Hash Tables (finish)
Review

CS206

March 9

Open Addressing
Probing

- Store only <K,V> at each location in array
- If key is different and location is in use then go to next
  - repeat until free location found

Quadratic Probing

- Show the final contents of the hashtable using quadratic probing assuming
  - table size is 13
  - h(t) = t % 13
- Data: <0,a> <32,b> <39,c> <12,d> <14,e> <35,f> <27,g> <13,h> <15,i> <5,j> <12,k> <13,l> <4,m> <0,n> <35,o>
- Recall quadratic probing
  - first go to h(x)
    - next to h(x)+1
    - next to h(x)+4 …
- What is the most number of steps you needed to take to find a free location?
  - during put?
  - Given the current table, contains key

Hash Value	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Growing Probe Hashtables

- O(1) get and push when lightly loaded so want to keep the table lightly loaded.
- Need to add a private “Grow” function to put
  - Grow:
    - make a new array bigger than old array (2x)
    - copy each item from old array into new array (into the correct location)
    - forget old array

Growing Hashtables

```
public class ProbeHTInc<K, V> implements Map151Interface<K, V> {
  private Pair<K, V>[] backingArray;
  private int hash(K key) {
    return Math.abs(key.hashCode()) % backingArray.length;
  }
private void grow() {
    // write me
}
```

Java

- Classes and Inheritance
    - Overloading
        - method with same name but different parameters
            - equals(Object ob) vs equals(String st)
    - Overriding of methods
        - same name, same args but in extending class
        - marked by @Override
- Exceptions **Chapter:Interlude 2,3**
- UML and Java Interfaces **Chapter: Prelude**
- Generics **Chapter Interlude 1,8**
- Inner classes

Data Structures

- Arrays
- Bags **Chapter 1,2**
- ArrayList **Chapter 10**
- Maps **Chapter 20,21**
  - key-value pairs
- Hashtables **Chapter 22,23**

Theory

- Complexity Analysis — Big-O — **Chapter 4**
  - drop constants
  - focus on dominant term
  - always look at worst case
  - Look for loops
    - loops incrementing using + or -: O(n)
    - loops incrementing using * or /: O(lg n)
    - loops inside loops (inside loops): multiply
    - loops next to loops: add
- Modularity, Abstraction and **Encapsulation** —
  - **Chapter: Prelude**