CS136: Data Structures & Advanced Programming

Spring 2020
Williams College

Administrative Details

- Lab 2 is due Monday
    - See office hour calendar for TA support this weekend
- Colloquium Today
    - Thesis student proposals
- I will be gone next week beginning at 4pm Monday
    - Going to CA for a conference called FAST
    - I'll be checking Piazza, our anonymous (to classmates) forum for questions, but office hours will be hard to hold from CA... a good chance to introduce yourself to Dan!

Last Time

Potpourri of Topics helpful for lab

Essential `Object` methods:

- `public String toString()`
- `public boolean equals(Object other)`

`Association<KeyType, ValueType>`

`WordFreq.java` (example code)

Today

Deep dive into `Vector` class, including:

- Where to find the source code
- Important implementation details
- Design tradeoffs

We'll start to think about "efficiency"

- What is the cost of our implementation *in the worst case*?
    - We'll define performance by counting "operations"
    - We'll define space usage by counting "elements" or "slots"
    - We'll formally explore asymptotic analysis next week

Vector Implementation: Getting the Code

Structure5:

- Code associated with the textbook is publicly available
    - bailey.jar - archive used by javac
    - structure-source.tgz - compressed bundle of Java text files
        - After uncompressing, `src/structure5` has the code we want!
- Javadoc for the code is publicly available too

See "handouts" page for instructions on how to set up a Unix machine so you can use structure5 code in your own programs

Vector<E> API (select methods)

- `get(int), set(int, E)`
- `firstElement(), lastElement()`
- `contains(E), indexOf(E)`
- `add(E), addElement(E), add(int,E)`
- `remove(E)`

- `capacity()`
- `ensureCapacity()`
- `clear()`

- `toString()`

Vector Details: Storing Data

Internally, the `Vector` class stores an array of type `Object`

- The array is not necessarily filled
- We keep track of the number of current elements in the array using an explicit `elementCount` variable
  - How do we ensure that `elementCount` stays in sync with our actual count?
  - What happens if we try to add an element but the array is full?
- Overloaded constructor(s) allow us to specify an initial array size (the Vector's capacity)
  - Default capacity used if none is provided

Vector Details: get(int)/set(int, E)

Arrays use bracket notation to access and update elements at a given index

- How do I determine where to find an element given it's index?
- How expensive is it to find the offset if the array has:
  - 1 element?
  - 10 elements?
  - 100 elements?
  - 1 million elements?
- `v.get(int)` uses bracket notation to access `elementData[i]`
- `v.set(int, E)` uses bracket notation to update `elementData[i]`

Get/set cost is the same as the cost of accessing/updating an array.

Vector Details: add(E)

Arrays don't have any notion of "appending"

- What does it mean to “append” to a Vector?

When we think about performance, we often care most about the "worst case"

- What are the "worst cases" that we need to consider when appending to a Vector?
- How expensive are these worst cases when the Vector has:
  - 1 element?
  - 100 elements?
  - *n* elements?

Vector Details: add(int, E)

Arrays don't have any notion of "inserting"

- What does it mean to insert into the middle of a Vector?

When we think about performance, we often care most about the "worst case"

- What are the "worst cases" that we need to consider when inserting an element into a Vector?
- How expensive are these worst cases when the Vector has:
    - 1 element?
    - 10 elements?
    - *n* elements?

Vector Details: contains(int, E)

contains(E) determines if a value appears in the Vector

- What does it mean for a value to "appear in" a Vector?
  - elementData[i].equals(obj) == true   (for some index i)

- What are the "worst cases" that we need to consider when searching for an element in a Vector?
- How expensive are these worst cases when the Vector has:
  - 1 element?
  - 100 elements?
  - *n* elements?

Example: v.contians(new Associaton(...))

The Association class defines the equality of a and b as:

```
        return a.getKey().equals(b.getKey());
```

How would I search a `Vector<Association<String, Integer>` for the wordcount associated with the String "forefathers"?

- I'd probably use a for loop, comparing each element's key against the String "forefathers"

Are there any Vector methods I could use instead? How?

Thinking beyond the Vector API

Consider a Vector of student GitHub IDs. I want to make sure everyone's IDs are included *exactly once*. This means I need to check if there are any duplicates.

How would I check for duplicates?

- How expensive is this (in the worst case) when the Vector has:
  - 2 element?
  - 200 elements?
  - *n* elements?

Lab 2: WordGen

Think about the roles of each class:

- `FrequencyList` counts letters, samples letters
  - Nothing else!
- `Table` maps `Strings` to `FrequencyLists`. Your table should let you:
  - add an observation that a letter follows a particular string
  - yield a new letter that is randomly chosen from the distribution of letters that follow a particular string
- `WordGen` should parse your text, and then generate a new text
- Keep the state of each class limited to the minimum it needs to do its task
  - Abstraction is the key to writing clean, testable, and debuggable code!