CS 600.226: Data Structures

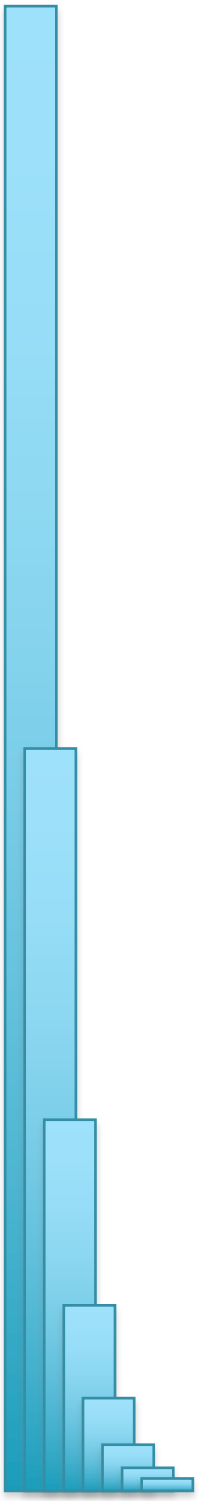
Michael Schatz

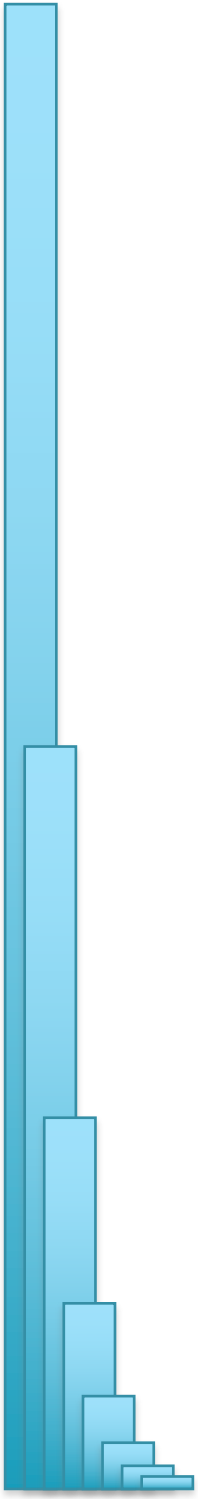
Oct 15, 2018

Lecture 20. Sets

Agenda

1. ***Recap on Graphs***
2. ***Sets***



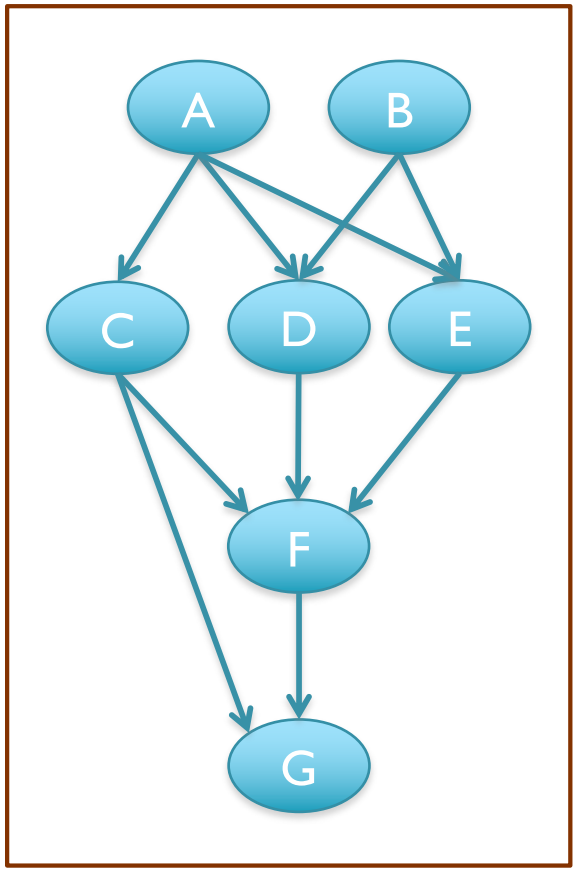


Part 1: Graphs

Graphs are Everywhere!

Computers in a network, Friends on Facebook, Roads & Cities on GoogleMaps, Webpages on Internet, Cells in your body, …

Representing Graphs



Incidence List

Good for sparse graphs

Compact storage

A: C, D, E

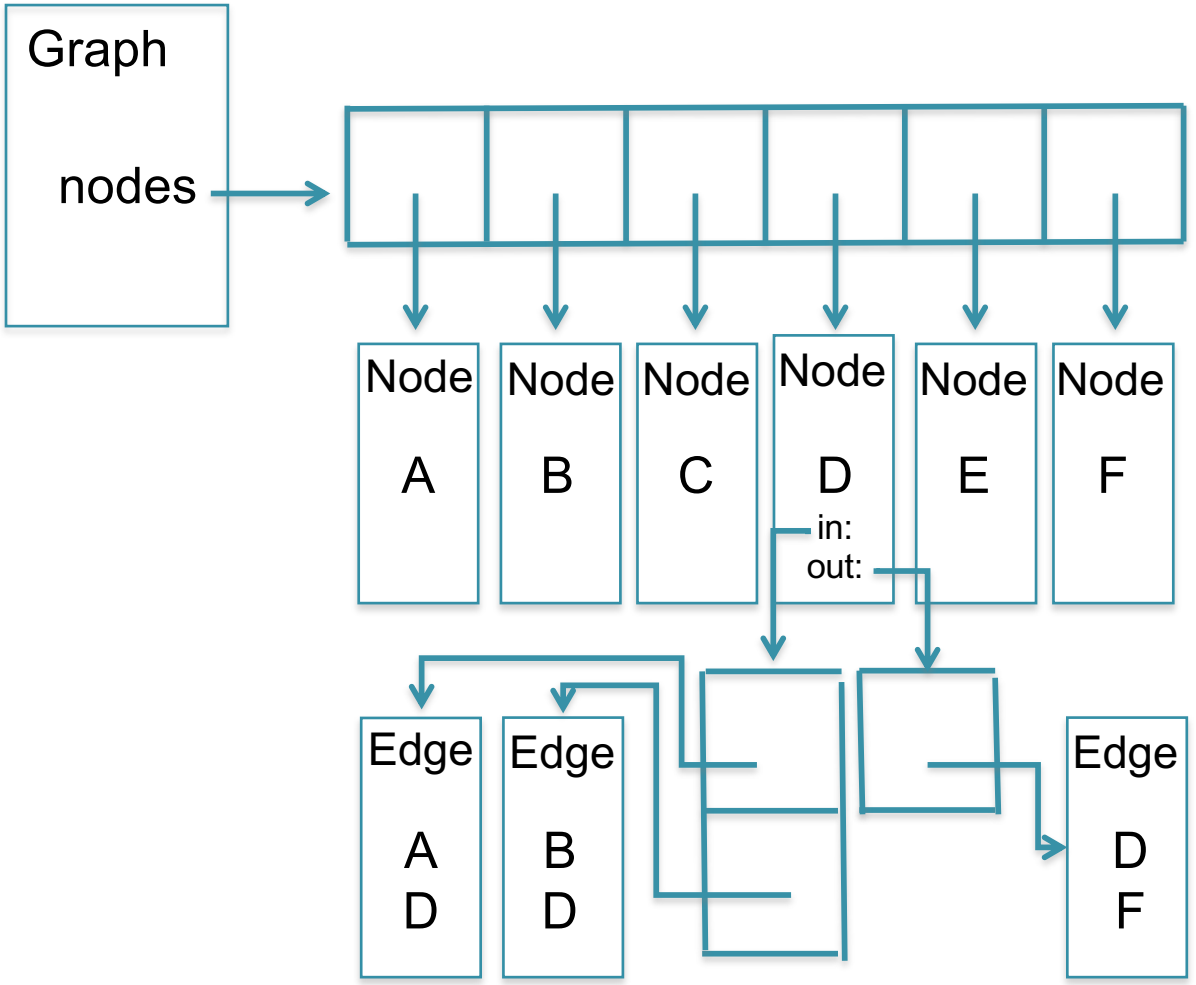
B: D, E

C: F, G

D: F

E: F

G:



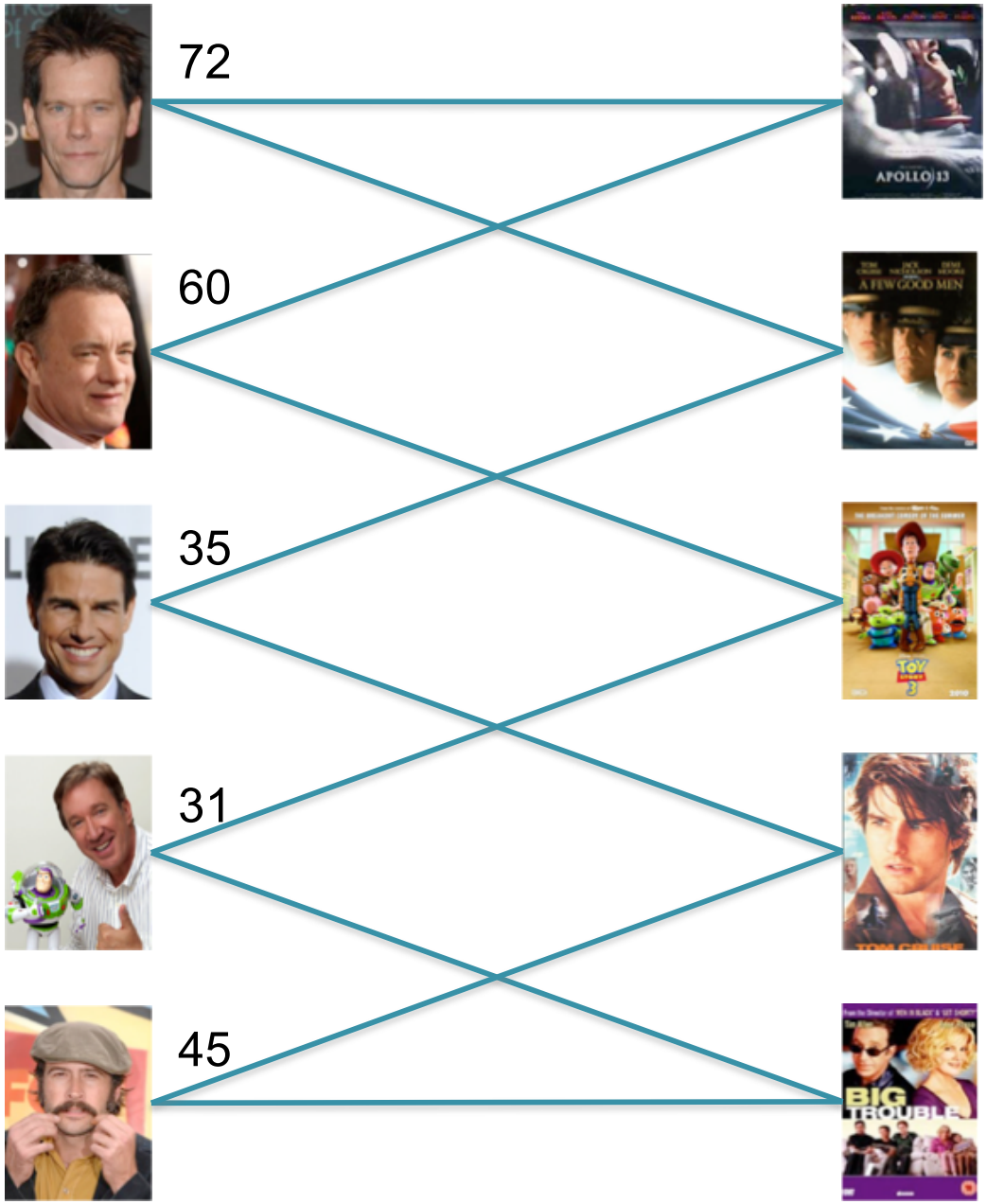
Note the labels in the edges are really references to the corresponding node objects!

Kevin Bacon and Bipartite Graphs

Find the ***shortest*** path from Kevin Bacon to Jason Lee

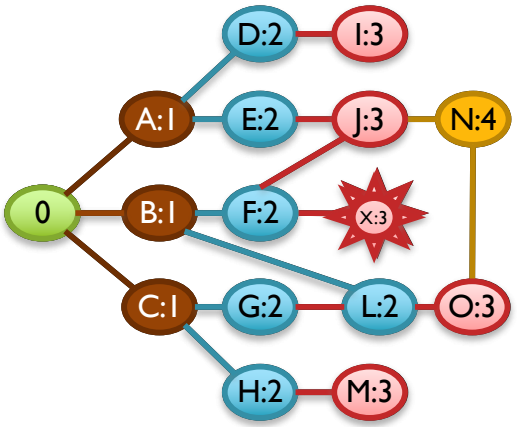
Breadth First Search: 4 hops

Bacon Distance: 2



BFS

```
BFS(start, stop)
// initialize all nodes dist = -1
start.dist = 0
list.addEnd(start)
while (!list.empty())
   cur = list.begin()
   if (cur == stop)
      print cur.dist;
   else
      foreach child in cur.children
         if (child.dist == -1)
            child.dist = cur.dist+1
            list.addEnd(child)
```



0

A,B,C
B,C,D,E
C,D,E,F,L

D,E,F,L,G,H
E,F,L,G,H,I
F,L,G,H,I,J
L,G,H,I,J,X
G,H,I,J,X,O
H,I,J,X,O

I,J,X,O,M
J,X,O,M
X,O,M,N
O,M,N
M,N

N

[How many nodes will it visit?]

[What's the running time?]

[What happens for disconnected components?]

BFS: Queue

What is the runtime complexity?

What is the space complexity?

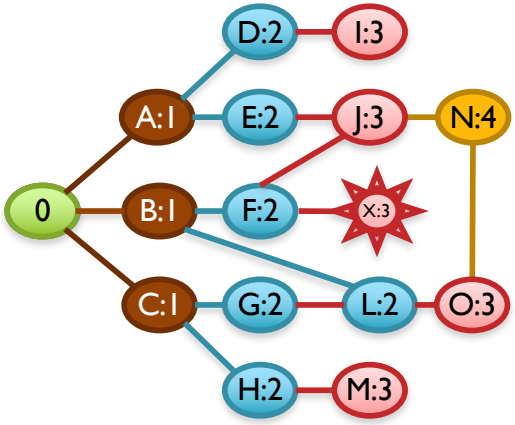
```
BF
// i
sta
list
while (!list.empty())
   cur = list.begin()
   if (cur == stop)
      print cur.dist;
   else
      foreach child in cur.children
         if (child.dist == -1)
            child.dist = cur.dist+1
            list.addEnd(child)
```

B,C,D,E
C,D,E,F,L

D,E,F,L,G,H
E,F,L,G,H,I
F,L,G,H,I,J
L,G,H,I,J,X
G,H,I,J,X,O
H,I,J,X,O

I,J,X,O,M
J,X,O,M
X,O,M,N
O,M,N
M,N

N



DFS: Stack

What is the runtime complexity?

What is the space complexity?

```
DF
// i
sta
list
while (!list.empty())
   cur = list.end()
   if (cur == stop)
      print cur.dist;
   else
      foreach child in cur.children
         if (child.dist == -1)
            child.dist = cur.dist+1
            list.addEnd(child)
```

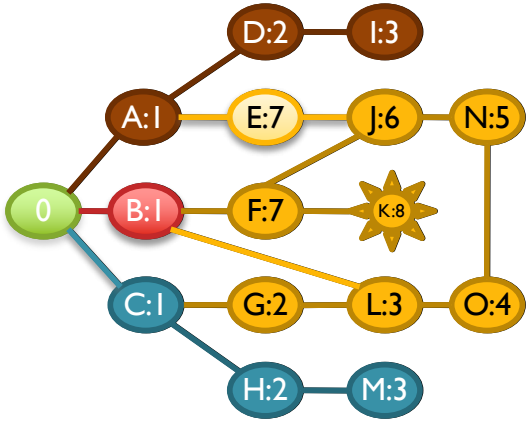
C

A,B,G,H
A,B,G,M

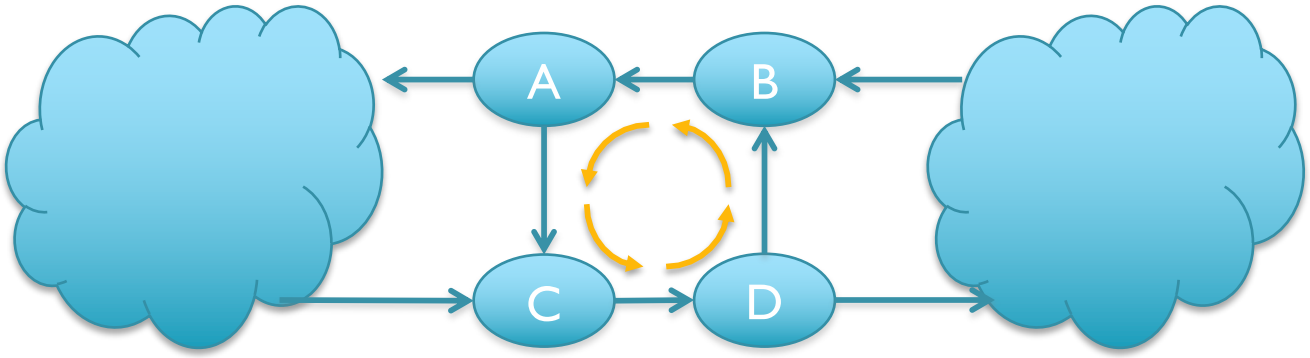
A,B,G
A,B,L
A,B,O
A,B,N
A,B,J
A,B,E,F
A,B,E,K
A,B,E

A,B

A
D
I



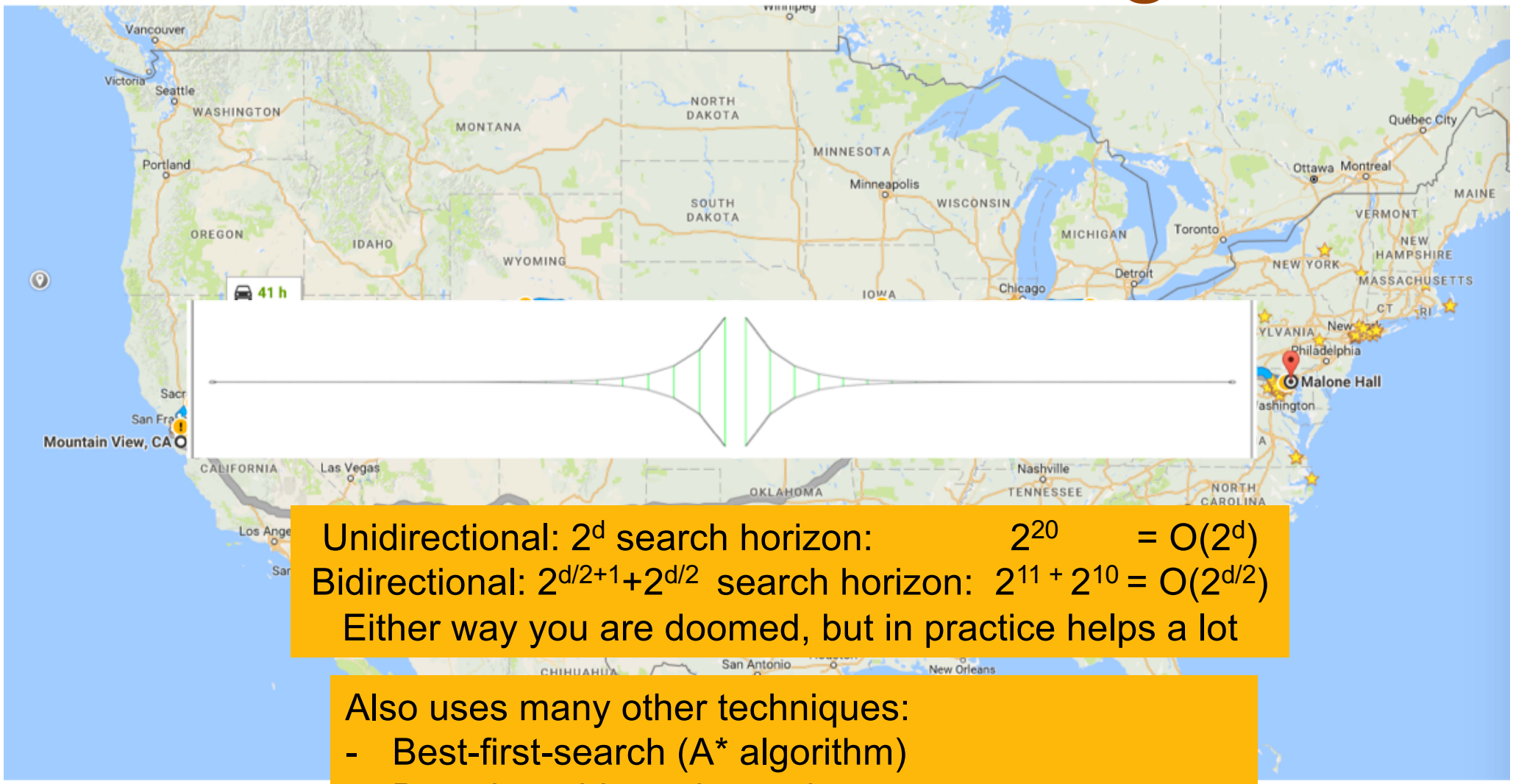
Graph Interface 6




```
public interface Graph<V,E> {
    ...
    Object label(Vertex<V> v);
    Object label(Edge<E> e);
    void label(Vertex<V> v, Object label);
    void label(Edge<E> e, Object label);
    void clearLabels();
    ...
}
```

Very flexible, but client will have to cast Object to correct type

Bi-Directional Breath First Searching



Unidirectional: 2^d search horizon: 2^{20} = $O(2^d)$

Bidirectional: $2^{d/2+1}+2^{d/2}$ search horizon: $2^{11} + 2^{10}$ = $O(2^{d/2})$

Either way you are doomed, but in practice helps a lot

Also uses many other techniques:

- Best-first-search (A* algorithm)
- Branch-and-bound search
- Multiple levels, Zones, & Precomputation

More to come…

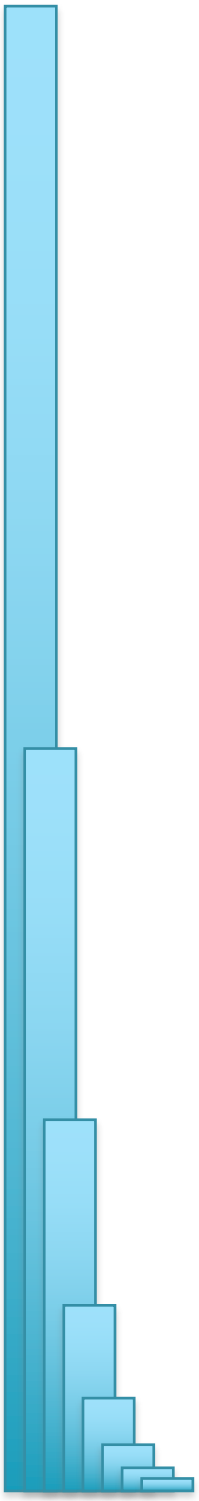
19.3 Splay Trees 181
19.4 Treaps 183
19.5 Bloom Filters 185

20 Graph Algorithms **188**
20.1 Topological Sorting 188
20.2 Minimum Spanning Trees 191
20.3 Shortest Paths 192

A How to write code… **197**
A.1 One Change at a Time aka Incremental Coding aka Baby Steps 197
A.2 Prototype to Learn aka Explore Small Examples 198
A.3 Don't Repeat Yourself aka Once and Only Once 198

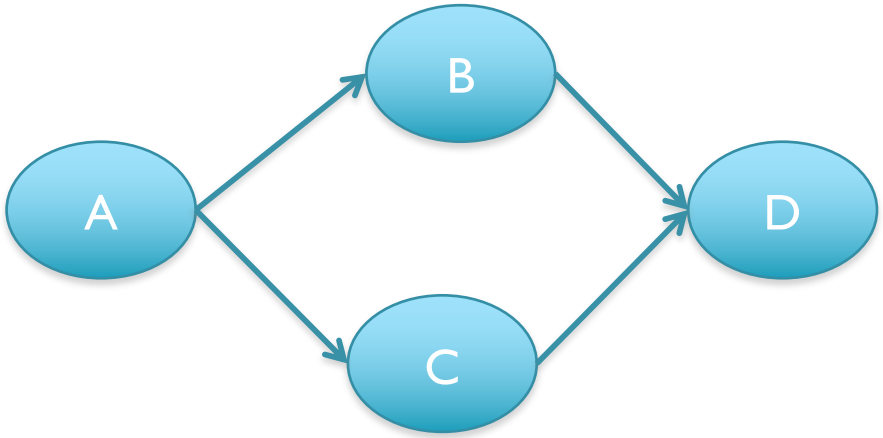
B Hacking AVL Trees **200**
B.1 Play with an AVL tree applet 200
B.2 Understand height/balance calculations 200
B.3 Figure out single rotations, implement, test 200
B.4 Figure out double rotations, implement, test 201
B.5 Write a general balancing method 202
B.6 Add balancing to your insert/remove code 202
B.7 Test, test, and test again 202

C Data Compression **203**
C.1 Communication 203
C.2 Encoding and Decoding 204
C.2.1 Fixed-Length Codes 204
C.2.2 Variable-Length Codes 205
C.3 Runlength Coding 206
C.3.1 Implementation Notes 207
C.4 Huffman Coding 207
C.4.1 Implementation Notes 210

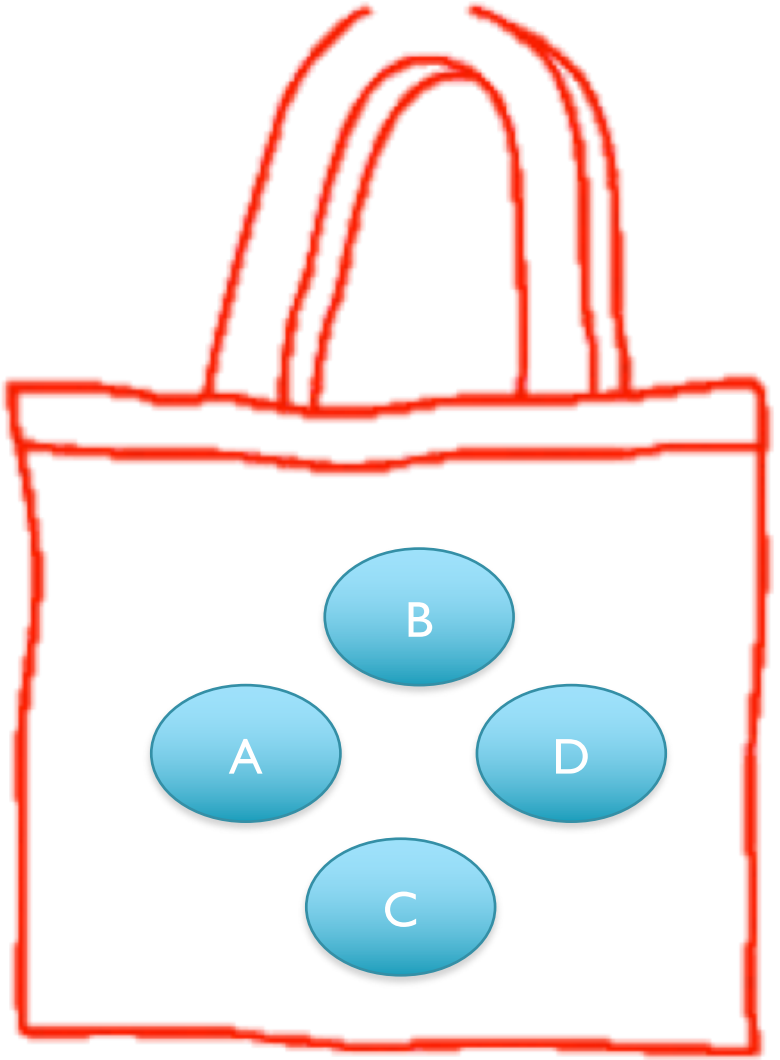


Part 2:Sets

Graphs versus Sets



Position-Based



Value-Based

Set Interface

```
public interface Set<T> {
    void insert(T t);
    void remove(T t);
    boolean has(T t);
}
```

Unordered collection of unique values

It doesn't matter how many times you insert X, it only "counts" once

It doesn't matter how many times you remove X, gone after first remove

This interface isnt super useful – Why?

Set Interface

```
public interface Set<T> {
    void insert(T t);
    void remove(T t);
    boolean has(T t);

    boolean empty();
    T any() throws EmptySetException;
}
```

any() returns an arbitrary (but not necessarily random) element from set:
First or last added, could even be the same one over and over again

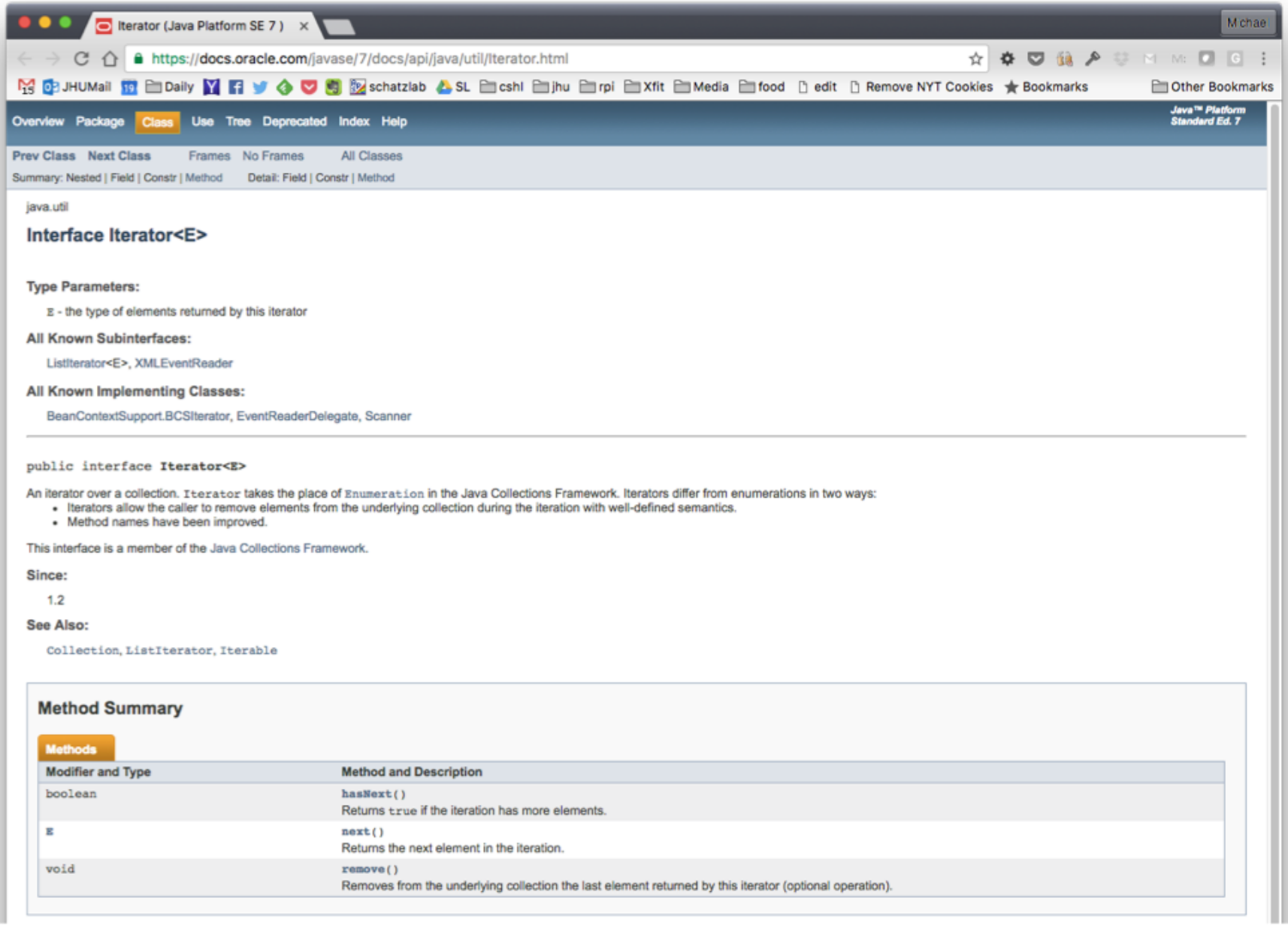
How would you use this interface to print
every element in the set?

Set Interface

```
public interface Set<T> implements Iterable<T> {
    void insert(T t);
    void remove(T t);
    boolean has(T t);

    boolean empty();
    T any() throws EmptySetException;

    Iterator<T> iterator();
}
```

Now we can actually get all the values without destroying the set ☺

Iterator Interface



Implementations

Ideas????

(Doubly) Linked List: Fast add/remove once found, slow to find

Array: Fast add, remove can be slow, still have to find

Tree: Fast add, maybe fast to remove?, still have to find, keep tree balanced

Stack, Queue, Deque, Graph: Probably wont work ☹

We can make add O(1) by "cheating" and allowing multiple copies and delete them all on remove … but requires O(n) space!

Speed up Array by marking elements deleted but don't shift things down (allow gaps in array)

Sorting may be really useful so can do lookup in lg(n) time ☺
… but only if the type supports a comparison!

ArraySet (1)

```
public class ArraySet<T> implements Set<T> {
    private int length;
    private T[] data;

    private static class SetIterator <T> implements Iterator<T> {
        private int current;
        private int length;
        private T[] data;

        public SetIterator (int length, T[] data) {
            this.data = data;
            this.length = length;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }

        public boolean hasNext () {
            return this.current < this.length;
        }

        public T next () {
            T t = this.data[this.current];
            this.current +=1;
            return t;
        }
    }
```

In case the set changes

ArraySet (2)

```
…
    public ArraySet() {
        this.data = (T[]) new Object[1];
    }

    public Iterator <T> iterator () {
        return new SetIterator <T> (this.length, this.data);
    }

    private void grow() {
        T[] bigger = (T[]) new Object[2 * this.length ];
        for (int i =0; i <this.length; i++){
            bigger[i] = this.data[i];
            this.data = bigger;
        }
    }

    public void insert(T t) {
        if (this.has(t)) { return; }
        if (this.length == this.data.length) { this.grow(); }
        this.data[this.length] = t;
        this.length += 1;
    }
```

ArraySet (3)

```
    private int find(T t) {
        for (int i = 0; i < this.length; i++) {
            if (this.data[i].equals(t)) { return i; }
        }
        return -1;
    }

    public void remove(T t) {
        int position = this.find(t);
        if (position == -1) {return; }
        for (int i = position; i < this.length -1; i++) {
            this.data[i] = this.data[i+1];
        }
        this.length -= 1;
    }

    public boolean has(T t) {
        return this.find(t) != -1;
    }

    public boolean empty() {
        return this.length == 0;
    }
}
```

ArraySet (3)

```
    private int find(T t) {
        for (int i = 0; i < this.length; i++) {
            if (this.data[i].equals(t)) { return i; }
        }
        return -1;
    }
```

Why .equals()?

== is a reference comparison, i.e. both objects point to the same memory location
.equals() evaluates to the comparison of values in the objects

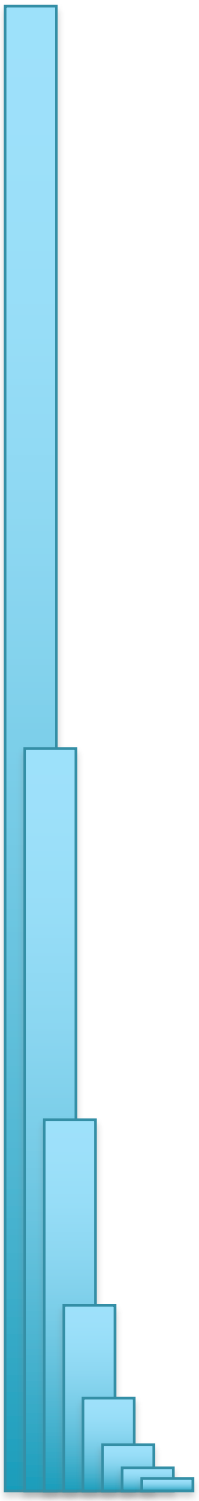
```
        if (position == -1) {return; }
        for (int i = position; i < this.length -1; i++) {
            this.data[i] = this.data[i+1];
        }
        this.length -= 1;
    }

    public boolean has(T t) {
        return this.find(t) != -1;
    }

    public boolean empty() {
        return this.length == 0;
    }
}
```

TestSetIterator (1)

```
import org.junit.Test;
import org.junit.Before;
import static org.junit.Assert.assertEquals;
import java.util.Iterator;

public class TestSetIterator {
    Set<String> set;

    @Before
    public void setupSet() {
        set = new ArraySet<String>();
    }

    @Test
    public void testEmptyIterator() {
        Iterator<String> i = set.iterator();
        assertEquals(false, i.hasNext());
    }
…
```

TestSetIterator (2)

```
…
    @Test
    public void testSingletonIterator() {
        set.insert("Paul");
        Iterator<String> i = set.iterator();
        assertEquals(true, i.hasNext());
        String s = i.next();
        assertEquals("Paul", s);
        assertEquals(false, i.hasNext());
    }
…
```



TestSetIterator (3)

```
…

    @Test
    public void testIterator() {
        String [] data = {"Peter", "Paul", "Mary", "Beverly"};
        for (String d: data) {
            set.insert(d);
        }

        for(String s : set) {
            int count = 0;
            for (String d: data) {
                if (s.equals(d)) {
                    count += 1;
                }
            }
            assertEquals(1, count);
        }
    }
}
```

./ runTestSetIterator.sh

What does this code remind you of?

Unique!

```
import java.util.Scanner;

public final class Unique {
    private static Set<Integer> data;
    private Unique() { }

    public static void main(String[] args) {
        data = new ArraySet<Integer>();
        Scanner scanner = new Scanner(System.in);
        while (scanner.hasNextInt()) {
            int i = scanner.nextInt();
            data.insert(i);
        }

        for (Integer i : data) {
            System.out.println(i);
        }
    }
}
```

Performance Testing

```
$ seq 1 1000    | awk '{print int(rand()*100000)}' > random1k.txt
$ seq 1 10000   | awk '{print int(rand()*100000)}' > random10k.txt
$ seq 1 100000  | awk '{print int(rand()*100000)}' > random100k.txt

$ time java Unique < random1k.txt > bla
real 0m0.176s
user 0m0.242s
sys  0m0.041s

$ wc –l bla
993 bla

$ time java Unique < random10k.txt > bla
real 0m0.368s
user 0m0.786s
sys  0m0.088s

$ wc –l bla
9529 bla

$ time java Unique < random100k.txt > bla
real 0m4.466s
user 0m4.735s
sys  0m0.279s

$ wc –l bla
63110 bla
```

Performance Testing

```
$ seq 1 1000     | awk '{print int(rand()*100000)}' > random1k.txt
$ seq 1 10000
$ seq 1 100000

$ time java Uniq
real 0m0.176s
user 0m0.242s
sys  0m0.041s

$ wc –l bla
993 bla

$ time java Uniq
real 0m0.368s
user 0m0.786s
sys  0m0.088s

$ wc –l bla
9529 bla

$ time java Uniq
real 0m4.466s
user 0m4.735s
sys  0m0.279s

$ wc –l bla
63110 bla
```

```
$ seq 1 100000   | awk '{print int(rand()*1000)}' > spread1k.txt
$ seq 1 100000   | awk '{print int(rand()*10000)}' > spread10k.txt
$ seq 1 100000   | awk '{print int(rand()*100000)}' > spread100k.txt

$ time java Unique < spread1k.txt > bla
real 0m0.426s
user 0m0.750s
sys  0m0.104s

$ wc –l bla
 1000 bla

$ time java Unique < spread10k.txt > bla
real 0m1.083s
user 0m1.754s
sys  0m0.195s

$ wc –l bla
 10000 bla

$ time java Unique < spread100k.txt > bla
real 0m5.930s
user 0m5.910s
sys  0m0.649s

$ wc –l bla
   63110 bla
```

ListSet (1)

```
import java.util.Iterator;
    public class ListSet <T> implements Set<T> {
        private static class Node<T> {
            T data;
            Node<T> next;
            Node<T> prev;
        }

        private Node<T> head;

        private static class SetIterator <T> implements Iterator<T> {
            private Node<T> current;

            public SetIterator (Node<T> head) {
                this.current = head;
            }

            public void remove() {
                 throw new UnsupportedOperationException();
            }

            public boolean hasNext() {
                 return this.current != null;
            }

            public T next () {
                T t = this.current.data;
                this.current = this.current.next;
                return t;
            }
        }
```

ListSet (2)

```
public void insert (T t) {
    if (this.has(t)) { return; }
    Node<T> n = new Node<T>(); n.data = t;
    n.next = this.head;
    n.prev = null;
    if (this.head != null) {
        this.head.prev = n;
    }
    this.head = n;
}

private Node<T> find(T t) {
    for (Node<T> n = this.head; n != null; n = n.next) {
        if (n.data.equals(t)) { return n; }
    }
    return null;
}

public void remove(T t) {
    Node<T> position = this.find(t);
    if (position == null) { return; }
    if (position.next != null) {
        position.next.prev = position.prev;
    }

    if (position.prev != null) {
        position.prev.next = position.next;
    }
}
```

ListSet (3)

```
        public boolean has(T t) {
            return this.find(t) != null;
        }

        public Iterator <T> iterator () {
            return new SetIterator <T>(this.head);
        }
}
```

TestListSetIterator (1)

```
import org.junit.Test;
import org.junit.Before;
import static org.junit.Assert.assertEquals;
import java.util.Iterator;

public class TestListSetIterator {
    Set<String> set;

    @Before
    public void setupSet() {
        set = new ListSet<String>();
    }

    @Test
    public void testEmptyIterator() {
        Iterator<String> i = set.iterator();
        assertEquals(false, i.hasNext());
    }

    @Test
    public void testSingletonIterator() {
        set.insert("Paul");
        Iterator<String> i = set.iterator();
        assertEquals(true, i.hasNext());
        String s = i.next();
        assertEquals("Paul", s);
        assertEquals(false, i.hasNext());
    }
…
```

TestListSetIterator (2)

```
…

    @Test
    public void testIterator() {
        String [] data = {"Peter", "Paul", "Mary", "Beverly"};
        for (String d: data) {
            set.insert(d);
        }

        for(String s : set) {
            int count = 0;
            for (String d: data) {
                if (s.equals(d)) {
                    count += 1;
                }
            }
            assertEquals(1, count);
        }
    }
}
```

UniqueList

```
import java.util.Scanner;

public final class UniqueList {
          private static Set<Integer> data;
          private UniqueList() { }

          public static void main(String[] args) {
               data = new ListSet<Integer>();
               Scanner scanner = new Scanner(System.in);

               long before = System.nanoTime();

               while (scanner.hasNextInt()) {
                    int i = scanner.nextInt();
                    data.insert(i);
               }

               for (Integer i : data) {
                    System.out.println(i);
               }

               long duration = System.nanoTime() - before;
               System.err.println(duration / 1e9);
          }
}
```

Let the code report how long it ran

Performance Testing

```
$ time java UniqueList < random1k.txt > bla
$ time java UniqueList < random10k.txt > bla
$ time java UniqueList < random100k.txt > bla

## Enable nanosecond timing

$ java UniqueList < spread1k.txt > bla
$ java UniqueList < spread10k.txt > bla
$ java UniqueList < spread100k.txt > bla
```

How do Unique (ArraySet) and UniqueList (ListSet) compare?

Why?

Where does the time go?

```
$ java -agentlib:hprof=cpu=times UniqueList < random1k.txt > bla
$ less java.hprof.txt

…

CPU TIME (ms) BEGIN (total = 3556) Wed Oct 19 23:51:10 2016
rank   self  accum   count trace method
   1 30.34% 30.34%  494953 304975 java.lang.Integer.equals
   2 16.73% 47.08%    1000 304958 ListSet.find
   3 12.09% 59.17%  494953 304974 java.lang.Integer.intValue
   4  0.96% 60.12%    8918 304826 java.nio.HeapCharBuffer.get
   5  0.87% 61.00%    5918 304837 java.util.regex.Pattern$CharProperty.match
   6  0.84% 61.84%    1988 305038 sun.nio.cs.UTF_8$Encoder.encodeArrayLoop
   7  0.82% 62.65%    6921 304827 java.nio.CharBuffer.charAt
   8  0.79% 63.44%    1008 304796 java.util.Scanner.getCompleteTokenInBuffer
   9  0.73% 64.17%    5918 304835 java.lang.Character.codePointAt
  10  0.59% 64.76%    4903 304874 java.util.regex.Pattern$BmpCharProperty.match
  11  0.56% 65.33%    1988 305064 sun.nio.cs.StreamEncoder.writeBytes
  12  0.56% 65.89%    6921 304833 java.lang.Character.isWhitespace
  13  0.53% 66.42%    1988 305045 sun.nio.cs.StreamEncoder.implWrite
  14  0.51% 66.93%    4903 304943 java.lang.Character.digit
  15  0.51% 67.44%    8918 304832 java.lang.CharacterDataLatin1.isWhitespace
  16  0.45% 67.89%    1988 305036 sun.nio.cs.UTF_8.updatePositions
  17  0.45% 68.34%    4903 304865 java.nio.HeapCharBuffer.get
  18  0.45% 68.79%    3000 304908 java.nio.HeapCharBuffer.subSequence
  19  0.42% 69.21%    4903 304866 java.nio.CharBuffer.charAt
  20  0.42% 69.63%    5918 304838 java.util.regex.Pattern$Curly.match
```

Where does the time go?

```
$ java -agentlib:hprof=cpu=times Unique < random1k.txt > bla
$ less java.hprof.txt

…

CPU TIME (ms) BEGIN (total = 3673) Wed Oct 19 23:52:45 2016
rank   self  accum   count trace method
   1 28.34% 28.34%  494926 304968 java.lang.Integer.equals
   2 18.35% 46.69%    1000 304958 ArraySet.find
   3 12.55% 59.24%  494926 304967 java.lang.Integer.intValue
   4  1.03% 60.28%    6921 304827 java.nio.CharBuffer.charAt
   5  0.76% 61.04%    4903 304866 java.nio.CharBuffer.charAt
   6  0.76% 61.80%    1988 305032 sun.nio.cs.UTF_8$Encoder.encodeArrayLoop
   7  0.74% 62.54%    1008 304796 java.util.Scanner.getCompleteTokenInBuffer
   8  0.68% 63.22%    3000 304908 java.nio.HeapCharBuffer.subSequence
   9  0.65% 63.87%    6921 304833 java.lang.Character.isWhitespace
  10  0.57% 64.44%    4903 304874 java.util.regex.Pattern$BmpCharProperty.match
  11  0.57% 65.01%    1988 305058 sun.nio.cs.StreamEncoder.writeBytes
  12  0.54% 65.56%    5918 304837 java.util.regex.Pattern$CharProperty.match
  13  0.54% 66.10%    1988 305039 sun.nio.cs.StreamEncoder.implWrite
  14  0.52% 66.62%    5918 304835 java.lang.Character.codePointAt
  15  0.52% 67.14%    6016 304786 java.nio.CharBuffer.length
  16  0.52% 67.66%    8918 304832 java.lang.CharacterDataLatin1.isWhitespace
  17  0.49% 68.15%    1988 305052 java.io.FileOutputStream.write
  18  0.49% 68.64%    8918 304826 java.nio.HeapCharBuffer.get
  19  0.46% 69.10%    1000 304945 java.lang.Integer.parseInt
  20  0.44% 69.53%    1999 301324 java.nio.Buffer.<init>
```

Where does the time go?

```
$ java -agentlib:hprof=cpu=samples Unique < random1k.txt > bla
$ less java.hprof.txt

…

CPU SAMPLES BEGIN (total = 9) Wed Oct 19 23:54:25 2016
rank   self  accum   count trace method
   1 11.11% 11.11%       1 300194 ArraySet.has
   2 11.11% 22.22%       1 300196 java.io.FileOutputStream.writeBytes
   3 11.11% 33.33%       1 300043 java.util.zip.ZipFile.getZipEntry
   4 11.11% 44.44%       1 300023 sun.misc.Unsafe.ensureClassInitialized
   5 11.11% 55.56%       1 300154 sun.util.resources.LocaleData$LocaleDataResourceBundleCon
   6 11.11% 66.67%       1 300085 java.util.regex.Pattern$CharPropertyNames.<clinit>
   7 11.11% 77.78%       1 300192 java.util.regex.Pattern$CharProperty.match
   8 11.11% 88.89%       1 300193 ArraySet.find
   9 11.11% 100.00%       1 300195 sun.nio.cs.UTF_8$Encoder.encodeArrayLoop
CPU SAMPLES END
```

Samples every 10ms by default
Much faster to process, not as precise of counts

What memory did we use?

```
$ java -agentlib:hprof UniqueList < random1k.txt > bla
$ less java.hprof.txt

…

SITES BEGIN (ordered by live bytes) Wed Oct 19 23:58:28 2016
          percent          live          alloc'ed  stack class
 rank   self  accum     bytes objs     bytes  objs trace name
    1 14.47% 14.47%    304000 1000    304000  1000 301198 int[]
    2  8.00% 22.47%    168000 1000    168000  1000 301199 int[]
    3  6.86% 29.32%    144000 3000    144000  3000 301200 java.nio.HeapCharBuffer
    4  5.29% 34.61%    111024 1413    111024  1413 300000 char[]
    5  4.61% 39.22%     96768 1008     96768  1008 301195 int[]
    6  4.57% 43.78%     95952 1999     95952  1999 300265 java.nio.HeapCharBuffer
    7  4.47% 48.26%     93960 3000     93960  3000 301202 char[]
    8  3.43% 51.68%     72000 3000     72000  3000 301201 java.lang.String
    9  1.94% 53.62%     40680  172     40680   172 300011 char[]
   10  1.62% 55.24%     34096 1399     34096  1399 300000 java.lang.String
   11  1.52% 56.77%     32024  554     32024   554 300000 java.lang.Object[]
   12  1.48% 58.25%     31096  993     31096   993 301207 char[]
   13  1.21% 59.46%     25504    8     25504     8 300000 byte[]
   14  1.17% 60.63%     24624    3     24624     3 300259 byte[]
   15  1.16% 61.80%     24448  326     24448   326 301241 int[]
   16  1.13% 62.93%     23832  993     23832   993 301206 java.lang.String
   17  1.13% 64.07%     23832  993     23832   993 301205 ListSet$Node
   18  0.78% 64.85%     16400    1     16400     1 300674 char[]
   19  0.78% 65.63%     16400    1     16400     1 300262 char[]
   20  0.77% 66.40%     16128 1008     16128  1008 301196 int[]
```

What memory did we use?

```
$ java8 -agentlib:hprof Unique < random1k.txt > bla
$ less java.hprof.txt

…

SITES BEGIN (ordered by live bytes) Wed Oct 19 23:57:02 2016
          percent          live          alloc'ed  stack class
 rank   self  accum     bytes objs     bytes  objs trace name
    1 14.62% 14.62%    304000 1000    304000  1000 301199 int[]
    2  8.08% 22.70%    168000 1000    168000  1000 301200 int[]
    3  6.93% 29.63%    144000 3000    144000  3000 301201 java.nio.HeapCharBuffer
    4  5.34% 34.97%    111040 1415    111040  1415 300000 char[]
    5  4.65% 39.63%     96768 1008     96768  1008 301196 int[]
    6  4.62% 44.24%     95952 1999     95952  1999 300265 java.nio.HeapCharBuffer
    7  4.52% 48.76%     93960 3000     93960  3000 301203 char[]
    8  3.46% 52.22%     72000 3000     72000  3000 301202 java.lang.String
    9  1.90% 54.12%     39424  167     39424   167 300011 char[]
   10  1.64% 55.76%     34144 1401     34144  1401 300000 java.lang.String
   11  1.54% 57.30%     32024  554     32024   554 300000 java.lang.Object[]
   12  1.50% 58.80%     31096  993     31096   993 301208 char[]
   13  1.23% 60.03%     25504    8     25504     8 300000 byte[]
   14  1.18% 61.21%     24624    3     24624     3 300259 byte[]
   15  1.18% 62.39%     24448  326     24448   326 301242 int[]
   16  1.15% 63.53%     23832  993     23832   993 301209 java.lang.String
   17  0.79% 64.32%     16400    1     16400     1 300262 char[]
   18  0.79% 65.11%     16400    1     16400     1 300675 char[]
   19  0.78% 65.89%     16128 1008     16128  1008 301197 int[]
   20  0.77% 66.66%     15984  999     15984   999 301204 java.lang.Integer
...
   26  0.40% 70.01%      8344   10      8344    10 301206 java.lang.Object[]
```

jb: Simple Go-style benchmarking for Java

Welcome to `jb`, also known as `jaybee` out in the shell.

This is a port of Go's benchmarking facilities to Java. I needed `jb` for the data structures course I taught in Fall 2016, mostly to make it easier on students to get some basic performance metrics for their code. The tool has been reasonably well-received over the years (Spring 2017, Fall 2017, Spring 2018) and students who have previously learned about JUnit 4 seem to pick it up with ease.

I borrowed *very* liberally from the Go original as well as from the ports listed below. One might say that my only accomplishment was to package things in a slightly more Java-like way. (Or maybe the only *real* advantage is that I also remixed a cute logo for it? Bzzz! Oh, I would like to apologize for taking the whole bee metaphor a little *too* far in the source code.)

I welcome pull requests!

Building

If you want to build `jaybee` yourself, you'll first need to install maven. I actually feel a bit guilty about having switched from a hacky `Makefile` to an insanely huge monster of a build system, but hey, it's 2018 so let's waste lots of cycles and even more space to do very simple things.

Once you have maven installed go ahead and clone the repository. Then say `mvn install` and after waiting patiently for a while you'll have the JAR file referenced below.

If you are on a UNIX system, you should be able to say `make` which will run `mvn install` as above before packaging the JAR into a "fake executable" called `jaybee` that can be used to run benchmarks more conveniently. (At some point I might add `jaybeecee` as well, let's see.)

Usage

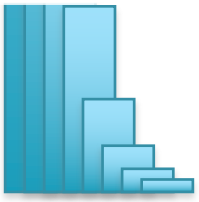
If `StringAppend.java` contains a bunch of methods annotated with `@Bench`, here's how you compile it (provided `jaybee.jar` is in the current directory):

```
$ javac -classpath jaybee.jar:. StringAppend.java
```

If `StringAppend.class` contains a bunch of compiled benchmark methods, here's how you run them (provided `jaybee.jar` is in the current directory):

```
$ java -jar jaybee.jar StringAppend
       string           5,000         326,858 ns/op           8.84 MB/s             156,993 B/op
 stringBuffer          50,000          26,474 ns/op         109.16 MB/s              18,685 B/op
stringBuilder         100,000          13,622 ns/op         212.15 MB/s                 339 B/op
```

You get (a) the number of times the benchmark was run, (b) the time it took per iteration, (c) the amount of data processed per second, and (d) the number of bytes allocated per iteration. (That last number can be rather flakey because garbage collection behavior is not exactly deterministic.)



Webpage: https://github.com/phf/jb
Available in resources/jaybee-1.0.jar

CompareSets.java

```
import com.github.phf.jb.Bench;
import com.github.phf.jb.Bee;
import java.util.Random;

public final class CompareSets {
    private static final Random r = new Random();
    private static final int SPREAD = 100000;
    private static final int SIZE   = 1000;

    @Bench
    public void randomArraySet(Bee b) {
      for (int i = 0; i < b.reps(); i++)
      {
        ArraySet<Integer> set = new ArraySet<Integer>();
        for (int j = 0; j < SIZE; j++)
        {
          set.insert(r.nextInt(SPREAD));
        }
        b.bytes(SIZE * 4);
      }
    }

    @Bench
    public void randomListSet(Bee b) {
      for (int i = 0; i < b.reps(); i++)
      {
        ListSet<Integer> set = new ListSet<Integer>();
        for (int j = 0; j < SIZE; j++)
        {
          set.insert(r.nextInt(SPREAD));
        }
        b.bytes(SIZE * 4);
      }
    }
}
```

Just like Junit!
@Bench

Just like Junit!
@Bench

CompareSets.java

```
import com.github.phf.jb.Bench;
import com.github.phf.jb.Bee;
import java.util.Random;

public final class CompareSets {
    private static final Random r = new Random();
    private static final int SPREAD = 100000;
    private static final int SIZE   = 1000;

    @Bench
    public void randomArraySet(Bee b) {
      for (int i = 0; i < b.reps(); i++)
      {
        ArraySet<Integer> set = new ArraySet<Integer>();
        for (int j = 0; j < SIZE; j++)
        {
          set.insert(r.nextInt(SPREAD));
        }
        b.bytes(SIZE * 4);
      }
    }

    public void randomListSet(Bee b) {
      for (int i = 0; i < b.reps(); i++)
      {
        ListSet<Integer> set = new ListSet<Integer>();
        for (int j = 0; j < SIZE; j++)
        {
          set.insert(r.nextInt(SPREAD));
        }
        b.bytes(SIZE * 4);
      }
    }
}
```

```
$ javac -cp .:jb/bin/jaybee.jar CompareSets.java
$ java –jar jaybee.jar CompareSets

randomArraySet   3000     434119 ns/op      9.21 MB/s      1839 B/op
 randomListSet   1000    1270848 ns/op      3.15 MB/s      6190 B/op
```

Next Steps

1. Get Ready for HW5

2. Check on Piazza for tips & corrections!