CS310
Data Structures

K. Raven Russell

krusselc@gmu.edu

George Mason University

Today

- **Last Lecture**
  - Dynamic Array Lists
- **Today**
  - Dynamic Array List Analysis
  - Review: Iterators
  - (Time Permitting) Side Track: Streams

Questions?

Dynamic – Changes size when more space is needed

Array – What’s under the hood (vs. “linked” list)

List – The human idea

Dynamic Array List Analysis

Data Structure Operations Big-O

- Limitations of these structures?

Operation Implementation	get set	add remove end	insert remove begin	insert remove middle	search	grow?
Array	1	-	-	-	-	no
List (Static Array)	?	?	?	?	?	no
List (Dynamic Array)	?	?	?	?	?	Yes

Data Structure Operations Big-O

– Limitations of these structures?

Operation Implementation	get set	add remove end	insert remove begin	insert remove middle	search	grow?
Array	1	-	-	-	-	no
List (Static Array)	1	1	n	n	n	no
List (Dynamic Array)	1	?	n	n	n	Yes

Data Structure Operations Big-O

- Limitations of these structures?

Operation Implementation	get set	add remove end	insert remove begin	insert remove middle	search	grow?
Array	1	-	-	-	-	no
List (Static Array)	1	1	n	n	n	no
List (Dynamic Array)	1	**n?**	n	n	n	Yes

O(1) → O(n) not good...

- We've lost O(1) for adding items to an array
- Would it just be better to allocate huge arrays?
  - What's the initial capacity of the array?
- Wait... we are only occasionally expanding the array...

Whiteboard

Let’s Literally Count the Work

Amortized Analysis

- Looks at the time to perform a sequence of operations averaged over the number of operations: T(n)/n
- Shows that the average cost over time isn't as bad as the worse case for a single operation
- Not the same as average case analysis!
  - Average Case: the expected cost of each operation (probabilistic)
  - Amortized: the average cost of each operation in the worst case (no probability involved!)

Aggregate Method: Dynamic Arrays

- If we always double the array...
- let’s say c_i is the cost of the i-th call
- If i-1 is an exact power of 2, we need to expand and $c_i = i$
- Otherwise $c_i = 1$

$$
\begin{aligned}
\sum_{i=1}^{n} c_i &\le \underline{n} + \underline{\sum_{j=0}^{\lfloor \log n \rfloor} 2^j} \\
&< n + 2^{\lfloor \log n \rfloor + 1} \\
&= n + 2 * 2^{\lfloor \log n \rfloor} \\
&\le n + 2n \\
&= 3n
\end{aligned}
$$

Each item needs to be added at least once.

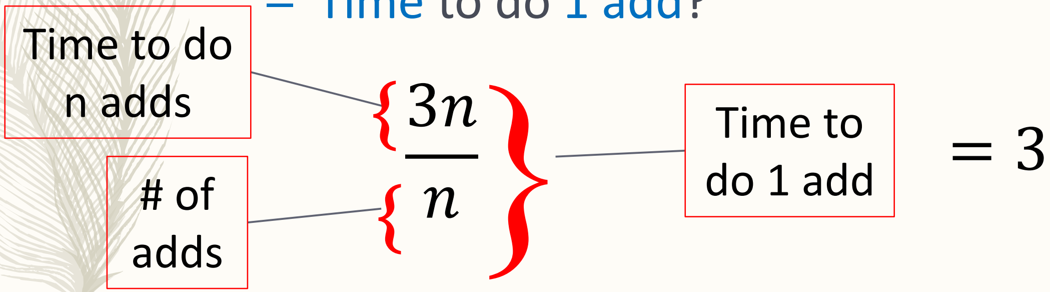
Each expansion is 2 times bigger than the last, and there are log n expansions.

Aggregate Method: Dynamic Arrays

- Time to do “n” adds…

$$
\begin{aligned}
\sum_{i=1}^{n} c_i &\le n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j \\
&< n + 2^{\lfloor \log n \rfloor + 1} \\
&= n + 2 * 2^{\lfloor \log n \rfloor} \\
&\le n + 2n \\
&= \boxed{3n}
\end{aligned}
$$

- Time to do 1 add?



Time to do n adds

# of adds

$$\left\{ \frac{3n}{n} \right\}$$

Time to do 1 add

$= 3$

Whiteboard

Seeing This With $$

Dynamic List Add: Pre-Paying

- At an intuitive level, we can just pretend that every insert operation costs 3 units of time
  - 1 to insert it into the list
  - 1 to move it later
  - 1 to move an item that was previously moved
- What's three units of time in big-O?
  - Yay!

Data Structure Operations Big-O

– Limitations of these structures?

Operation Implementation	get set	add remove end	insert remove begin	insert remove middle	search	grow?
Array	1	-	-	-	-	no
List (Static Array)	1	1	n	n	n	no
Dynamic Array	1	**O(1) Amortized**	n	n	n	Yes

Review: Iterators

Iterators

- The bookmark for data structures!
  - Give access to all the items in a collection in some unspecified order
  - Conceptually the iterator has a position between two elements
- Operations
  - Most important: next() and hasNext()
  - Optional: previous(), hasPrevious(), add(), remove()
- Finger on the structure demo...

ConcurrentModificationException

- Java doesn't try to coordinate multiple iterators.
  - Easy for reading/viewing
  - Difficult for modification

```
itr1 = list.iterator();
itr2 = list.iterator();
itr1.remove();
itr2.next(); // Error
```

- But you’d need to code this if you wanted this to happen on your own data structures…

Iterators Basics: Dynamic Arrays

What we want:

```
curr is a new iterator //initialization
while(curr.hasNext()) { //stop condition
    value = curr.next(); //get value AND increment
}
```

What this replaces:

```
int curr = 0
while(curr < size) {
    value = arr[curr++];
}
```

So the iterator needs to do this:

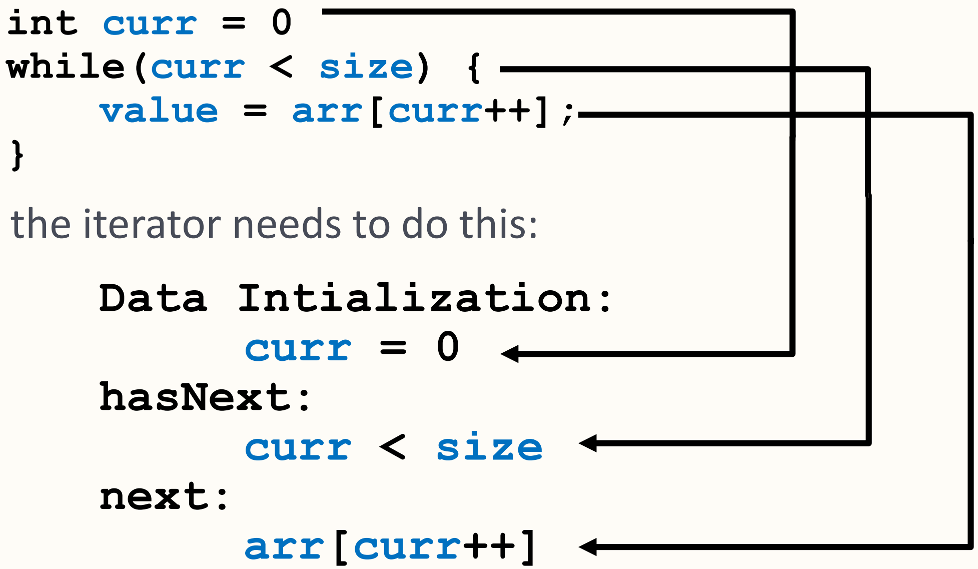
```
    data & initialization?

          __________

    when should we stop?

          __________

    how do we get the next object and move over?

          __________
```

Iterators Basics: Dynamic Arrays

What we want:

```
curr is a new iterator //initialization
while(curr.hasNext()) { //stop condition
    value = curr.next(); //get value AND increment
}
```

What this replaces:



```
int curr = 0
while(curr < size) {
    value = arr[curr++];
}
```

So the iterator needs to do this:

```
    Data Intialization:
          curr = 0
    hasNext:
          curr < size
    next:
          arr[curr++]
```

Java Iterators

- Interface Iterable<T>
  - **java.lang**
  - Iterator<T> iterator()
  - http://docs.oracle.com/javase/8/docs/api/java/lang/Iterable.html

- Interface Iterator<E>
  - **java.util**
  - boolean hasNext()
  - E next()
  - http://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html

Nested Classes - Four Types

- Static Class
- Inner Class
  - a.k.a. non-static nested class
- Local Class
  - special type of inner class
- Anonymous Class ← Today
  - special type of inner class

Anonymous Class Example

```
interface Exporter {
    public String export();
}

class MyClass {
    public Exporter getExporter() {
        return new Exporter() {
            public String export() {
                return "Export";
            }
        }; //<- very important
    }
}
```

Review: What is an interface?

Nested Class Rules – Lots of them

- Static classes
  - can't access non-static fields & methods of outer class
- Inner classes
  - can't be declared without an instance of the outer class
- Local/Anonymous classes
  - can only access local variables defined as final
- Lots more, see Java Tutorial:
  - http://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

Typical Anonymous Class Style

```
import java.util.Iterator;

class MyList<T> implements Iterable<T> {
    public Iterator<T> iterator() {
        return new Iterator<T>() {
            public boolean hasNext() { ... }
            public T next() { ... }
        };
    }
}
```

Two Ways to Use an Iterator

```
public static void main(String[] args) {
    MyList<String> list = new MyList<>();
    list.add("Alpha");
    list.add("Bravo");
    list.add("Charlie");
    list.add("Delta");

    Iterator<String> iter = list.iterator();
    while(iter.hasNext()) {
        String item = iter.next();
        System.out.println(item);
    }

    for(String item : list) {
        System.out.println(item);
    }
}
```

Other Data Structures? Same!

```java
public static void main(String[] args) {
    //if dataStruct implements Iterable<String>
    //then the following code will work!

    //Option 1: Manual Iteration
    Iterator<String> iter = dataStruct.iterator();
    while(iter.hasNext()) {
        String item = iter.next();
        System.out.println(item);
    }

    //Option 2: Enhanced for-loop
    for(String item : dataStruct) {
        System.out.println(item);
    }
}
```

Why do we need Iterable/Iterator?

- Each data structure is fundamentally different.

- Without Iterable/Iterator, clients would have to develop code tailored to each data structure in order to accomplish this.

- Using common interface, want to traverse...
  - Lists (Dynamic Arrays, Linked Lists, ...)
  - Stacks and Queues
  - Trees and Graphs
  - etc.

Design/Implementation Choices

- Many design choices for ever iterator implementation:
  - Where does it point when it's created?
  - For add/remove, where are they added/removed?
    - *remember conceptually the iterator is...*
  - Can you have multiple iterators?
  - What methods do you want to offer? previous?
    - *Think about this question again when we cover linked lists…*

- Design depends on the data structure!
  - documentation should always explain to users…

Questions?

Later in the semester…

Iterators for…

- Note: the implementations listed below are what one “usually” does, it isn’t the one any only way to do them.
- **Dynamic Arrays:**
  - Linear traversal
    - *store the index of “next”*
- **Linked Lists:**
  - Linear traversal
    - *store a reference to the node that’s “next”*
- **Stacks / Queues:**
  - Top to bottom (often) or front to back
    - *use underlying storage array/dynamic array/linked list*
- **Priority Queues:**
  - Priority order (often)
    - *use underlying storage*

Iterators for…

- Note: the implementations listed below are what one “usually” does, it isn’t the one any only way to do them.
- **Trees:**
  - Level-order traversal
    - *If tree stored as an array, store index of next*
    - *If tree stored as a linked structure, store a queue of node pointers*
  - Pre-order / Post-order / In-order traversal
    - *store a stack of node pointers*
- **Graphs:**
  - Breadth-first traversal
    - *Store a queue*
  - Depth-first traversal
    - *Store a stack*
- **Sets / Maps:**
  - Doesn’t matter, just visit each item once
    - *use underlying structure walk*

Time?

Java Streams Explained…

TANGENT: Streams

- As a programmer it is important that you understand the difference between the following types of things:
  - Character streams and Byte streams
  - Buffered and Unbuffered streams
- This is definitely introduced in CS262
- If you want to be a professional Java developer…
  - You should know when to use: Scanner vs. FileInputStream vs. FileReader vs. BufferedInputStream vs. BufferedReader vs. DataInputStream vs ObjectInputStream ...
  - This should have also been introduced in 211
  - **Go here** if you don’t know these: https://docs.oracle.com/javase/tutorial/essential/io/streams.html