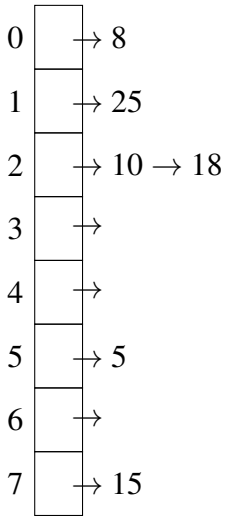
CS 61B Discussion 9 Exam Prep Spring 2017

1 Warmup (Spring 2015 MT2: 1c)

Draw the External Chaining Hash Set that results if we insert 5. As part of this insertion, you should also resize from 4 buckets to 8 (in other words, the implementer of this data structure seems to be resizing when the load factor reaches 1.5). Assume that we're using the default hashCode for integers, which simply returns the integer itself.



2 Hashtable Runtimes (Fall 2016 MT2: Q3)

Consider a hash table that uses external chaining and also keeps track of the number of keys that it contains. It stores each key at most once; adding a key a second time has no effect. It takes the steps necessary to ensure that the number of keys is always less than or equal to twice the number of buckets (i.e., that the load factor is ≤ 2). Assume that its hash function and comparison of keys take constant time. All bounds should be a function of N, the number of elements in the table.

1. Give $\Theta()$ bounds on the worst-case times of adding an element to the table when the load factor is 1 and when it is exactly 2 before the addition.

   ```
   Bound for load factor 1: Θ(N)
   Bound for load factor 2: Θ(N)
   ```

2. Assume that the hashing function is so good that it always evenly distributes keys among buckets. What now are the bounds on the worst-case time of adding an element?

   ```
   Bound for load factor 1: Θ(1)
   Bound for load factor 2: Θ(N)
   ```

3. Making no assumption about the goodness of the hashing function, suppose that instead of using linked lists for the buckets, we use some kind of binary search tree that somehow keeps itself "bushy." What bound can you place on the worst-case time for testing to see if an item is in the table?

   Bound: $\Theta(\lg N)$

4. Using the same representation as in part (c), but with a very good hash function, as in part (b), what bound can you place on the worst-case time for testing to see if an item is in the table?

   Bound: $\Theta(1)$

3 Zubat (Summer 2016 MT2: Q3)

Consider the following classes and their hashcodes and equality definitions. There is a problem with each hashCode() method below (correctness, distribution, efficiency). Provide a one sentence explanation. Do not list more than one problem. Assume there are no problems with the correctness of equals; any code for handling casting is omitted for space.

1. DynamicString

```
1  class DynamicString {
2      ArrayList<Character> vals;
3      public int hashCode() {
4          int h = 0;
5          for (int i = 0; i < vals.size(); i++) {
6              h = 31 * h + vals.get(i);
7          }
8          return h;
9      }
10
11     public boolean equals(Object o) {
12         DynamicString d = (DynamicString) o;
13         return vals.equals(d.vals);
14     }
15 }
```

   Problem(s), if any: **No problems.** (This one is done for you as an example using Java's String::hashCode.)

2. PokeTime

```
1  class PokeTime {
2      int startTime;
3      int duration;
4
5      public int getCurrentTime() {
```

```
        // Gets the current system clock time
    }

    public int hashCode() {
        return 1021 * (startTime + 1021
                * duration + getCurrentTime());
    }

    public boolean equals(Object o) {
        PokeTime p = (PokeTime) o;
        return p.startTime == startTime
                && p.duration == duration;
    }
}
```

Problem(s), if any:

Incorrect: The hashCode is non-deterministic.

3. Phonebook

```
class Phonebook {
    List<Human> humans;

    public int hashCode() {
        int h = 0;
        for (Human human : humans) {
            // Assume Human hashcode is correct
            h = (h + human.hashCode()) % 509;
        }
        return h;
    }

    public boolean equals(Object o) {
        Phonebook p = (Phonebook) o;
        return p.humans.equals(humans);
    }
}
```

Problem(s), if any:

Poor distribution: The hashCode only distributes numbers between -508 and 508, which is an inefficient use of the full integer range and will cause more collisions than necessary.

4. Person

```
class Person {
    Long id;
    String name;
    Integer age;

```

```
    public int hashCode() {
        return id.hashCode() + name.hashCode()
               + age.hashCode();
    }

    public boolean equals(Object o) {
        Person p = (Person) o;
        return p.id == id;
    }
}
```

Problem(s), if any:

Incorrect: Persons that are equals() do not necessarily have the same hashCode().

5. DblCharSeq

```
class DblCharSeq {
    char[] seq1;
    char[] seq2;

    public int hashCode() {
        int h = 0;
        for (char c1 : seq1) {
            for (char c2 : seq2) {
                h = 31 * (31 * h + c1) + c2;
            }
        }
        return h;
    }

    public boolean equals(Object o) {
        DblCharSeq d = (DblCharSeq) o;
        return Arrays.equals(seq1, d.seq1)
               && Arrays.equals(seq2, d.seq2);
    }
}
```

Problem(s), if any:

Inefficient: The hashCode takes quadratic time to compute when it could have taken only linear time.