CS 600.226: Data Structures

Michael Schatz

Oct 3 2018

Lecture 16. More Graphs

Agenda

1. ***Questions on HW4***
2. ***Recap on Trees***
3. ***Graphs***

Assignment 4: Due Friday Oct 5 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment04/README.md

Assignment 4: Stacking Queues

Out on: September 28, 2018
Due by: October 5, 2018 before 10:00 pm
Collaboration: None
Grading:

Packaging 10%,
Style 10% (where applicable),
Testing 10% (where applicable),
Performance 10% (where applicable),
Functionality 60% (where applicable)

Overview
The fourth assignment is mostly about stacks and dequeues. For the former you'll build a simple calculator application, for the latter you'll implement the data structure in a way that satisfies certain performance characteristics (in addition to the usual correctness properties).

Assignment 4: Due Friday Oct 5 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment04/README.md

Problem 1: Calculating Stacks (50%)

Your first task is to implement a basic RPN calculator that supports integer operands like 1, 64738, and -42 as well as the (binary) integer operators +, -, *, /, and %. Your program should be called `Calc` and work as follows:

- You create an empty `Stack` to hold intermediate results and then repeatedly accept input from the user. It doesn't matter whether you use the `ArrayStack` or the `ListStack` we provide, what does matter is that those specific types appear only once in your program.
- If the user enters a ***valid integer***, you ***push*** that integer onto the stack.
- If the user enters a ***valid operator***, you ***pop*** two integers off the stack, ***perform*** the requested operation, and ***push*** the result back onto the stack.
- If the user enters the symbol `?` (that's a question mark), you ***print*** the current state of the stack using its `toString` method followed by a new line.
- If the user enters the symbol `.` (that's a dot or full-stop), you ***pop*** the top element off the stack and ***print*** it (only the top element, not the entire stack) followed by a new line.
- If the user enters the symbol `!` (that's an exclamation mark or bang), you exit the program.

Assignment 4: Due Friday Oct 5 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment04/README.md

```
$ java Calc
?
[]
10
?
[10]
20 30
?
[30, 20, 10]
*
?
[600, 10]
+
?
[610]
.
610
!
$
```

```
$ java Calc
? 10 ? 20 30 ? *
? + ? . !
[]
[10]
[30, 20, 10]
[600, 10]
[610]
610
$
```

Assignment 4: Due Friday Oct 5 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment04/README.md

Problem 2: Hacking Growable Dequeues (50%)

Your second task is to implement a generic `ArrayDequeue` class as outlined in lecture. As is to be expected, `ArrayDequeue` must implement the `Dequeue` interface we provided on github.

- Your implementation must be done in terms of an array that grows by doubling as needed. It's up to you whether you want to use a basic Java array or the SimpleArray class you know and love; just in case you prefer the latter, we've once again included it on the github directory for this assignment. Your initial array must have a length of one slot only! (Trust us, that's going to make debugging the "doubling" part a lot easier.)
- Your implementation must support all `Dequeue` operations except insertion in (worst-case) constant time; insertion can take longer every now and then (when you need to grow the array), but overall all insertion operations must be constant amortized time as discussed in lecture.
- You should provide a `toString` method in addition to the methods required by the `Dequeue` interface. A new dequeue into which 1, 2, and 3 were inserted using `insertBack()` should print as [1, 2, 3] while an empty dequeue should print as []

Assignment 4: Due Friday Oct 5 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment04/README.md

Bonus Problem (5 pts)

Develop an **algebraic specification** for the **abstract data type Queue**. Use `new`, `empty`, `enqueue`, `dequeue`, and `front` (with the meaning of each as discussed in lecture) as your set of operations. Consider unbounded queues only.

The difficulty is going to be modelling the FIFO (first-in-first-out) behavior accurately. You'll probably need at least one axiom with a case distinction using an if expression; the syntax for this in the Array specification for example.

Doing this problem without resorting to Google may be rather helpful for the upcoming midterm. There's no need to submit the problem, but you can submit it if you wish; just include it at the end of your README file.

Agenda

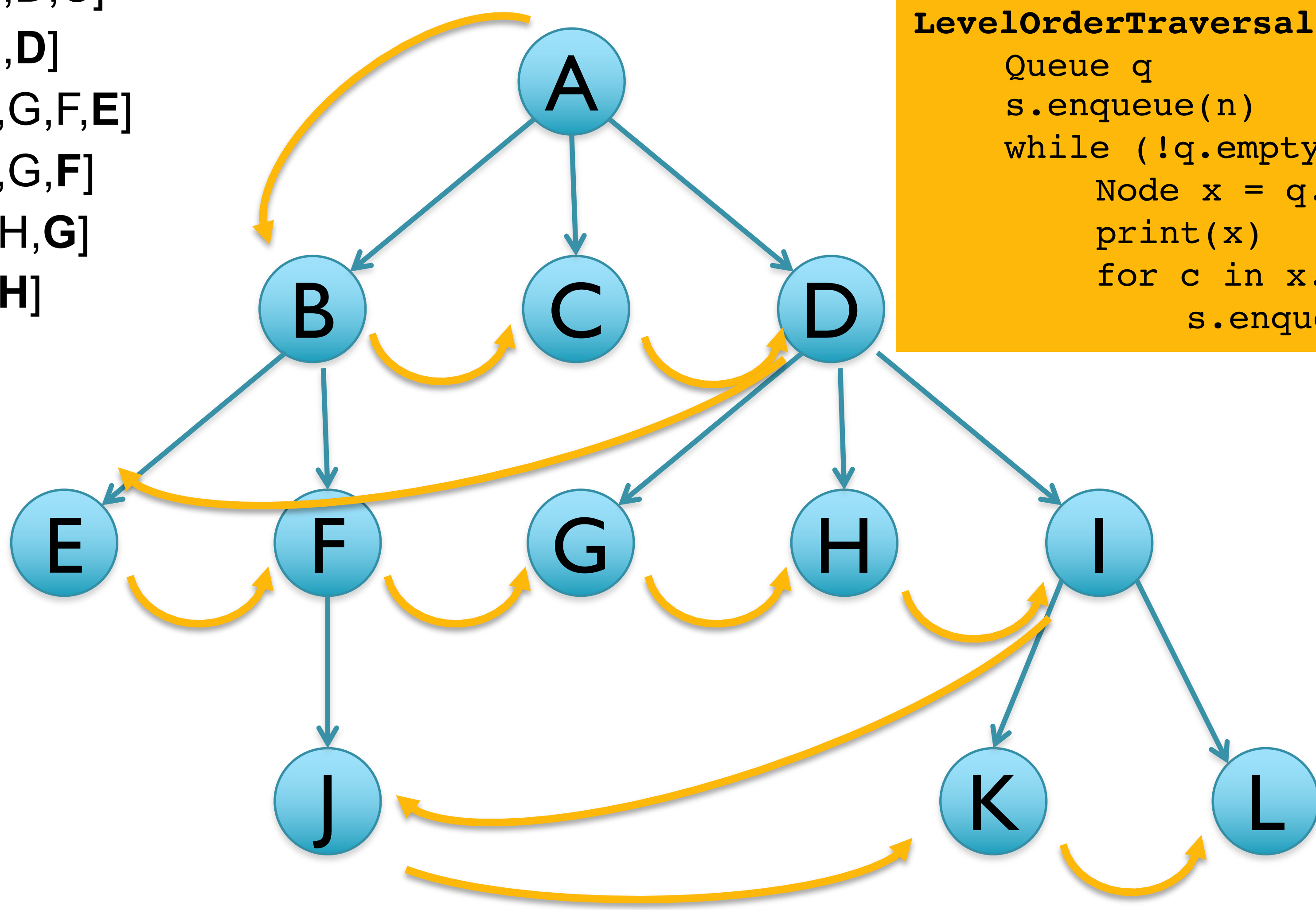
1. ***Questions on HW4***
2. ***Recap on Trees***
3. ***Graphs***

Trees

Level Order Traversals

[**A**]
[D,C,**B**]
[F,E,D,**C**]
[F,E,**D**]
[I,H,G,F,**E**]
[I,H,G,**F**]
[J,I,H,**G**]
[J,I,**H**]
…

Queue leads to a Breadth-First Search

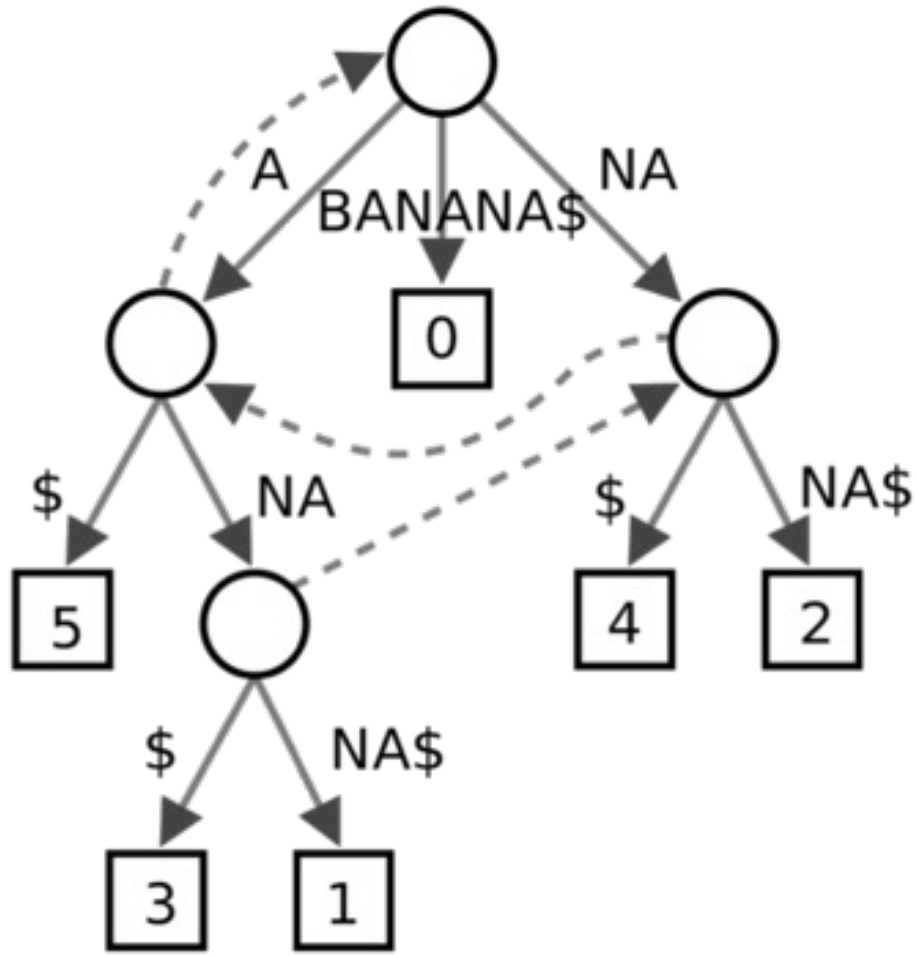


How to level order print?

(A) (B C D) (E F G H I) (J K L)

```
LevelOrderTraversal(Node n):
    Queue q
    s.enqueue(n)
    while (!q.empty()):
        Node x = q.dequeue()
        print(x)
        for c in x.children:
            s.enqueue(c)
```

Would you ever add “jump edges”?



Yes! In a special tree called a “suffix tree”, ”suffix links” allow us to navigate between nodes that have a special relationship that is hard to otherwise compute

More to come…

Inheritance

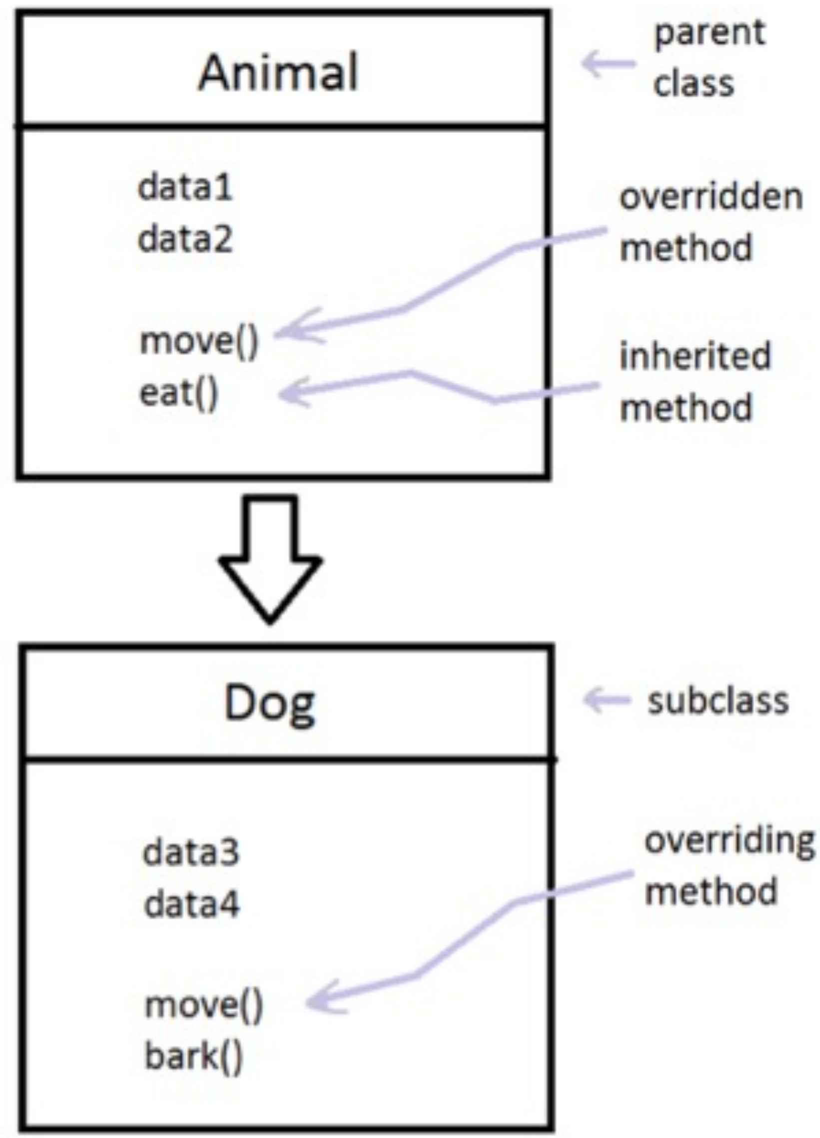
Why Inheritance?

Code Reuse
- Subclass gets to use all methods of the parent class "for free" by extending the parent class

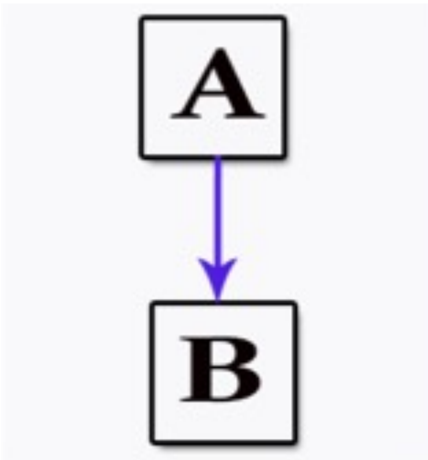
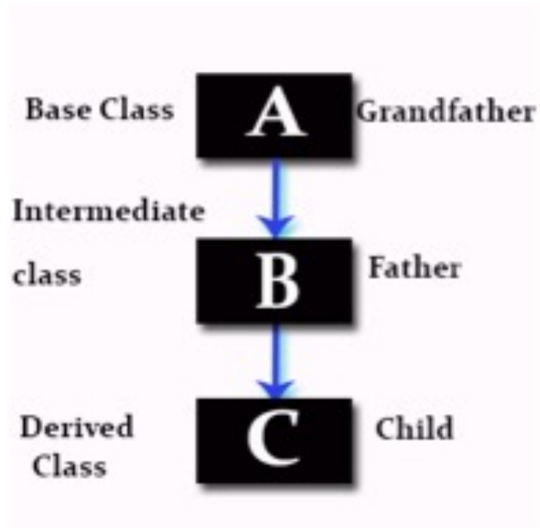
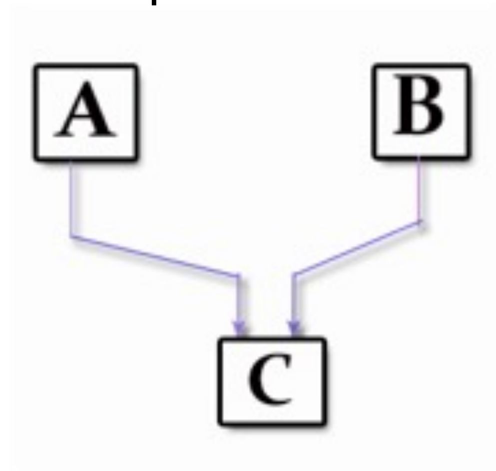
Overriding
- Subclass can have more specific implementation than the parent class

Design constraints
- Subclasses get all of the features of the parent, whether you like them or not!

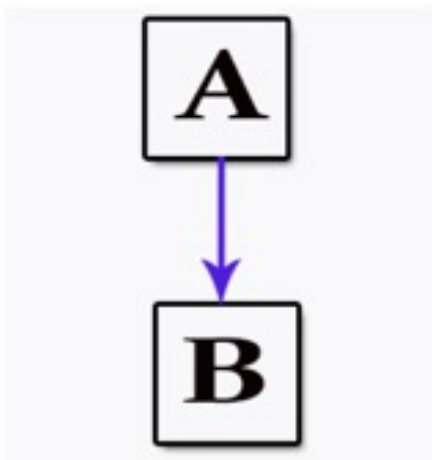
Saying B inherits from A is a very strong relationship: anytime that A could be used, B could be instead



Inheritance Types

Single Inheritance	Multilevel Inheritance	Multiple Inheritance
		
B isa A	C isa B, B isa A	C isa A, C isa B
Square isa Rectangle	Square isa Rectangle Rectangle isa Shape => Square isa Shape	Mike isa CS Prof Mike isa Bio Prof (not in Java!)

Inheritance versus Encapsulation



B isa A

Square isa
Rectangle

Photo

Position pos
String name
String URL

StringList favorites

Position

Double lat
Double long
Double altitude

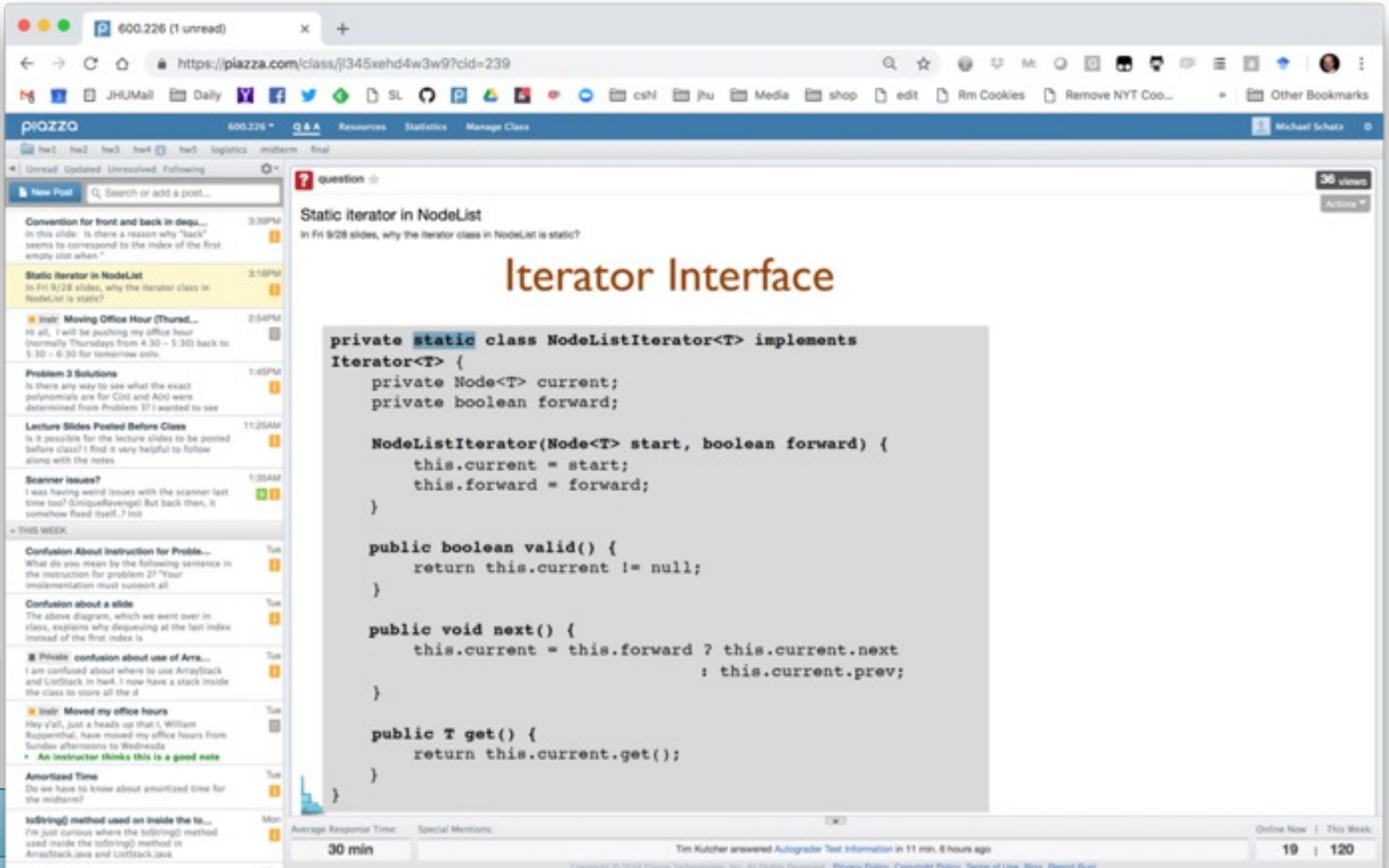
StringList

int capacity
String [] data

Photo hasa Position,
Photo hasa favorites,
etc

Encapsulation is used to hide the values or state of a structured data object inside a class,

When to use static?

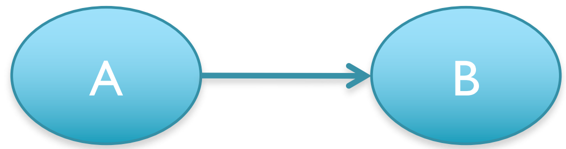


Graphs

Graphs are Everywhere!

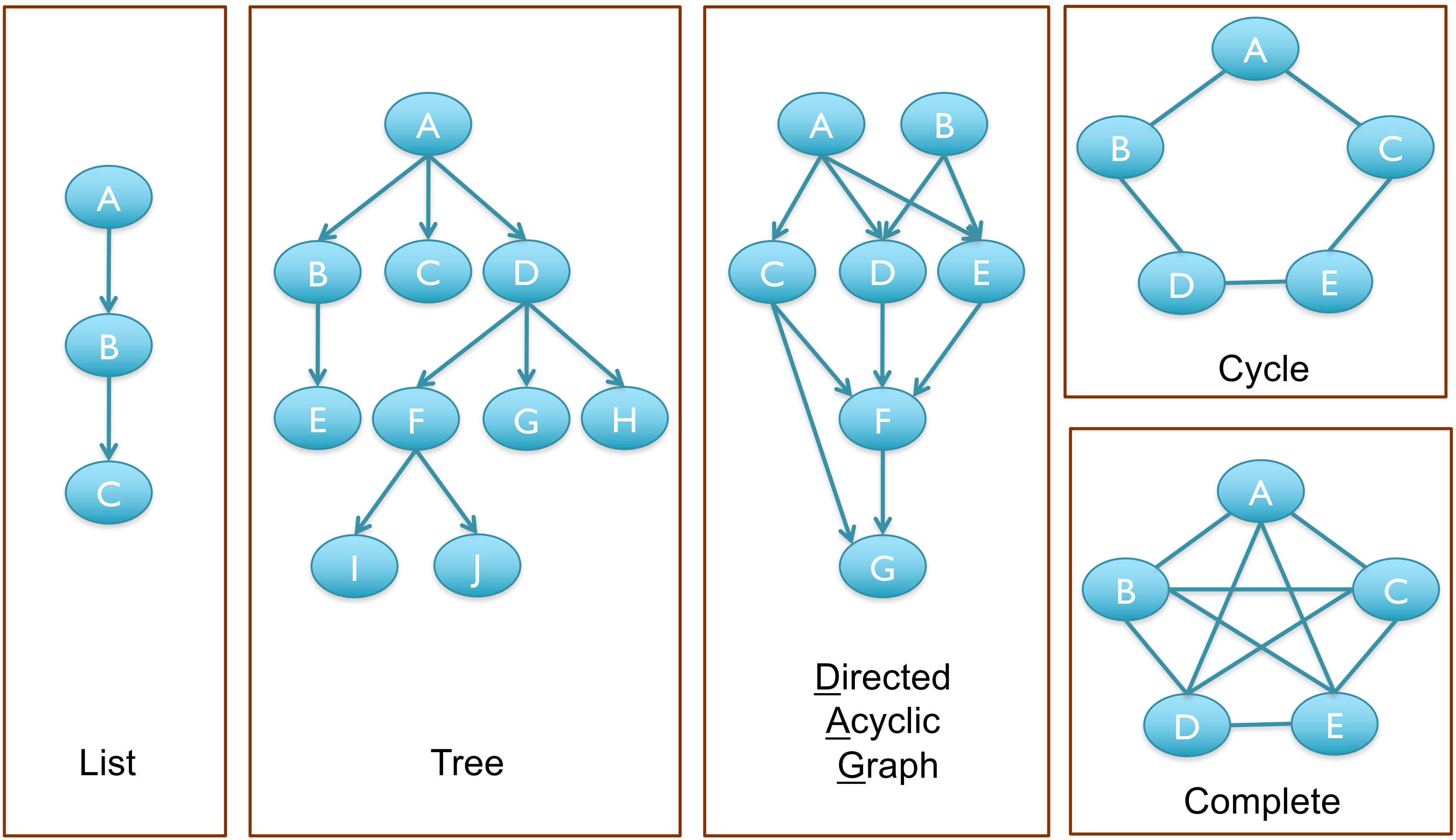
Computers in a network, Friends on Facebook, Roads & Cities on GoogleMaps, Webpages on Internet, Cells in your body, …

Graphs

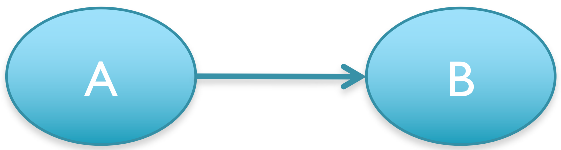


- Nodes aka vertices
  - People, Proteins, Cities, Genes, Neurons, Sequences, Numbers, …
- Edges aka arcs
  - A is connected to B
  - A is related to B
  - A regulates B
  - A precedes B
  - A interacts with B
  - A activates B
  - …

Graph Types

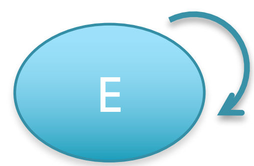

A
B
C
List
A
B
C
D
E
F
G
H
I
J
Tree
A
B
C
D
E
F
G
Directed
Acyclic
Graph
A
B
C
D
E
Cycle
A
B
C
D
E
Complete

Definitions (I)



Directed Edge

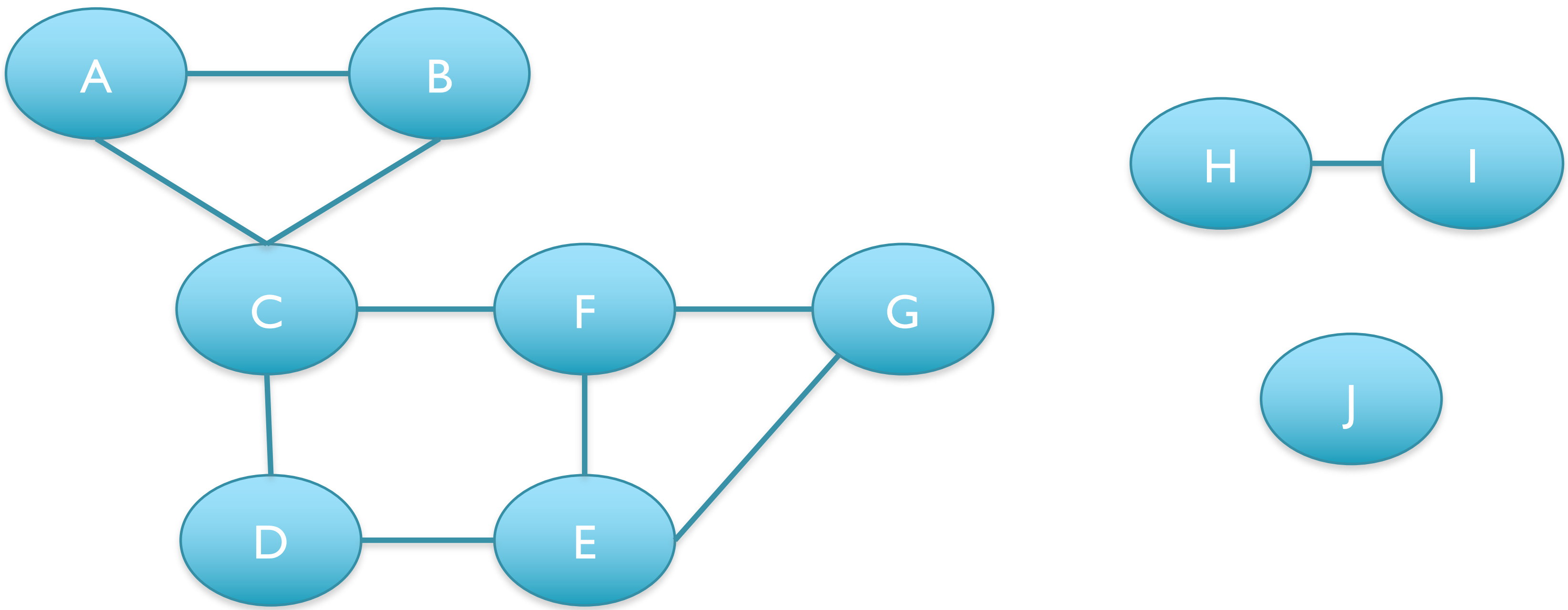
Undirected Edge



Self Edge
(Unusual but usually allowed)

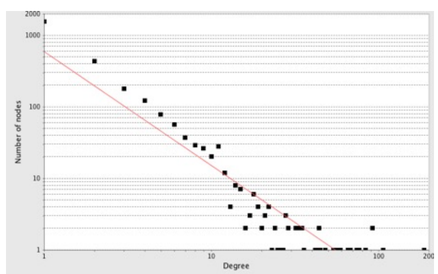
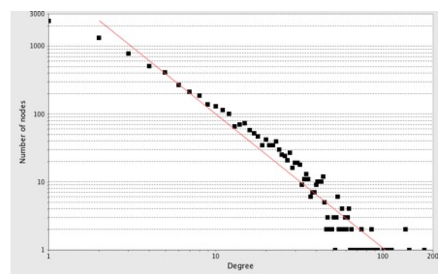
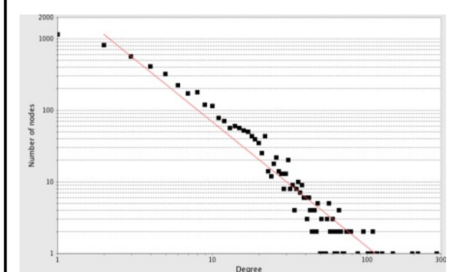
- A and B are ***adjacent***
- An edge connected to a vertex is ***incident*** on that vertex
- The number of edges incident on a vertex is the ***degree*** of that vertex
    - For directed graphs, separately report ***indegree*** and ***outdegree***
- A ***multigraph*** allows multiple edges between the same pair of nodes, a ***simple*** graph does not allow multiple edges (most common)

Definitions (2)



- A ***path*** is a sequence of edges e_1, e_2, … e_n in which each edge starts from the vertex the previous edge ended at
- A path that starts and ends at the same node is a ***cycle***
- The number of edges in a path is called the ***length*** of the path
- A graph is ***connected*** if there is a path between every pair of nodes, otherwise it is ***disconnected*** into >1 ***connected components***

Network Characteristics

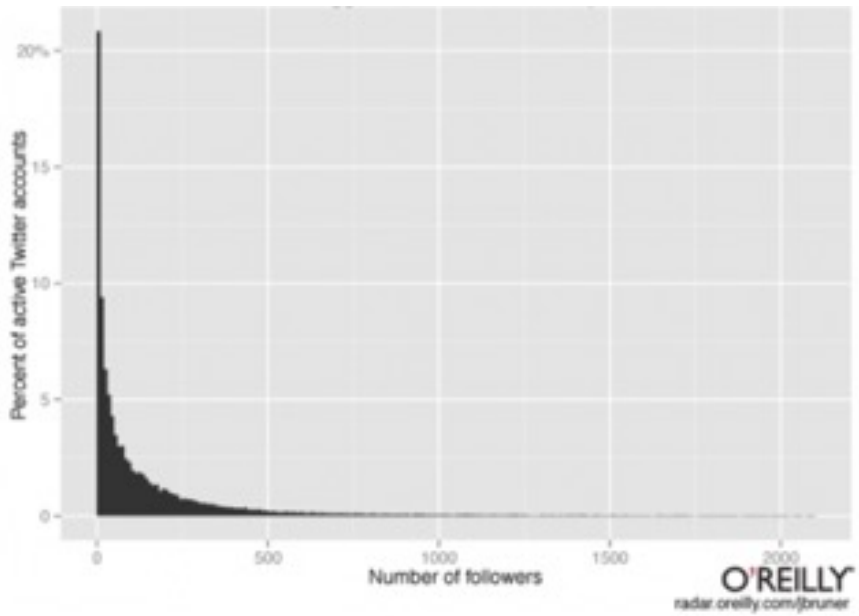
	C. elegans	*D. melanogaster*	*S. cerevisiae*
# Nodes	2646	7464	4965
# Edges	4037	22831	17536
Avg. / Max Degree	3.0 / 187	6.1 / 178	7.0 / 283
# Components	109	66	32
Largest Component	2386	7335	4906
Diameter	14	12	11
Avg. Shortest Path	4.8	4.4	4.1
Degree Distributions			

Diameter: Maximum length of shortest path between two nodes

Scale Free: Power law distribution of degree => Avg. Shortest Path between nodes is small

(Most people have ~100 twitter followers, but some have >1M)

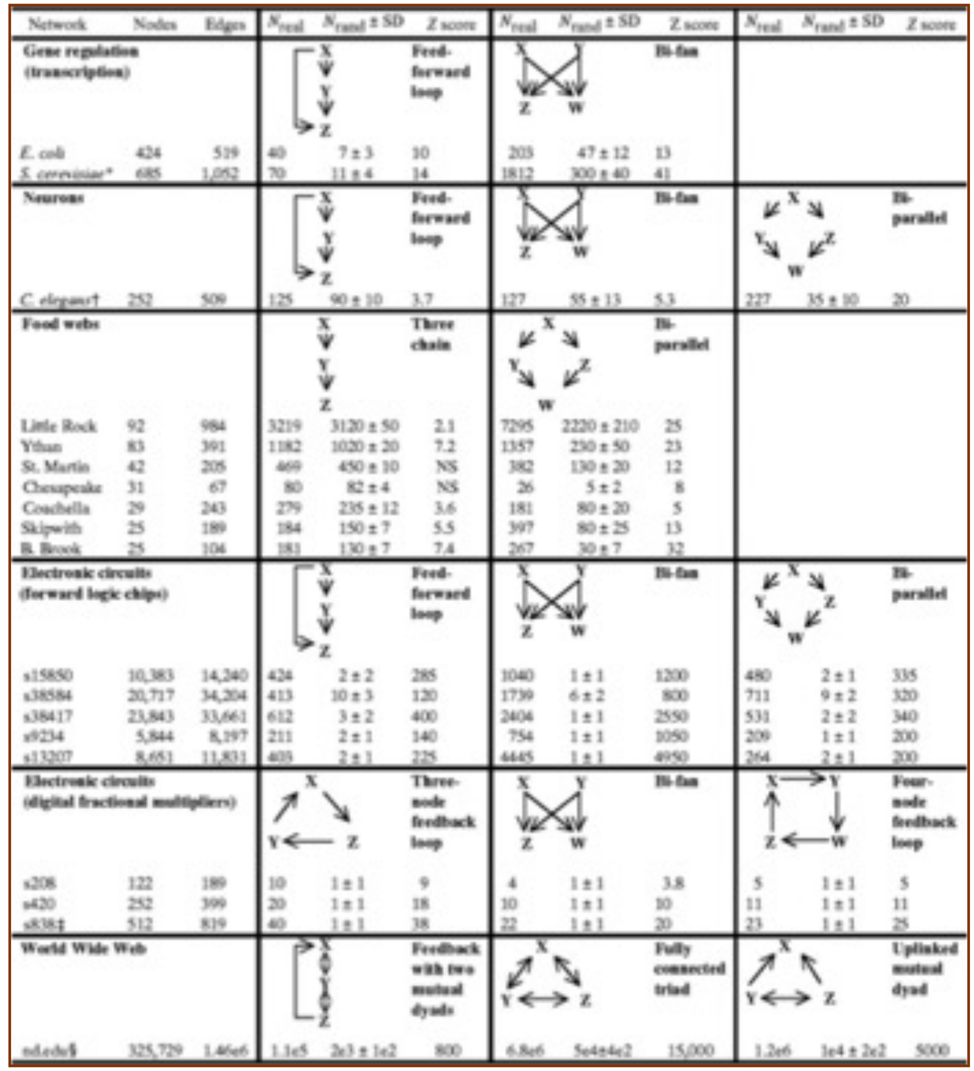
Network Characteristics



Diameter: Maximum length of shortest path between two nodes
Scale Free: Power law distribution of degree => Avg. Shortest Path between nodes is small
(Most people have ~100 twitter followers, but some have >1M)

Network Motifs

- Network Motif
  - Simple graph of connections
  - Exhaustively enumerate all possible 1, 2, 3, … k node motifs
- Statistical Significance
  - Compare frequency of a particular network motif in a real network as compared to a randomized network
- Certain motifs are “characteristic features” of the network



Network	Nodes	Edges	N_{real}	N_{rand} ± SD	Z score	N_{real}	N_{rand} ± SD	Z score	N_{real}	N_{rand} ± SD	Z score
Gene regulation (transcription)					**Feed-forward loop**			**Bi-fan**			
E. coli	424	519	40	7 ± 3	10	203	47 ± 12	13			
*S. cerevisiae**	685	1,052	70	11 ± 4	14	1812	300 ± 40	41			
Neurons					**Feed-forward loop**			**Bi-fan**			**Bi-parallel**
C. elegans†	252	509	125	90 ± 10	3.7	127	55 ± 13	5.3	227	35 ± 10	20
Food webs					**Three chain**			**Bi-parallel**			
Little Rock	92	984	3219	3120 ± 50	2.1	7295	2220 ± 210	25			
Ythan	83	391	1182	1020 ± 20	7.2	1357	230 ± 50	23			
St. Martin	42	205	469	450 ± 10	NS	382	130 ± 20	12			
Chesapeake	31	67	80	82 ± 4	NS	26	5 ± 2	8			
Coachella	29	243	279	235 ± 12	3.6	181	80 ± 20	5			
Skipwith	25	189	184	150 ± 7	5.5	397	80 ± 25	13			
B. Brook	25	104	181	130 ± 7	7.4	267	30 ± 7	32			
Electronic circuits (forward logic chips)					**Feed-forward loop**			**Bi-fan**			**Bi-parallel**
s15850	10,383	14,240	424	2 ± 2	285	1040	1 ± 1	1200	480	2 ± 1	335
s38584	20,717	34,204	413	10 ± 3	120	1739	6 ± 2	800	711	9 ± 2	320
s38417	23,843	33,661	612	3 ± 2	400	2404	1 ± 1	2550	531	2 ± 2	340
s9234	5,844	8,197	211	2 ± 1	140	754	1 ± 1	1050	209	1 ± 1	200
s13207	8,651	11,831	403	2 ± 1	225	4445	1 ± 1	4950	264	2 ± 1	200
Electronic circuits (digital fractional multipliers)					**Three-node feedback loop**			**Bi-fan**			**Four-node feedback loop**
s208	122	189	10	1 ± 1	9	4	1 ± 1	3.8	5	1 ± 1	5
s420	252	399	20	1 ± 1	18	10	1 ± 1	10	11	1 ± 1	11
s838‡	512	819	40	1 ± 1	38	22	1 ± 1	20	23	1 ± 1	25
World Wide Web					**Feedback with two mutual dyads**			**Fully connected triad**			**Uplinked mutual dyad**
nd.edu§	325,729	1.46e6	1.1e5	2e3 ± 1e2	800	6.8e6	5e4±4e2	15,000	1.2e6	1e4 ± 2e2	5000

Network Motifs: Simple Building Blocks of Complex Networks
Milo et al (2002) *Science. 298:824-827*

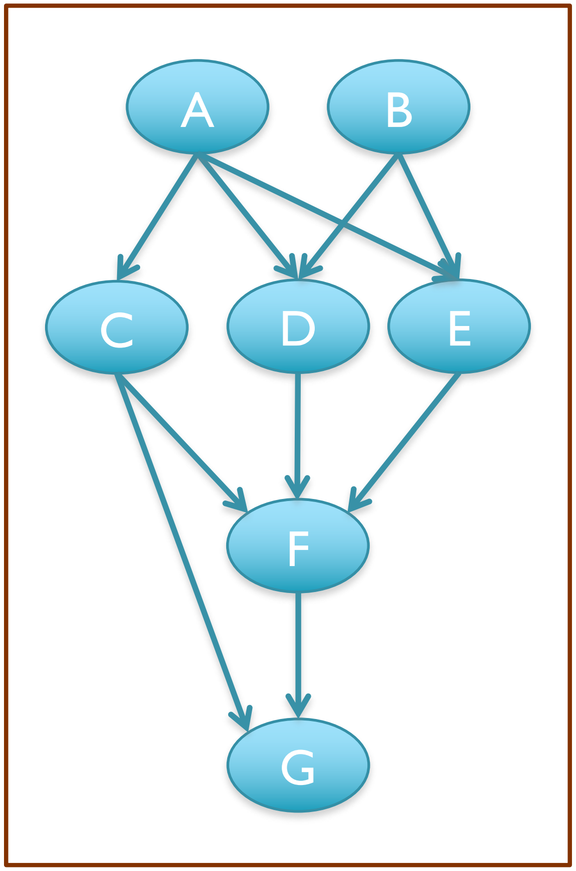
Graph Interface

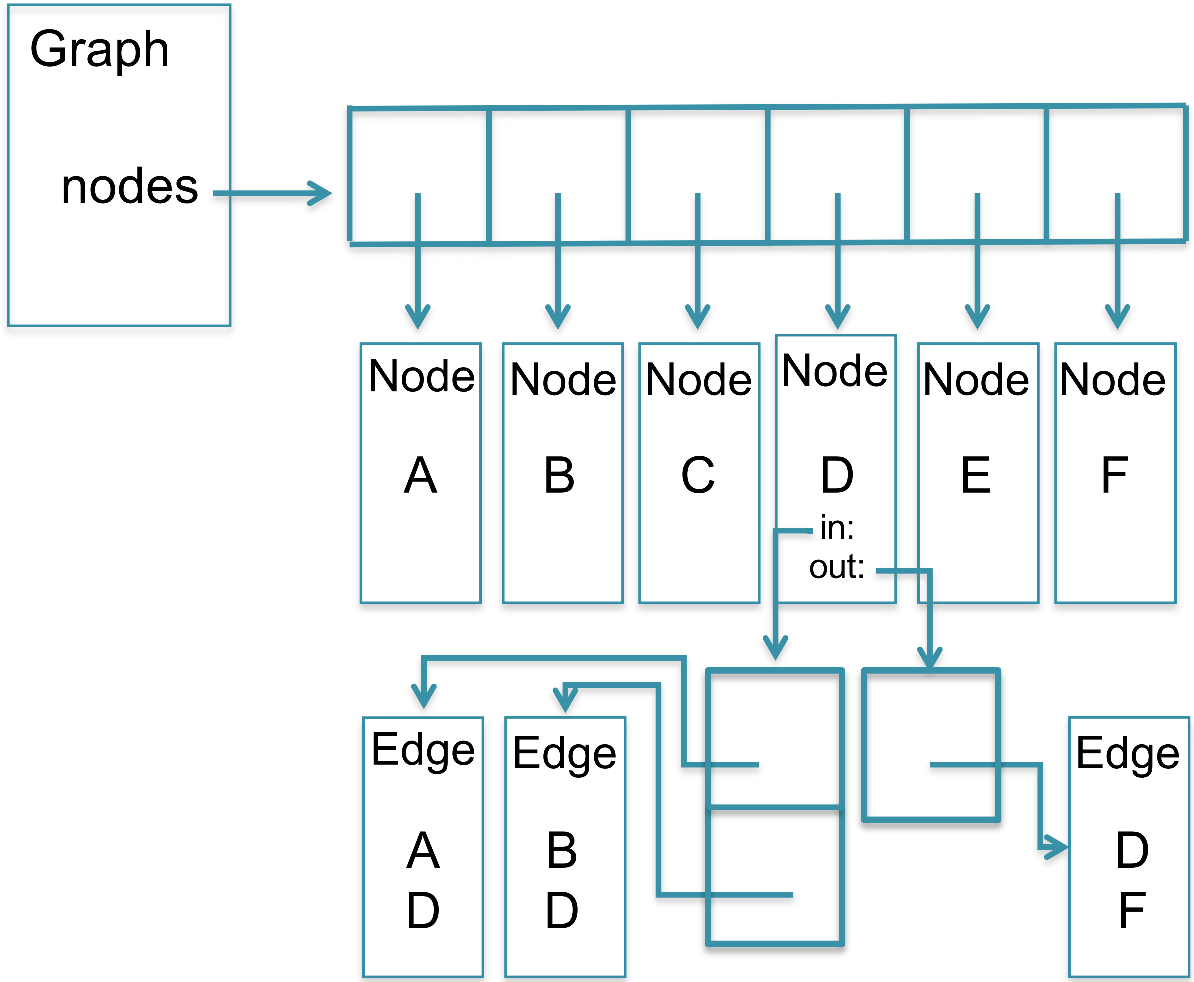
Separate generic types for vertices <V> and edges <E>

```
public interface Graph<V,E> {
    ...
    Position<V> insertVertex(V v);

    Position<E> insertEdge(Position<V> from, Position<V> to, E e)
            throws InvalidPositionException, InsertionException;

    V removeVertex(Position<V> p)
            throws InvalidPositionException, RemovalException;

    E removeEdge(Position<E> p)
                              throws InvalidPositionException;

    Iterable<Position<V>> vertices();

    Iterable<Position<E>> edges();

    Iterable<Position<E>> incomingEdges(Position<V> p)
                              throws InvalidPositionException;

    Iterable<Position<E>> outgoingEdges(Position<V> p)
                              throws InvalidPositionException;
}
```

Can iterate through all the nodes OR iterate through all the edges as different algorithms may require one or the other

Representing Graphs





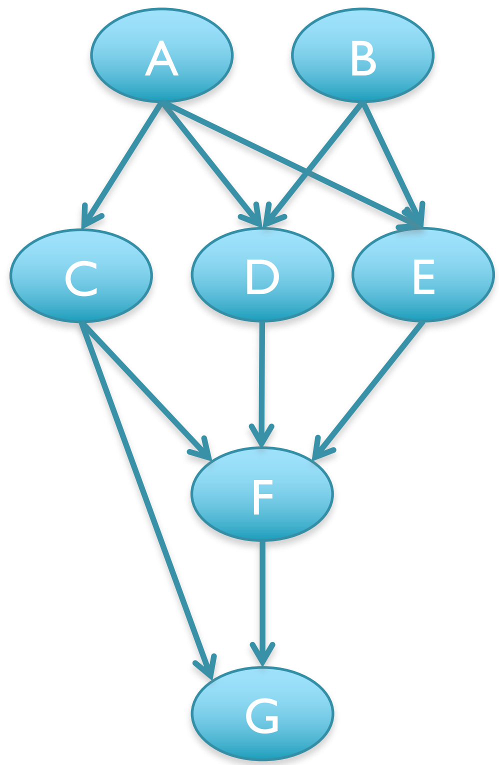
Incidence List

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!

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:

Complexity Analysis

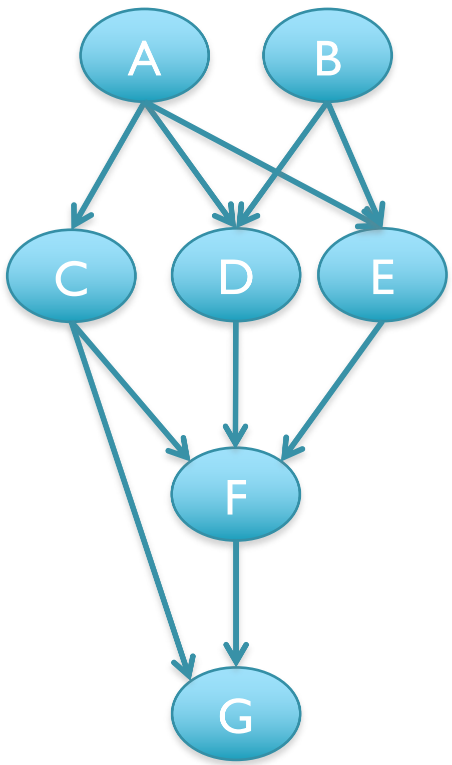
If n is the number of vertices, and m is the number of edges, we need O(n + m) space to represent the graph

When we insert a vertex, allocate one object and two empty edge lists: O(1)

When we insert an edge we allocate one object and insert the edge into appropriate lists for the incident vertices: O(1)

Remove a node?	O(1); Only after edges removed
Remove an edge?	O(d) where d is max degree; O(n) worse case
Find/check edge between nodes?	O(d); O(n) worst case

Representing Graphs



Adjacency Matrix
Good for dense graphs
Fast, Fixed storage: N^2 bits or N^2 weights

	A	B	C	D	E	F	G
A			1	1	1		
B				1	1		
C						1	1
D						1	
E						1	
F							1
G							

Incidence List
Good for sparse graphs
Compact storage: ~8 bytes/edge

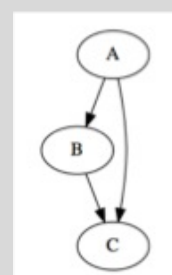
A: C, D, E
B: D, E
C: F, G
D: F
E: F
G:

Edge List
Easy, good if you (mostly) need to iterate through the edges
~16 bytes / edge

A,C
A,D
A,E
B,C
B,D
B,E
C,F
C,G
D,F
E,F
F,G

Tools
Graphviz: http://www.graphviz.org/
Gephi: https://gephi.org/
Cytoscape: http://www.cytoscape.org/

```
digraph G {
    A->B
    B->C
    A->C
}
$ dot -Tpdf -o g.pdf g.dot
```



Graph Interface

Separate generic types for vertices <V> and edges <E>

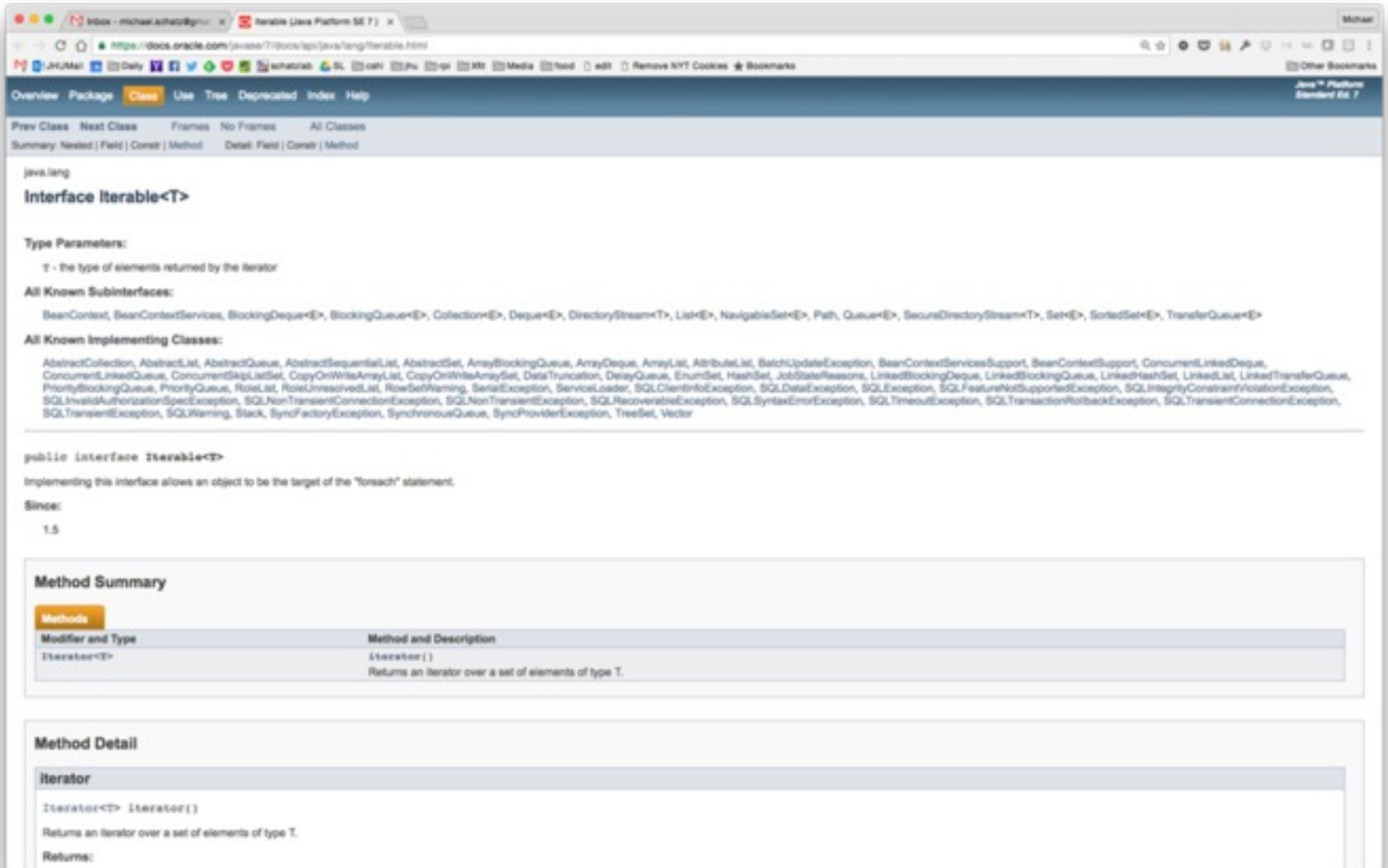
```
public interface Graph<V,E> {
    ...
    Position<V> insertVertex(V v);

    Position<E> insertEdge(Position<V> from, Position<V> to, E e)
            throws InvalidPositionException, InsertionException;

    V removeVertex(Position<V> p)
            throws InvalidPositionException, RemovalException;

    E removeEdge(Position<E> p)
                                throws InvalidPositionException;

    Iterable<Position<V>> vertices();

    Iterable<Position<E>> edges();

    Iterable<Position<E>> incomingEdges(Position<V> p)
                                throws InvalidPositionException;

    Iterable<Position<E>> outgoingEdges(Position<V> p)
                               throws InvalidPositionException;
    ...
}
```

Graph Interface

Separate generic types for vertices <V> and edges <E>

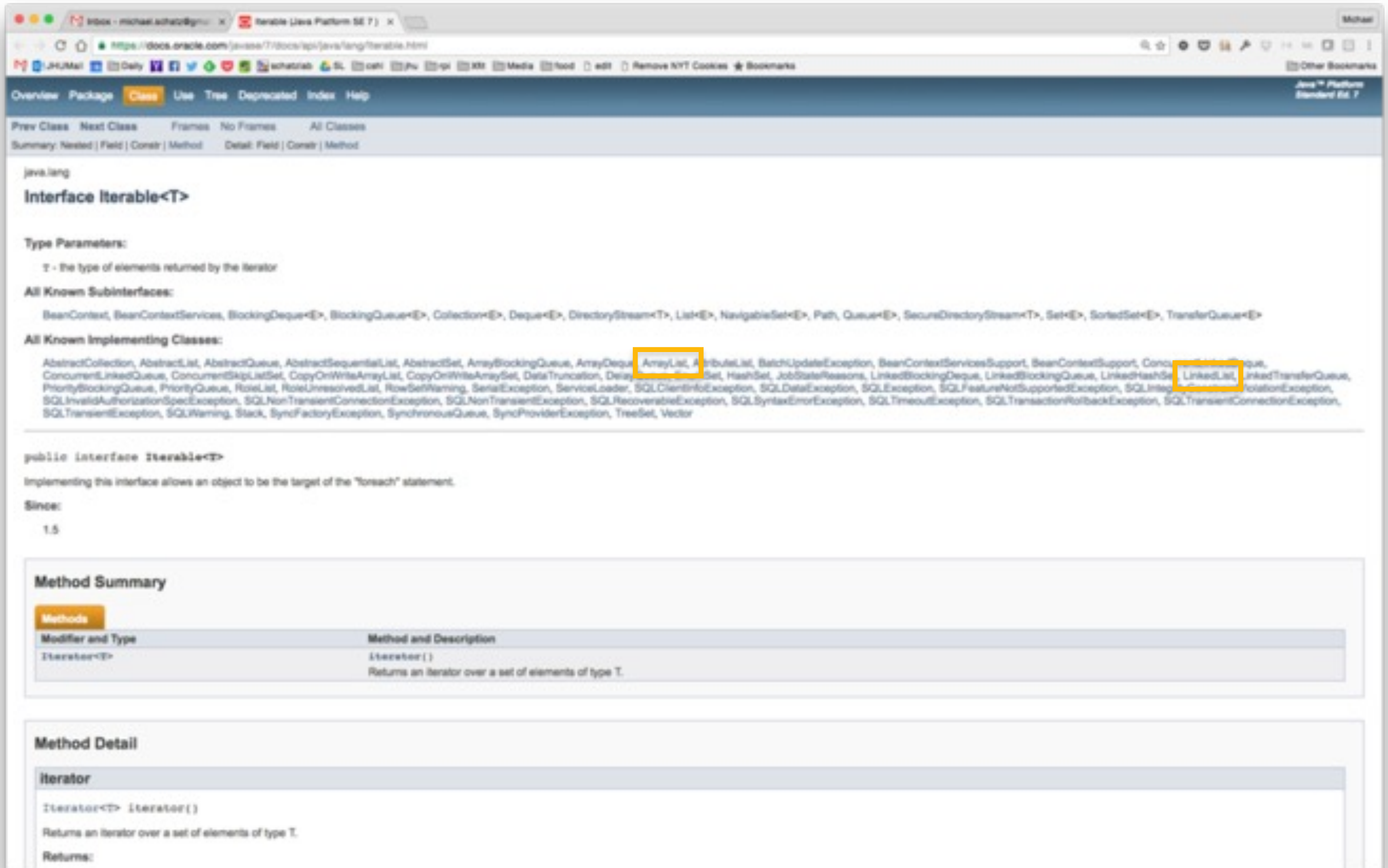
```
public interface Graph<V,E> {
    ...
    Position<V> insertVertex(V v);

    Position<E> insertEdge(Position<V> from, Position<V> to, E e)
            throws InvalidPositionException, InsertionException;

    V removeVertex(Position<V> p)
            throws InvalidPositionException, RemovalException;

    E removeEdge(Position<E> p)
                               throws InvalidPositionException;

    Iterable<Position<V>> vertices();

    Iterable<Position<E>> edges();

    Iterable<Position<E>> incomingEdges(Position<V> p)
                               throws InvalidPositionException;

    Iterable<Position<E>> outgoingEdges(Position<V> p)
                               throws InvalidPositionException;
    ...
}
```

Iterable



Overview Package Class Use Tree Deprecated Index Help
Prev Class Next Class Frames No Frames All Classes
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method
java.lang
Interface Iterable<T>
Type Parameters:
T - the type of elements returned by the iterator
All Known Subinterfaces:
BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Collection<E>, Deque<E>, DirectoryStream<T>, List<E>, NavigableSet<E>, Path, Queue<E>, SecureDirectoryStream<T>, Set<E>, SortedSet<E>, TransferQueue<E>
All Known Implementing Classes:
AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BatchUpdateException, BeanContextServicesSupport, BeanContextSupport, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DataTruncation, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, RowSetWarning, SerialException, ServiceLoader, SQLClientInfoException, SQLDataException, SQLException, SQLFeatureNotSupportedException, SQLIntegrityConstraintViolationException, SQLInvalidAuthorizationSpecException, SQLNonTransientConnectionException, SQLNonTransientException, SQLRecoverableException, SQLSyntaxErrorException, SQLTimeoutException, SQLTransactionRollbackException, SQLTransientConnectionException, SQLTransientException, SQLWarning, Stack, SyncFactoryException, SynchronousQueue, SyncProviderException, TreeSet, Vector
public interface Iterable<T>
Implementing this interface allows an object to be the target of the "foreach" statement.
Since:
1.5
Method Summary
Methods
Modifier and Type
Method and Description
Iterator<T>
iterator()
Returns an iterator over a set of elements of type T.
Method Detail
iterator
Iterator<T> iterator()
Returns an iterator over a set of elements of type T.
Returns:

Iterable


Overview Package Class Use Tree Deprecated Index Help
Prev Class Next Class Frames No Frames All Classes
Summary: Nested | Field | Constr | Method Detail: Field | Constr | Method
java.lang
Interface Iterable<T>
Type Parameters:
T - the type of elements returned by the iterator
All Known Subinterfaces:
BeanContext, BeanContextServices, BlockingDeque<E>, BlockingQueue<E>, Collection<E>, Deque<E>, DirectoryStream<T>, List<E>, NavigableSet<E>, Path, Queue<E>, SecureDirectoryStream<T>, Set<E>, SortedSet<E>, TransferQueue<E>
All Known Implementing Classes:
AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BatchUpdateException, BeanContextServicesSupport, BeanContextSupport, ConcurrentLinkedDeque, ConcurrentLinkedQueue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DataTruncation, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList, LinkedTransferQueue, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, RowSetWarning, SerialException, ServiceLoader, SQLClientInfoException, SQLDataException, SQLException, SQLFeatureNotSupportedException, SQLIntegrityConstraintViolationException, SQLInvalidAuthorizationSpecException, SQLNonTransientConnectionException, SQLNonTransientException, SQLRecoverableException, SQLSyntaxErrorException, SQLTimeoutException, SQLTransactionRollbackException, SQLTransientConnectionException, SQLTransientException, SQLWarning, Stack, SyncFactoryException, SynchronousQueue, SyncProviderException, TreeSet, Vector
public interface Iterable<T>
Implementing this interface allows an object to be the target of the "foreach" statement.
Since:
1.5
Method Summary
Methods
Modifier and Type
Method and Description
Iterator<T>
iterator()
Returns an iterator over a set of elements of type T.
Method Detail
iterator
Iterator<T> iterator()
Returns an iterator over a set of elements of type T.
Returns:

Graph Interface

Separate generic types for vertices <V> and edges <E>

```
public interface Graph<V,E> {
    ...
    Position<V> insertVertex(V v);

    Position<E> insertEdge(Position<V> from, Position<V> to, E e)
            throws InvalidPositionException, InsertionException;
...
```

What will insertEdge into Graph<String, Integer> return?

Position<Integer>

What will insertVertex into Graph<Integer, Integer> return?

Position<Integer>

:-(

Graph Interface 2

```
public interface Edge<T> extends Position<T> {}
public interface Vertex<T> extends Position<T> {}

public interface Graph<V,E> {
    ...
    Vertex<V> insertVertex(V v);

    Edge<E> insertEdge(Vertex<V> from, Vertex<V> to, E e)
            throws InvalidVertexException, InsertionException;

    V removeVertex(Vertex<V> p)
            throws InvalidVertexException, RemovalException;

    E removeEdge(Edge<E> p)
                                  throws InvalidEdgeException;

    Iterable<Vertex<V>> vertices();

    Iterable<Edge<E>> edges();

    ...
}
```

Now clients can check at compile time if their types are correct

What else is missing from the interface?

Graph Interface 3

```
public interface Graph<V,E> {
    ...
    Vertex<V> fromVertex(Edge<E> e) ...
    Vertex<V> toVertex(Edge<E> e) ...
    ...
}
```

Now clients can check their code to see where the edges go!

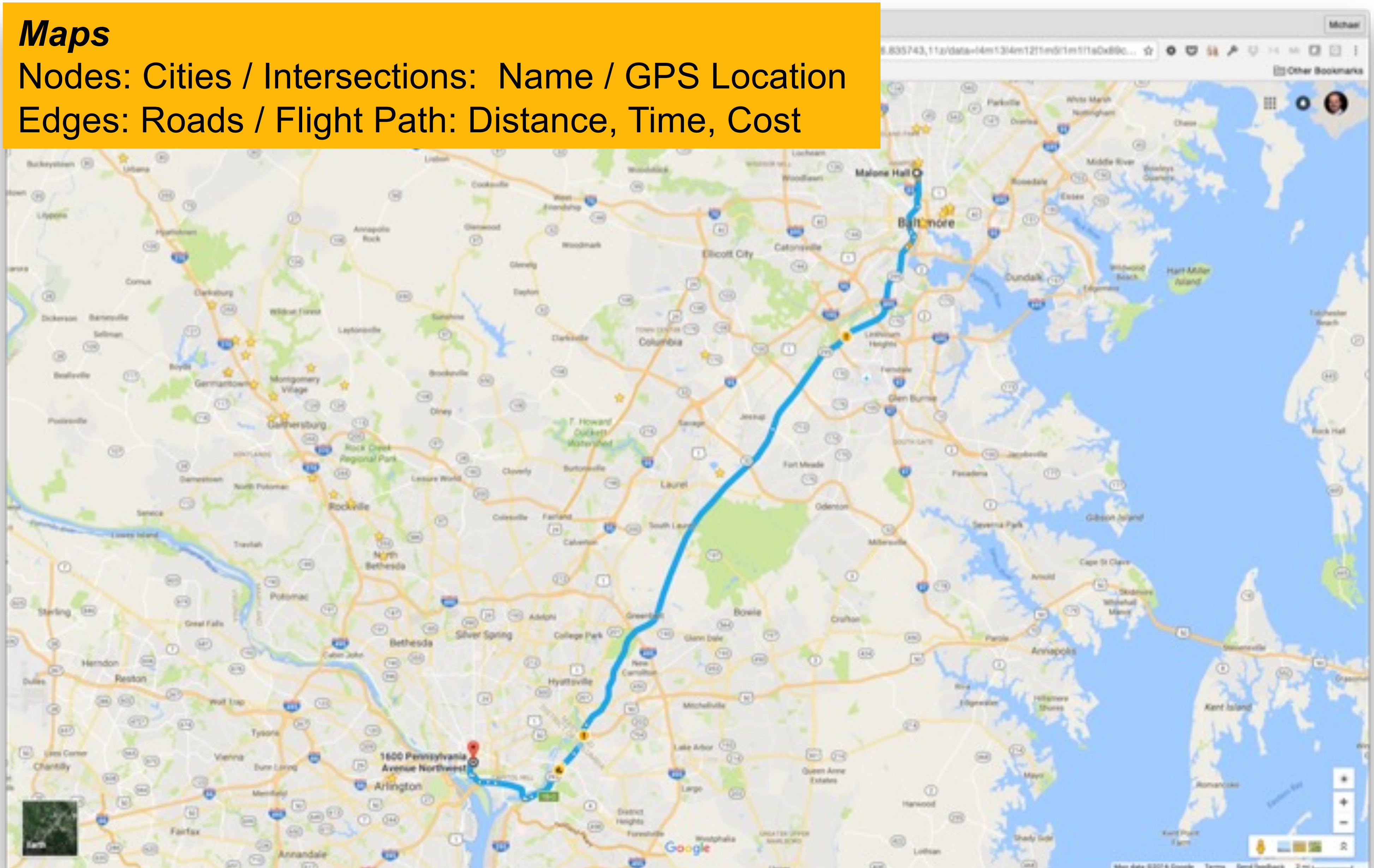
Don’t just define an interface, try to use it!

Graph Searching

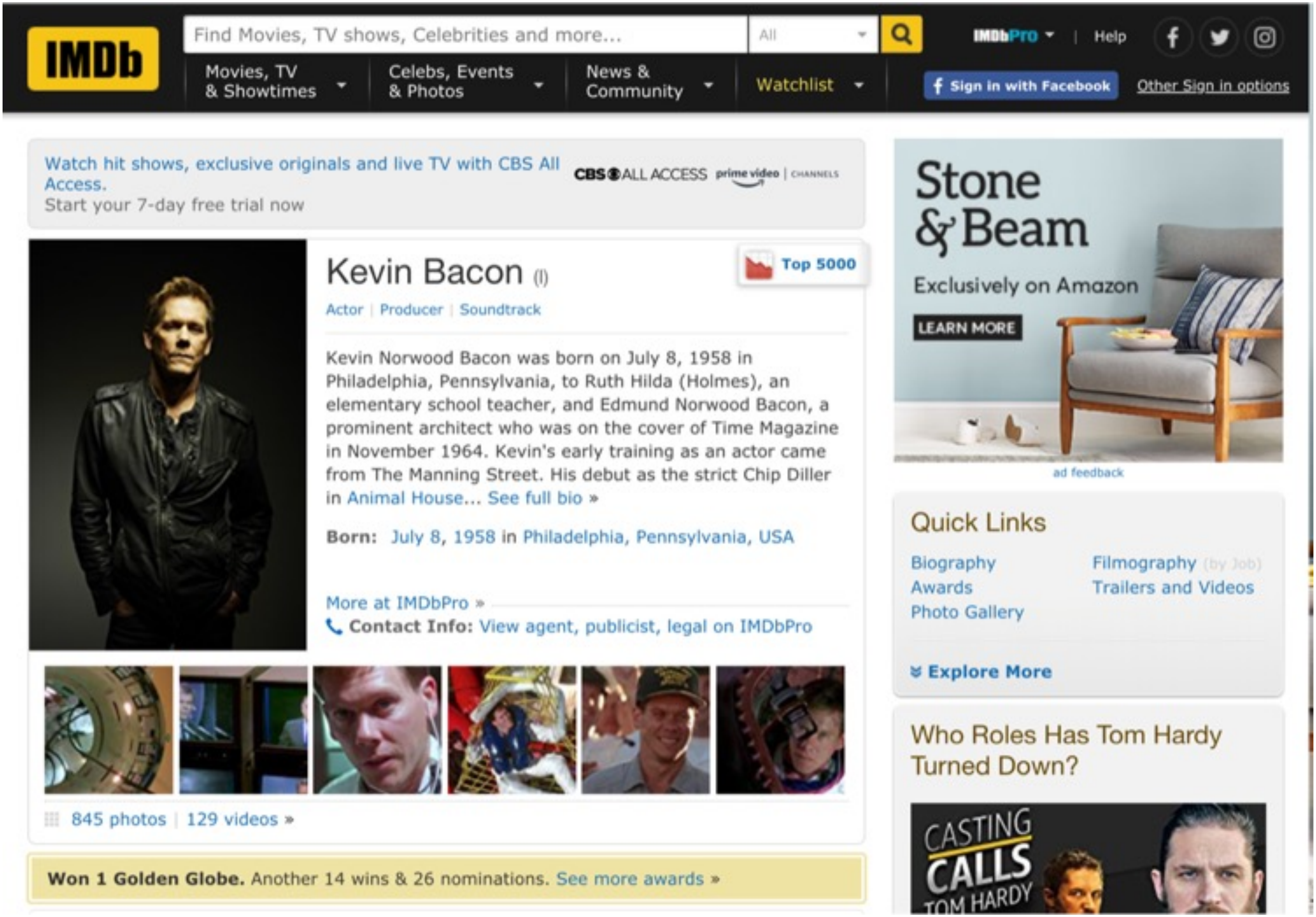
Maps

Nodes: Cities / Intersections: Name / GPS Location

Edges: Roads / Flight Path: Distance, Time, Cost



https://www.youtube.com/watch?v=Rmn-amJ9UA4

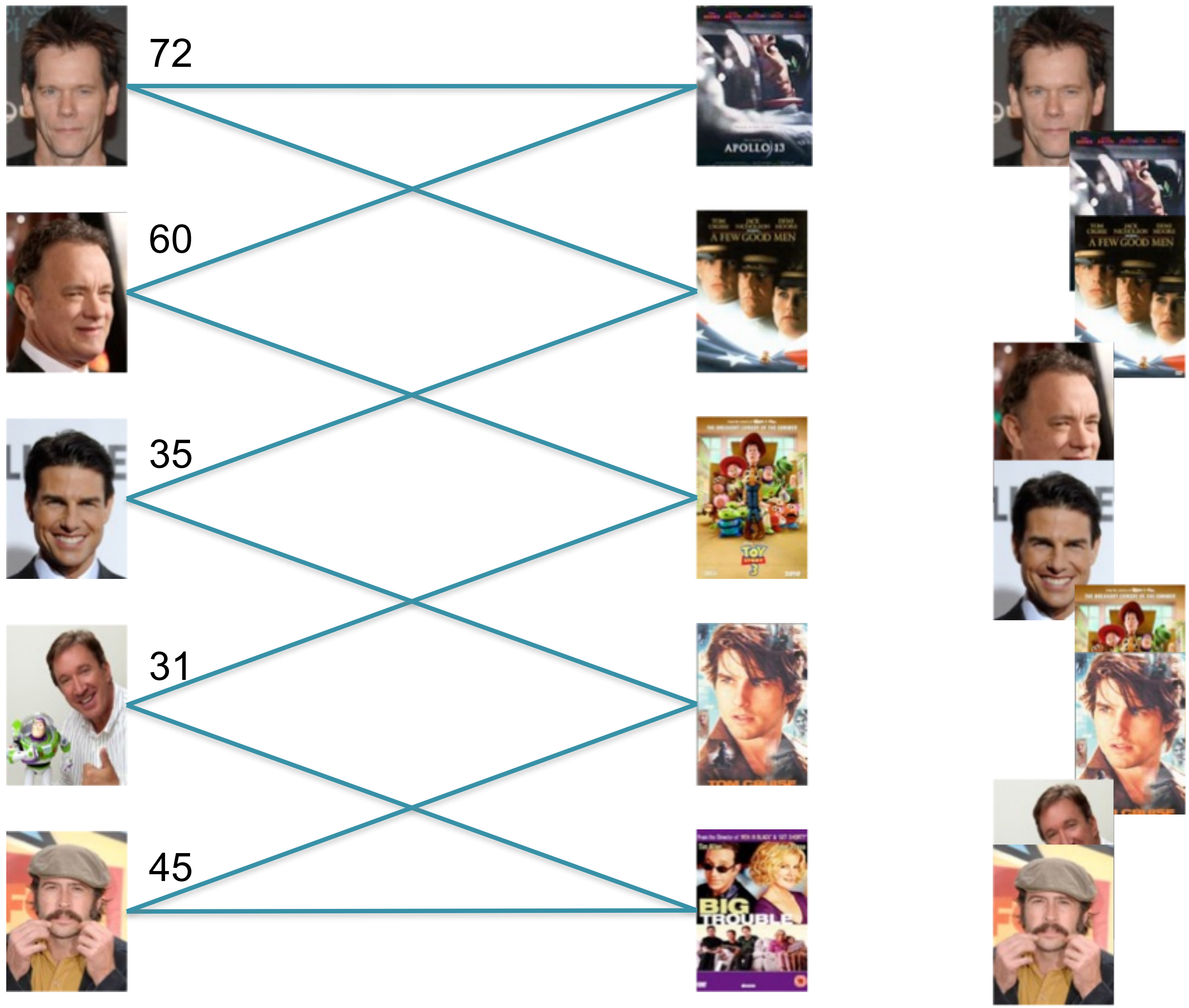

IMDb
Find Movies, TV shows, Celebrities and more...
All
IMDbPro
Help
Movies, TV & Showtimes
Celebs, Events & Photos
News & Community
Watchlist
Sign in with Facebook
Other Sign in options
Watch hit shows, exclusive originals and live TV with CBS All Access.
Start your 7-day free trial now
CBS ALL ACCESS prime video | CHANNELS
Kevin Bacon (I)
Top 5000
Actor | Producer | Soundtrack
Kevin Norwood Bacon was born on July 8, 1958 in Philadelphia, Pennsylvania, to Ruth Hilda (Holmes), an elementary school teacher, and Edmund Norwood Bacon, a prominent architect who was on the cover of Time Magazine in November 1964. Kevin's early training as an actor came from The Manning Street. His debut as the strict Chip Diller in Animal House... See full bio »
Born: July 8, 1958 in Philadelphia, Pennsylvania, USA
More at IMDbPro »
Contact Info: View agent, publicist, legal on IMDbPro
845 photos | 129 videos »
Won 1 Golden Globe. Another 14 wins & 26 nominations. See more awards »
Stone & Beam
Exclusively on Amazon
LEARN MORE
ad feedback
Quick Links
Biography
Awards
Photo Gallery
Filmography (by Job)
Trailers and Videos
Explore More
Who Roles Has Tom Hardy Turned Down?
CASTING CALLS
TOM HARDY

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)
```

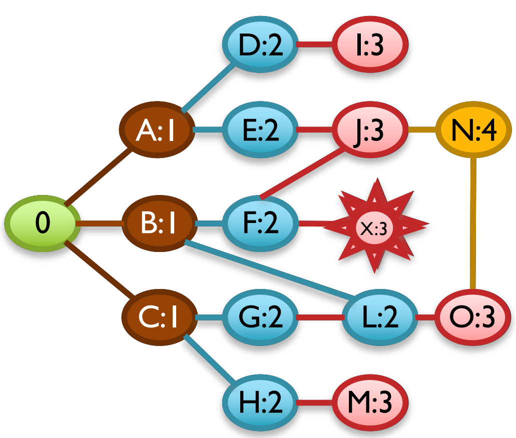
0

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

```
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)
```

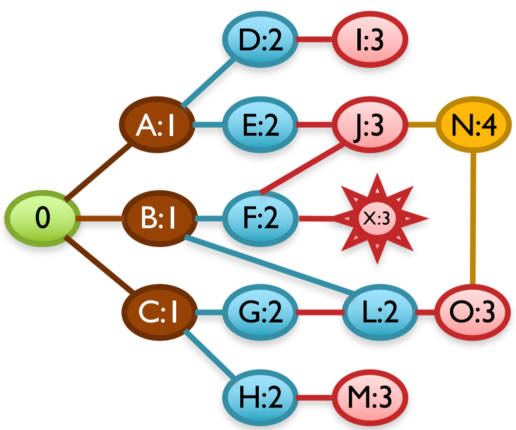
0

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



DFS

```
DFS(start, stop)
// initialize all nodes dist = -1
start.dist = 0
list.addEnd(start)
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)
```

0

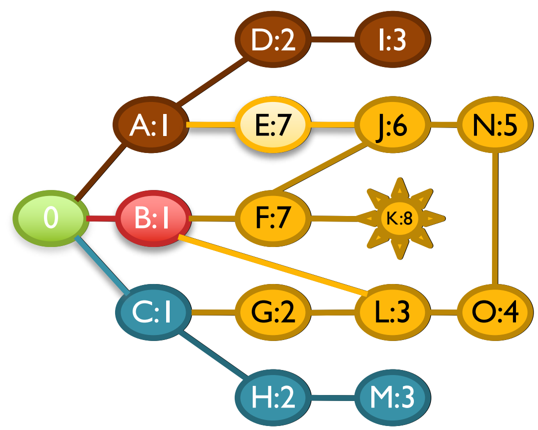
A,B,C

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

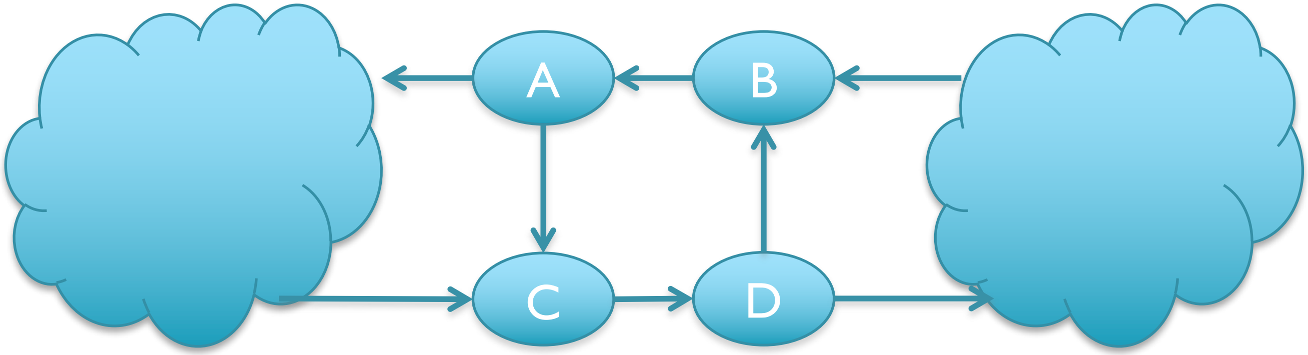
A,B,G
A,B,L
A,B,O
A,B,N
A,B,J
A,B,E,F
A,B,E,K
A,B,E

A,B

A
D
I

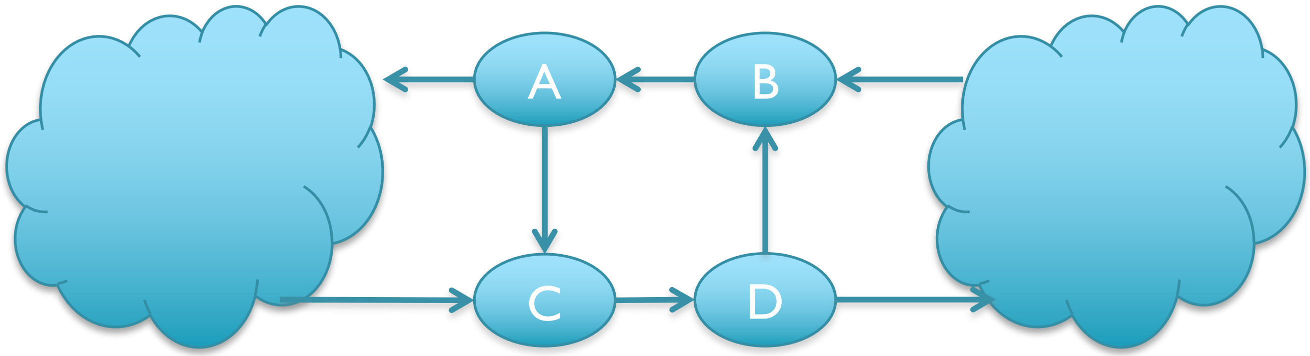


Graph Interface 4




```
public interface Graph<V,E> {
    ...
    boolean marked(Vertex<V> v);
    boolean marked(Edge<E> e);
    void mark(Vertex<V> v);
    void mark(Edge<E> e);
    void clearMarks();
    ...
}
```

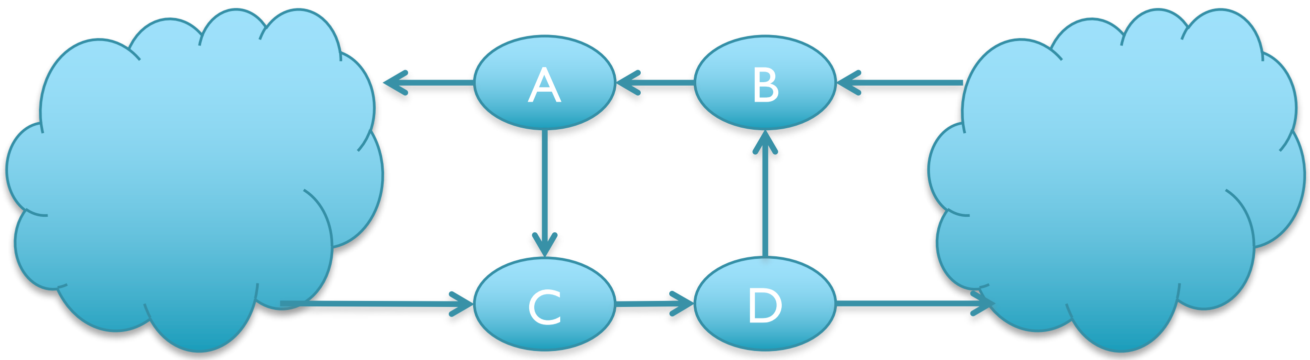
Graph Interface 5




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

More flexible, but client will have to use a consistent type for all labels

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

Note use of overloading: compiler will figure out which version you meant based on parameters passed on. Good for simple, closely related methods

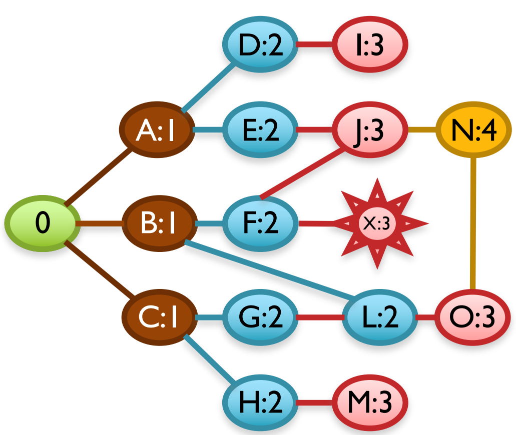
BFS: Queue

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



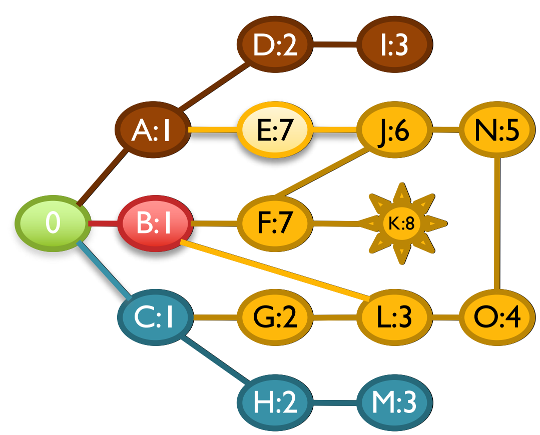
DFS: Stack

What is the runtime complexity?

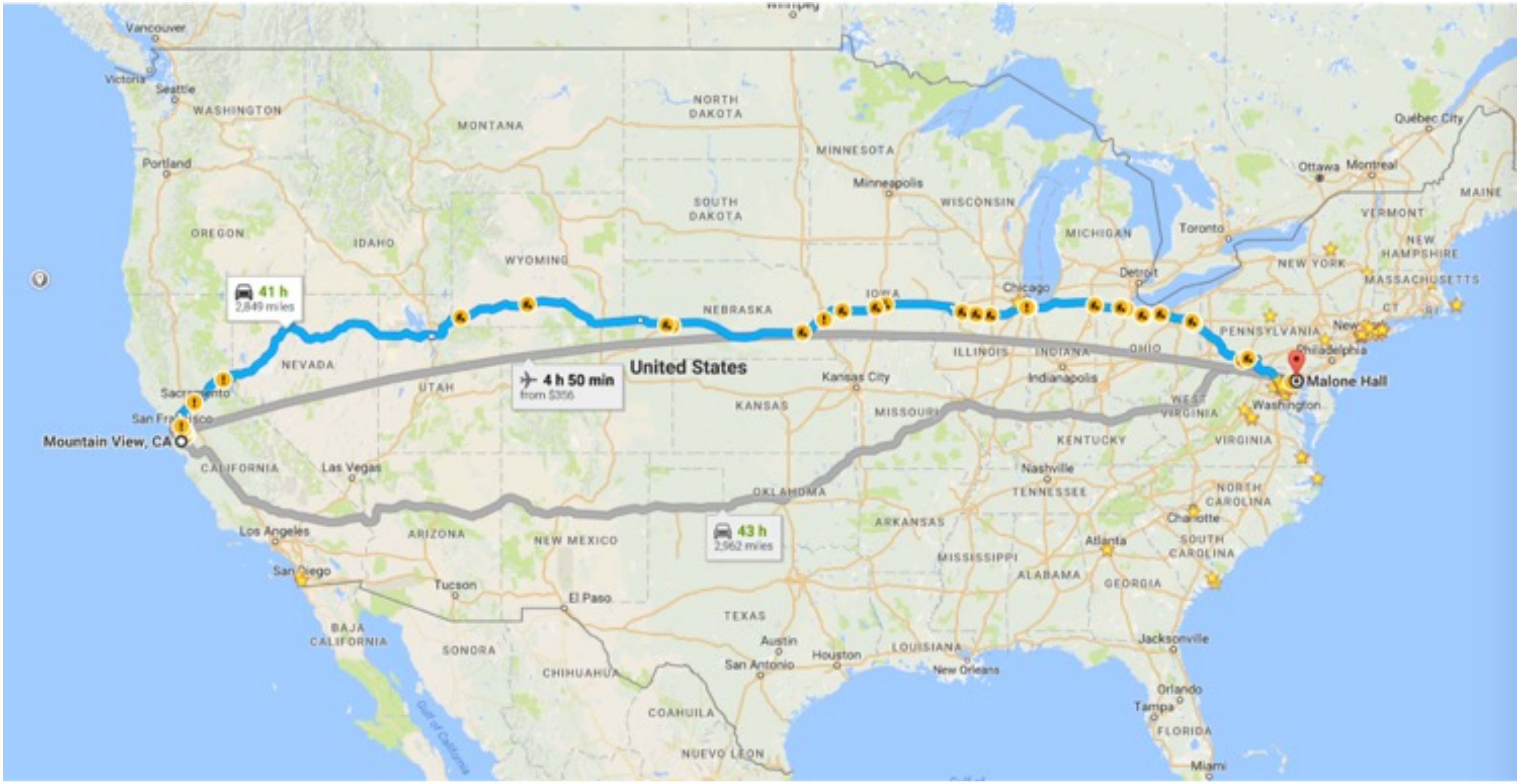
What is the space complexity?

```
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)
```

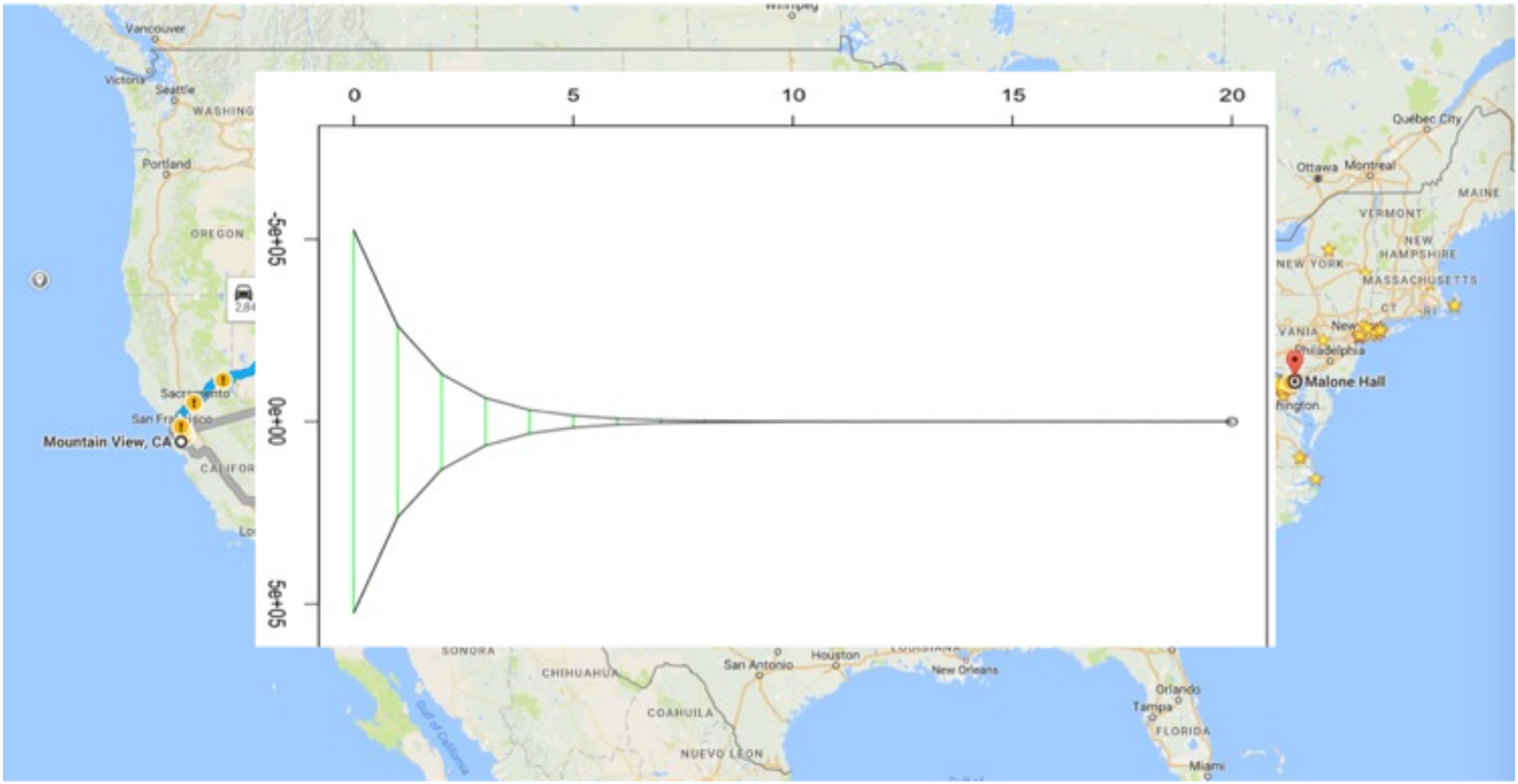
A,B,C
A,B,G,H
A,B,G,M
A,B,G
A,B,L
A,B,O
A,B,N
A,B,J
A,B,E,F
A,B,E,K
A,B,E
A,B
A
D
I



Breath First Searching

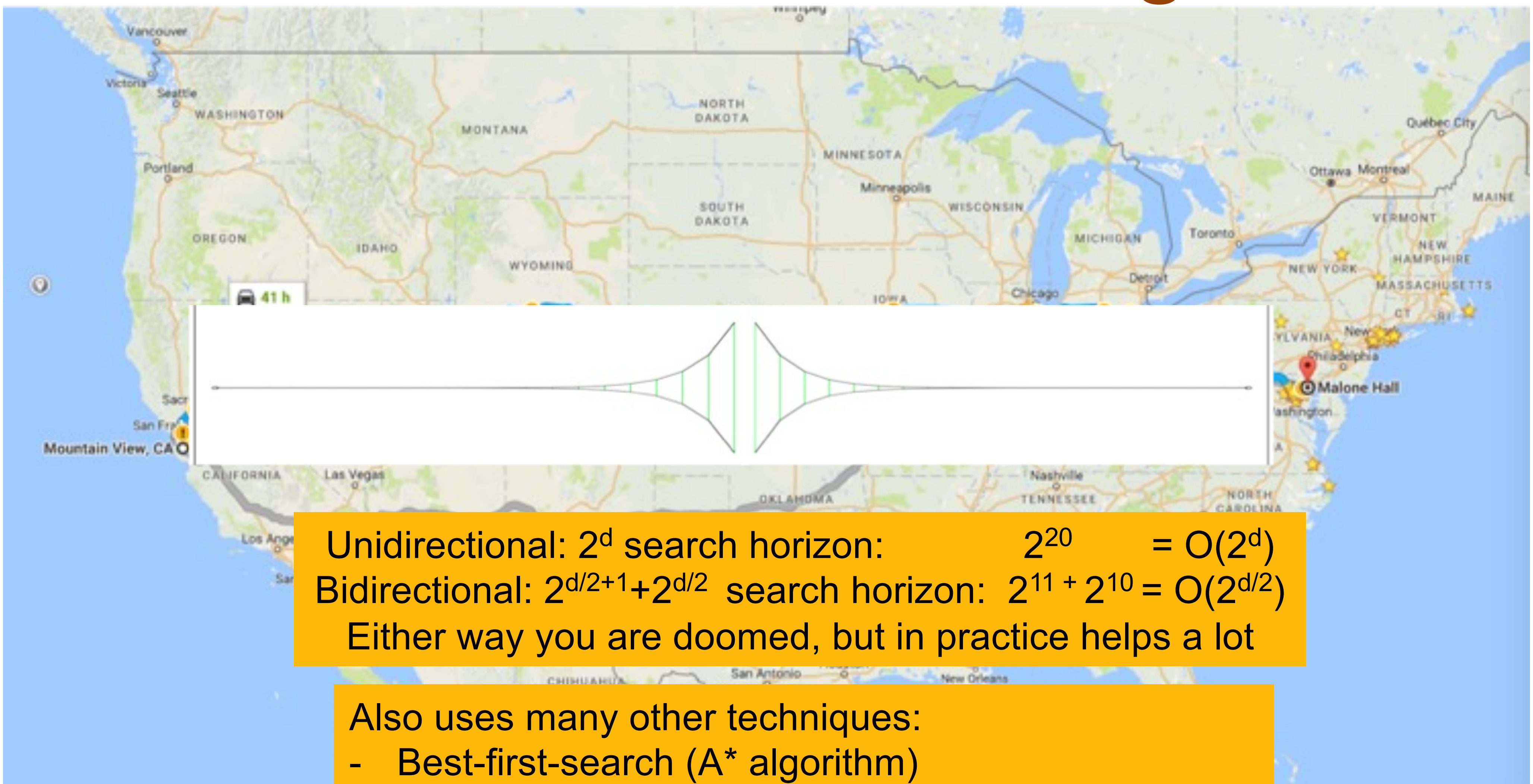


Breath First Searching



2^d search space: 2^{20}

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...

Next Steps

1. Work on HW4

2. Check on Piazza for tips & corrections!