Class Exercise

- have a program read Hamlet
- build a tree with, at each node
  - a word
  - a count of how often it appears
- allow search for words returning
  - 0 if the word is not in Hamlet
  - k if the word appears k times in Hamlet
Approach 1

- Vector of StringCountPair
  - when inserting a string
    - either update an existing entry, or
    - add a new entry to the vector if none
  - when searching
    - if string is in tree
      - search the Vector to get the count

- big O analysis?
Approach 2

- Read java documentation
  - find out about interface Map<K,V>
- Use a Map instead of Vector<StringCountPair>
  - different people use different implementations of Map
- big O?
  - glean from docs, or experiment
- Compare with others in the class
Approach 3

- Design a tree that has a count field at every node. **Use inheritance!!!**
  - interface ACTree
    - seek: like search but returns number
    - cinsert: like insert but updates count
    - (might also need draw, height, countNodes)
  - ECtree extends ETree implements ACTree
  - CTree extends Tree implements ACTree
Comparisons

- What was the big O behavior?
- Compare with others
- Conclusions?
More on Maps

- Chapter 15 of Bailey
Hashing

- a cruel joke
- perpetrated on victims
- of algorithms and data structures courses!
Hashing is crude

- function from \( T \) to fixed-range of int
  - Easy example: \( T \) is String
    - add positions of characters in alphabet
    - form the result mod 100
    - function from String \( \rightarrow [0, 99] \)
    - \( h(\text{csci}) = 2 + 18 + 2 + 8 = 30 \)
    - \( h(\text{supercalifragilistic...}) = \text{big \% 100} \)
  - Declare array[0..99]
  - Store str at array[h(str)]
So what can go wrong?

- Store `car` at 2+0+17. `array[19]`
- Store `dem` at 3+4+12. `array[19]`
- Oops!
- hash clash
- what to do?
Rehashing (open addressing)

- have an algorithm to calculate a new address for storing the data
- repeat as necessary
Chaining

- instead of storing data at array[i]
- store the head of a linked list of data
Analysis

- If there are no hash clashes
  - (extremely unlikely, but what can you do to improve the chances?)
- insert and search are both $O(1)$!!!
If

- there are not too many rehashes
  - (what can you do to make them less likely?)
- insert and search could still be O(1)
- especially when aggregated and amortized
If

- the lengths of the linked lists are short
  - (what can you do to try to arrange that?)
- insert and search could still be O(1)
- especially when aggregated and amortized
<table>
<thead>
<tr>
<th>structure</th>
<th>insert</th>
<th>delete</th>
<th>search</th>
</tr>
</thead>
<tbody>
<tr>
<td>linked list</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>unsorted array</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>sorted array</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>hashing best</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>hashing worst</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>hashing expected</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>red-black tree worst</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
The cruel joke:

<table>
<thead>
<tr>
<th>structure</th>
<th>insert</th>
<th>delete</th>
<th>search</th>
</tr>
</thead>
<tbody>
<tr>
<td>linked list</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>unsorted array</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>sorted array</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>BST worst</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>BST expected</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>red-black tree worst</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>hashing expected</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>