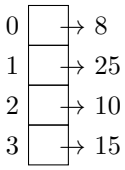
CS 61BL Hashing

Summer 2021 Recurring Section 9: Thursday July 29, 2021

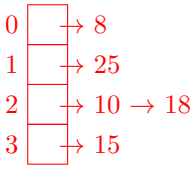
1 External Chaining

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.



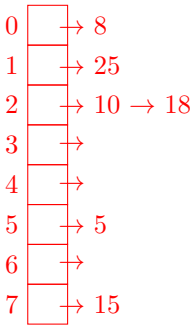
(a) Draw the External Chaining Hash Set that results if we insert 18.

Solution:



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

Solution:



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);
}
```

Solution: Invalid. The return type of the hashCode function should be an integer.

(ii)
```
public int hashCode() {
    Random randomGenerator = new Random();
    return randomGenerator.nextInt(Int);
}
```

Solution: Invalid. Using a random generator results in the hashcode function to be not deterministic. There is a possibility that the hashcode for the same Point object could be different across two attempts to hash the object.

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

Solution: Valid. This is a safe hashcode function because the fields this.x and this.y are private variables. However, if these fields were not private, then any changes made the variables would result in an entirely different hashcode function. For instance, assume that the variables x and y are public. First, hash a Point object and then change its variables (x and y). Now, it will be impossible to find the initial Point object because the hashcode function will not return the same hash value.

(iv)
```
public int hashCode() {
    return 4;
}
```

Solution: Valid. However, this is a bad hashcode function because all Point objects will be put in to the same bucket.

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

Solution: Invalid. Since count is a static variable (it is shared by all instances of Point), the same object will not return the same hash code if another Point instance is created between calls.

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

Solution:

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

With the `hashCode` above, observe that no two points will ever hash to the same bucket. Prove this to yourself!

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);
```

Solution: