CSCI 136
Data Structures & Advanced Programming

Jeannie Albrecht

Lecture 35

May 14, 2014

Administrative Details

- Final exam - self scheduled
  - You get 2.5 hours to complete it
  - Covers everything, with strong emphasis on Ch 14-16 (BSTs, HashTables, Maps, Graphs)
  - Study guide on handouts page
- Extra credit accepted through Tue, May 20 at 5pm
- You'll get midterms and Lab 9 (Darwin) back in a little while

Last Time

- Discussed graph traversal algorithms (Ch 16)
  - Depth first search
  - Breadth first search
  - Cycle detection
  - Shortest path (Dijkstra's algorithm)
- Any questions?

Today's Outline

- Continue learning about hash tables (finally)
  - You should also read Ch 15 for more info

Map Interface

- Methods for Map<K, V>
  - int size() - returns number of entries in map
  - boolean isEmpty() - true iff there are no entries
  - boolean containsKey(K key) - true iff key exists in map
  - boolean containsValue(V val) - true iff val exists at least once in map
  - V get(K key) - get value associated with key
  - V put(K key, V val) - insert mapping from key to val, returns value replaced (old value) or null
  - V remove(K key) - remove mapping from key to val
  - void clear() - remove all entries from map

Map Interface

- Other methods for Map<K,V>:
  - void putAll(Map<K,V> other) - puts all key-value pairs from Map other in map
  - Set<K> keySet() - return set of keys in map
  - Set<Association<K,V>> entrySet() - return set of key-value pairs from map
  - Structure<V> valueSet() - return set of values
  - boolean equals() - used to compare two maps
  - int hashCode() - returns hash code associated with map (stay tuned…)

Dictionary.java

```
public class Dictionary {

    public static void main(String args[]) {
         Map<String, String> dict = new Hashtable<String, String>();
         …
         dict.put(word, def);
         …
         System.out.println("Def: "+dict.get(word));
     }
}
```

Simple Map Implementation

- A simple implementation of the Map interface is the MapList class
- Uses a SinglyLinkedList of Associations as underlying data structure
- How would we implement put(K key, V val)?

MapList.java

```
public class MapList<K, V> implements Map<K, V>{

    //instance variable
    SinglyLinkedList<Association<K,V>> data;

    public V put (K key, V value) {
       Association<K,V> temp = new Association<K, V> (key, value);
       Association<K,V> result = data.remove(temp);

       data.addFirst(temp);
       if (result == null) return null;
       else return result.getValue();
    }
}
```

Simple Map Implementation

- A simple implementation of the Map interface is the MapList class
- Uses a SinglyLinkedList of Associations as underlying data structure
- How would we implement put(K key, V val)?
- What is the running time of:
  - containsKey(K key)?
  - containsValue(V val)?
- Bottom line: not O(1)!

Search/Locate Revisited

- How long does it take to search for objects in Vectors and Lists?
  - O(n) on average
- How about in BSTs?
  - O(log n)
- Can this be improved?
  - With hash tables, YES!
  - Can locate objects in roughly O(1) time!

Hashing in a Nutshell

- Group objects into "bins"
- When searching for object, go directly to appropriate bin
- If there are multiple objects in bin, then search (linearly) for correct one
- Important Insight: This works best when objects are evenly distributed among bins

Implementing a HashTable

- How can we represent bins?
- Slots in array (or Vector, but arrays are faster)
  - Initial size of array is a fixed-length prime number
- How do we find a bin number?
  - We use a *hash function* that converts keys into integers
  - In Java, we can use the hashCode() method that all Objects have

Implementing HashTable

- How do we add Associations to the array?
  - Can get complicated if collisions occur
- Two approaches
  - Open addressing (using linear probing)
  - External chaining

Linear Probing

- If a collision occurs at a given bin, just move forward (linearly) until an empty slot is available
  - Specify trivial hash function
  - Initial array size = 8
  - Add "algorithm" to hash table
  - Add "data"
  - Add "queue"
- Let's implement put(key, val) and get(key)…
- What happens when we remove "algorithm", and then lookup "queue"?
  - Need a "placeholder" for removed values…

Linear Probing

- Runtime
  - put
    - O(1)
  - get
    - O(1)
  - remove
    - O(1)