Lecture 6: ArrayList Implementation & Complexity

CS 62
Spring 2018
Alexandra Papoutsaki & William Devanny
Lab

- Timing `ArrayList` operations
- Encourage working in pairs
- `Stopwatch` class: `start()`, `stop()`, `getTime()`, `reset()`
- Java has Just-In-Time compiler
- Must “warm-up” before you get accurate timing
  - What can mess up timing?
- Uses `Vector` from Bailey rather than `ArrayList` from Java libraries because can change way it increases in size.
Programming Assignment

• Weak AI/Natural Language Processing:
• Generate text by building frequency lists based on pairs of words. `ArrayList` of `Associations` of `String` (words) and `Integer` (count of that word)
ArrayList

- Not using Bailey implementation
  - see code on-line for implementation by Tamassia & Goodrich
- Standard Java libraries have lots of extra methods not in our implementation
- Many involve working on other collections
  - irrelevant for us at this point.
  - `addAll`, `clear`, `contains`, `containsAll`, `listIterator`, `removeAll`, `replaceAll`, `retainAll`, `sort`, `spliterator`, `sublist`, `toArray`
Tamassia & Goodrich ArrayList

- Interface is **IndexList**<E>
- See **ArrayIndexList**<E>
  - Similar to ArrayList
  - Instance variables:
    - **elts**: array instance variable
    - **eltsFilled**: number of slots filled.
- Creating new **ArrayList** is weird
  - Can’t construct array of variable type!
  - Create array of **Object**, but coerce to believe array of **E**
ArrayList Implementation

• Some operations very cheap:
  • size, isEmpty, get, set take constant time (no search)
  • Others more expensive
Adding Elts in Slot i

• Easy if there is space:
  • At end, just add it
  • If before end, must move all elements at i and beyond to right before inserting
  • *Delete similar*

• What if we run out of space?
  • Create new array twice as big and copy old elements over before adding.
  • How expensive is this?
Order of Magnitude

- **Definition:** We say that $g(n)$ is $O(f(n))$ if there exist two constants $C$ and $k$ such that $|g(n)| \leq C |f(n)|$, for all $n > k$.

- Used to measure time and space complexity of algorithms on data structures of size $n$.

- **Examples:**
  - $2n + 1$ is $O(n)$
  - $n^3 - n^2 + 83$ is $O(n^3)$
  - $2^n + n^2$ is $O(2^n)$

- **Most common are:**
  - $O(1)$ - for any constant
  - $O(\log n), O(n), O(n \log n), O(n^2), ..., O(2^n)$
Complexity

\[ f(n) = Cn \quad \quad g(n) \quad \quad f(n) = \sqrt{n} \]

\[ f(n) = n \]
Figure 5.2 Near-origin details of common curves. Compare with Figure 5.3.

Figure 5.3 Long-range trends of common curves. Compare with Figure 5.2.
Comparing Orders of Magnitude

• Suppose have ops w/complexities given & problem of size n taking time t.
• How long if increase size of problem?

<table>
<thead>
<tr>
<th>Problem Size:</th>
<th>10 n</th>
<th>100n</th>
<th>1000n</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(\log n)$</td>
<td>3 + t</td>
<td>7 + t</td>
<td>10+ t</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>10 t</td>
<td>100 t</td>
<td>1000 t</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
<td>&gt; 10 t</td>
<td>&gt; 100 t</td>
<td>&gt; 1000 t</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>100 t</td>
<td>10,000 t</td>
<td>1,000,000 t</td>
</tr>
<tr>
<td>$O(2^n)$</td>
<td>$t^{10}$</td>
<td>$t^{100}$</td>
<td>$t^{1000}$</td>
</tr>
</tbody>
</table>
Rule of thumb
Adding to ArrayList

- Suppose n elements in ArrayList and add 1.
- If space:
  - Add to end is $O(1)$
  - Add to beginning is $O(n)$
- If not space:
  - What is cost of ensureCapacity?
  - $O(n)$ because n elements in array
EnsureCapacity

• What if only increase in size by 1 each time?
  • Adding \( n \) elements one at a time to end
    • Total cost of copying over arrays: \( 1 + 2 + 3 + \cdots + (n - 1) = n(n - 1)/2 \)
    • Total cost of \( O(n^2) \)
    • Average cost of each is \( O(n) \)

• What if double in size each time?
  • Suppose add \( n = 2^m \) new \texttt{elts} to end
    • Total cost of copying over arrays: \( 1 + 2 + 4 + \cdots + n/2 = n - 1, O(n) \)
    • Average cost of \( O(1) \), but “lumpy”
ArrayList Operations

- Worst case:
  - $O(1)$: size, isEmpty, get, set
  - $O(n)$: remove, add

- Add to end is on average $O(1)$