CSCI 136
Data Structures &
Advanced Programming

Jeannie Albrecht
Lecture 36
May 16, 2014

Administrative Details

- Final exam - self scheduled
  - You get 2.5 hours to complete it
  - Covers everything, w/ strong emphasis on Ch 14-16
    - BSTs, HashTables, Maps, Graphs
- NOTE: I will be out of town this weekend
  - Otherwise I'll be mostly around next week

Last Time

- Wrapped up graphs
- Started discussing Maps

Today's Outline

- Finish discussing HashMaps/HashTables

Recap: 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
- Must deal with collisions

Recap: Linear Probing

- If a collision occurs at a given bin, just move forward (linearly) until an empty slot is available
- Need a "placeholder" for removed values…

```
//OVERSIMPLIFIED VERSION OF PUT
public void put (K key, V val) {
    int index = key.hashCode() % arraySize;
    while (index < array.length && array[index]!=null &&
           !array[index].getKey().equals(key)) {
           index++;
    }
    array[index] = new Association(key, val);
}
```

Recap: Linear Probing

- If a collision occurs at a given bin, just move forward (linearly) until an empty slot is available
- Need a "placeholder" for removed values…

```
//OVERSIMPLIFIED VERSION OF GET
public V get (K key) {
    int index = key.hashCode() % arraySize;
    while (index < array.length && array[index]!=null &&
           !array[index].getKey().equals(key)) {
           index++;
    }
    if (index==array.length) return null;
    return array[index];
}
```

Linear Probing

- Runtime
  - put, get, remove
    - O(1)
  - …as long as the array isn't too full!

External Chaining

- Downsides of linear probing
  - What if array is almost full?
  - Linear probing is extremely difficult on almost-full arrays
- How can we avoid this problem?
  - Keep all values that hash to same bin in a "collection"
    - Usually a SLL
  - External chaining "chains" objects with the same hash value together

Example

- Recall our previous example
  - Add algorithm
  - Add data
  - Add queue
  - Now remove algorithm
  - No need for placeholders!

External Chaining

- Let's look at put(key, val)…

```
public V put(K key, V val) {
   //locate returns SLL in the appropriate bin
   SLL list = locate(key);
   ComparableAssociation newA =
             new ComparableAssociation(key, val);
   ComparableAssociation oldA = list.remove(newA);
   list.add(newA);
   return oldA.getValue();
}
```

External Chaining

- Let's look at put(key, val)…
- Runtime
  - put, get, and remove
    - O(ℓ) (ℓ is size of linked list at given bin)
      – As long as table isn't too full, this is almost O(1)

Load Factor

- Need to keep track of how full the table is
  - Why?
  - What happens when array fills completely?
- Load factor is a measure of how full the hash table is
  - LF = # elements/table size
- When LF reaches some threshold, need to double size of array (typically threshold = 0.6)
  - How?

Doubling Array

- Cannot just copy values
  - Why?
  - Hash values may change (i.e., element "list")
  - Example
    - key.hashCode() % 8 = 3;
    - key.hashCode() % 16 = 11;
- Have to recompute all hash codes
- This is expensive!

Good Hashing Functions

- Important point:
  - All of this hinges on using "good" hash functions that spread keys "evenly"
- Two properties of good hash functions:
  - Fast to compute
  - Uniformly distribute keys
- Almost always have to test "goodness" empirically

Example Hash Functions

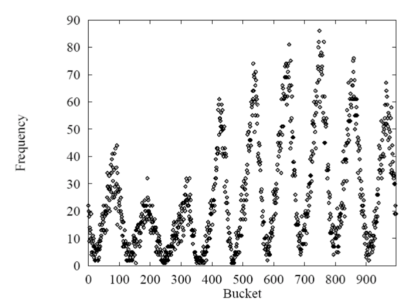
- What are some feasible hash functions for Strings?
  - First char ASCII value mapping
    - 0-255 only
    - Not uniform (some letters more popular than others)
  - Sum of ASCII characters
    - Not uniform - lots of small words
    - smile, limes, miles, slime are all the same

Example Hash Functions

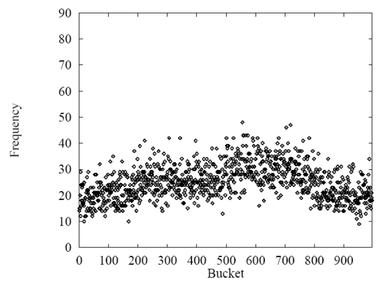
- String hash functions
  - Weighted sum
    - Small words get bigger codes
    - Distributes keys better than non-weighted sum
  - Let's look at different weights…

$$\sum_{i=0}^{n=s.length()} s.charAt(i)$$

Hash of all words in UNIX spelling dictionary (997 buckets)

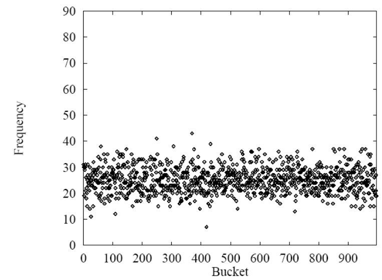


$\sum_{i=0}^{n}$ s.charAt(i) * 2^i



$\sum_{i=0}^{n}$ s.charAt(i) * 256^i

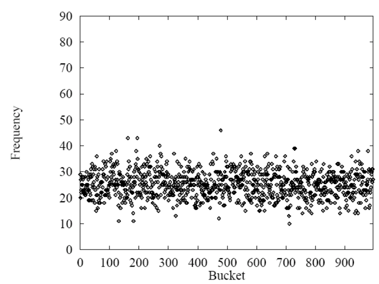
This looks pretty good, but 256^i is big…



$\sum_{i=0}^{n}$ s.charAt(i) * 31^i

Java uses:

$$\sum_{i=0}^{n} \text{s.charAt}(i) * 31^{(n-i-1)}$$



Dictionary Summary

	put	get	space
unsorted vector	O(n)	O(n)	O(n)
unsorted list	O(n)	O(n)	O(n)
ordered vector	O(n)	O(log n)	O(n)
balanced BST	O(log n)	O(log n)	O(n)
array indexed by key	O(1)	O(1)	O(key range)

Course Evals – Wrap Up

- This semester we have explored structures to represent data
  - Started with simple arrays
  - Ended with complicated graphs and hash tables
- We studied algorithms to manipulate and use these data structures
- We studied ways to evaluate and visualize them
- Emphasized importance of abstraction, modular design, correctness, and efficiency

Topics Covered

- Vectors (and arrays)
- Complexity (big O)
- Recursion + Induction
- Searching
- Sorting
- LinkedLists
- Stacks
- Queues
- Iterators
- Comparables/Comparators
- OrderedStructures
- Binary Trees
- Priority Queues
- Heaps
- Binary Search Trees
- Graphs
- Maps/Hashtables

What's Next?

- Data structures provides you with a complete tool box to study the rest of Computer Science
  - Programming Languages
  - Artificial Intelligence
  - Compilers
  - Networks
  - Distributed Systems
  - Algorithms
  - Operating Systems
  - …
- Also applicable to many areas outside of CS

Thanks!!

- Any questions?
- Thanks for a great semester!
- Have a great summer!
- Congrats seniors!