenerator.nextInt(Int);
}
```
invalid — not deterministic

(iii)
```
public int hashCode() {
    return this.x + this.y;
}
```
valid

(iv)
```
public int hashCode() {
    return 4;
}
```
valid, bad runtime

(v)
```
public int hashCode() {
    return count;
}
```
invalid, static

(b) *Extra:* Suppose we know all the `Points` have `x` and `y` coordinates between 0 and 10, inclusive. Suggest a **good** `hashCode` method.

```
public int hashCode() {
    return this.x * 11 + this.y;
}
```
→ unique hash code

3 Hashing Gone Crazy

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

```
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:

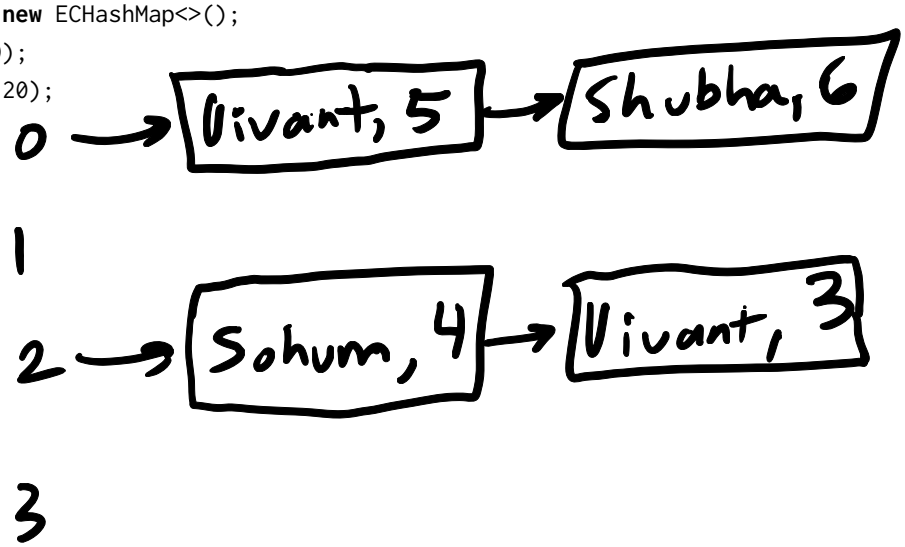
```
ECHashMap<TA, Integer> map = new ECHashMap<>();
TA sohum = new TA("Sohum", 10);
TA vivant = new TA("Vivant", 20);
map.put(sohum, 1);
map.put(vivant, 2);

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);
```



TA
Sohum
12

TA
Vivant
22

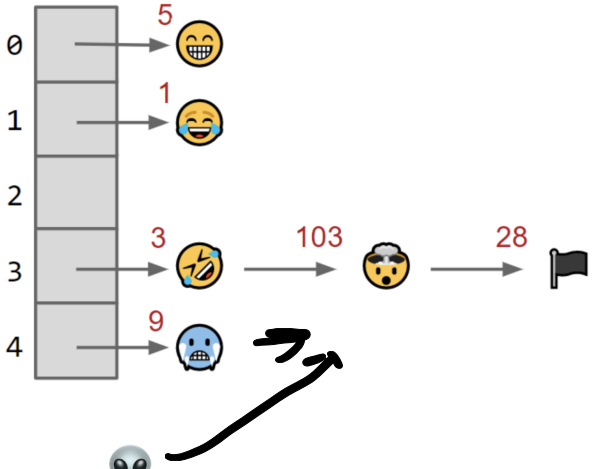
TA
Shubha
24

4. Hashing. (195 points)

a) Those are the facts (40 Points). Throughout this problem, assume we're using a hashtable (as seen in lecture) to represent a set. Suppose that each bucket of the hashtable is stored as a left leaning red black tree, and we are inserting items that implement the `Comparable` interface. Which of the following statements are true about such a hashtable?

1) **(10 points)** The runtime of contains is O(N).	◯ True ◯ False
2) **(10 points)** The runtime of contains is O(log N).	◯ True ◯ False
3) **(10 points)** The runtime of contains is O(1).	◯ True ◯ False
4) **(15 points)** One advantage of using an LLRB for the buckets is that it makes it possible to efficiently iterate over all of the keys in the set in ascending order.	◯ True ◯ False
5) **(15 points)** Assuming items are nicely spread out in the hash table, we expect that an LLRB bucket would yield significantly better performance for contains and add than if we used an ArrayList for each bucket.	◯ True ◯ False

b) Adding (120 points). Suppose now that we build a `Set<Picture>` using a hashtable, where we represent each bucket with a linked list. Suppose we've added the Picture objects below with the given hashcodes in red:



1) **(40 points)** We add a new picture 👽 with hashcode -6. In which bucket will 👽 end up in the hashtable? Assume that the hashtable does not resize.

◯ 0 ◯ 1 ◯ 2 ◯ 3 ◯ 4

2) **(40 points)** If we resize our hashtable by doubling its size, items with which hashcode will end up in a **different** bucket number than before the resize operation? Assume we're starting from the original figure above without the 👽.

☒ 5 ☐ 1 ☐ 3 ☐ 103 ☒ 28 ☒ 9

3) **(40 points)** Suppose that we perform the following actions on the original hashtable with 5 buckets illustrated in the image above:

1. `Picture x =` 
2. `hashTable.add(x); // as above, x's hashCode is 9!`
3. `x.turnPink(); // modifies x`
4. `System.out.println(hashTable.contains(x));`

Assume that the `turnPink` method changes some of x's pixels pink and adds a 3rd eye so that it looks like  . This change to the object may result in its hashcode changing.

For which of the following hashcodes will line 4 of the above code print out `true`? Note that a `Picture` object's `equals` method returns true only if their pixel values are exactly identical.

☒ `x.hashCode()` is -1 ☐ `x.hashCode()` is 0 ☒ `x.hashCode()` is 4 ☒ `x.hashCode()` is 14

☐ None of these

5. By the Numbers (370 Points).