CS 600.226: Data Structures

Michael Schatz

Oct 8 2018

Lecture 17. Machine Code Optimization

Agenda

1. ***Questions on HW4***
2. ***Recap on Graphs***
3. ***Machine Code Optimization***

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).

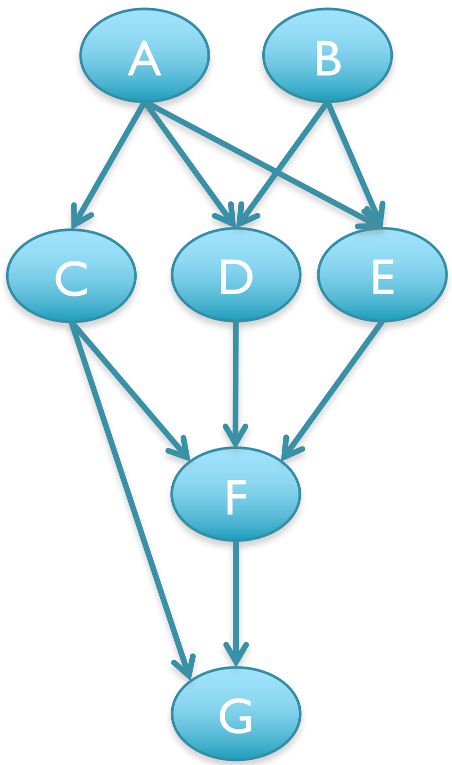
Agenda

1. ***Questions on HW4***
2. ***Recap on Graphs***
3. ***Machine Code Optimization***

Graphs are Everywhere!

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

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

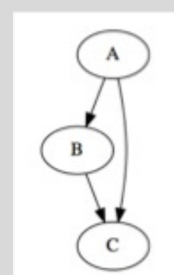
A: C, D, E
B: D, E
C: F, G
D: F
E: F
G:

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

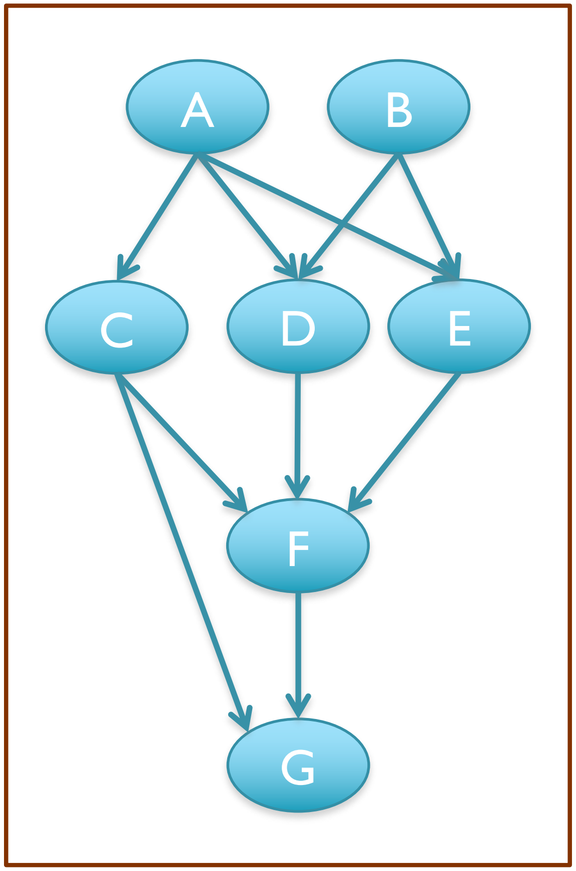
A,C
A,D
A,E
B,C
B,D
B,E
C,F
C,G
D,F
E,F
F,G

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
```



Representing Graphs

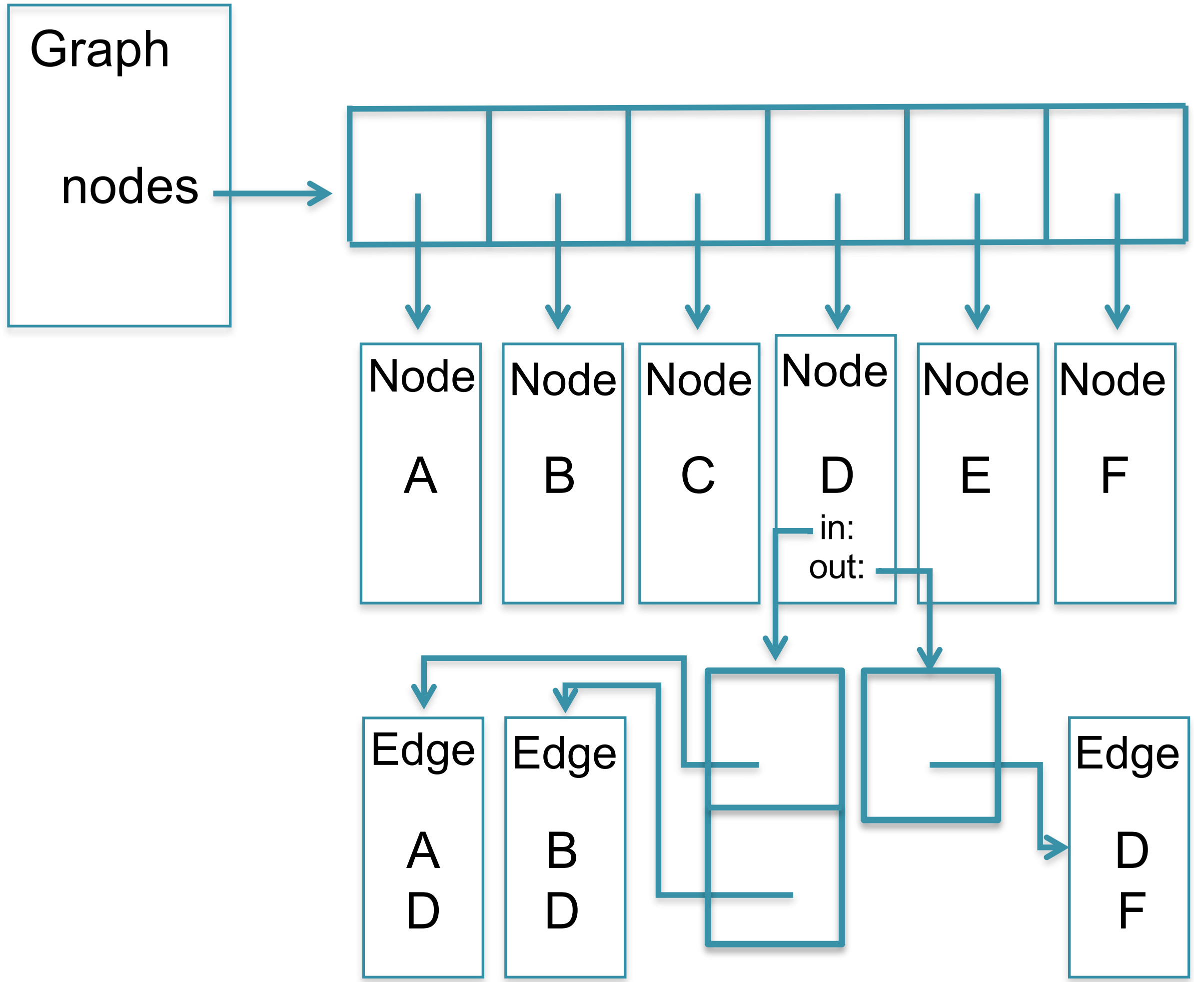


Incidence List

Good for sparse graphs

Compact storage

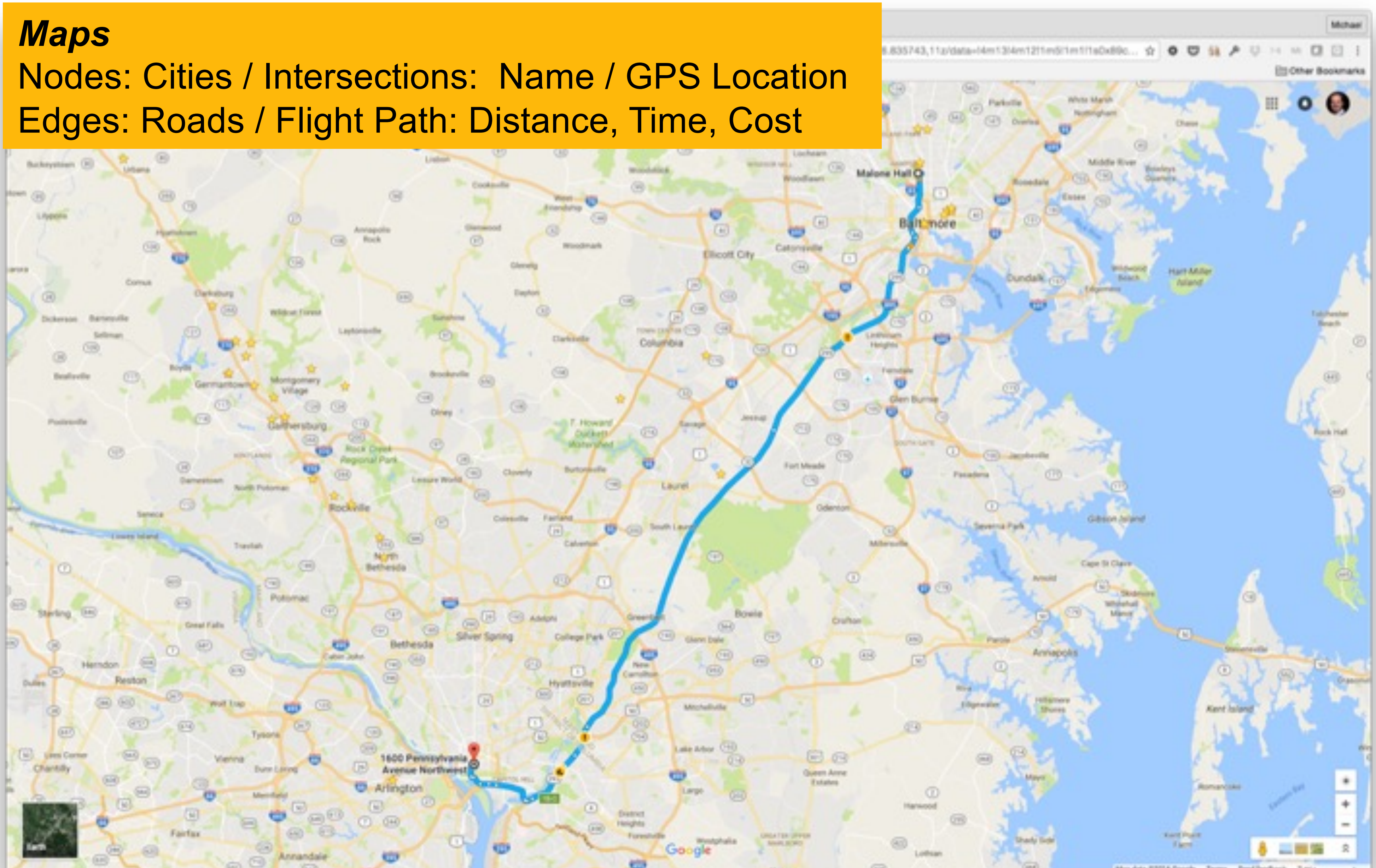
A: C, D, E
B: D, E
C: F, G
D: F
E: F
G:



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

Graph Searching

Maps
Nodes: Cities / Intersections: Name / GPS Location
Edges: Roads / Flight Path: Distance, Time, Cost



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)
```

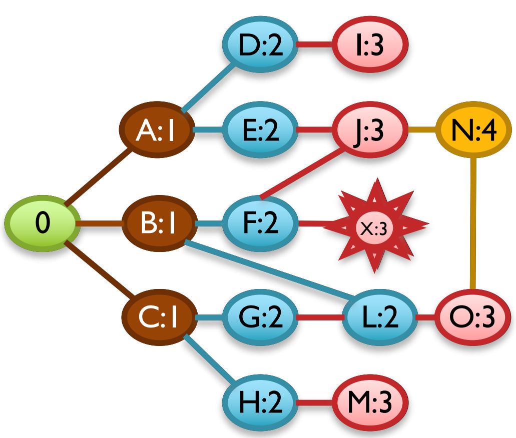
0

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)
```

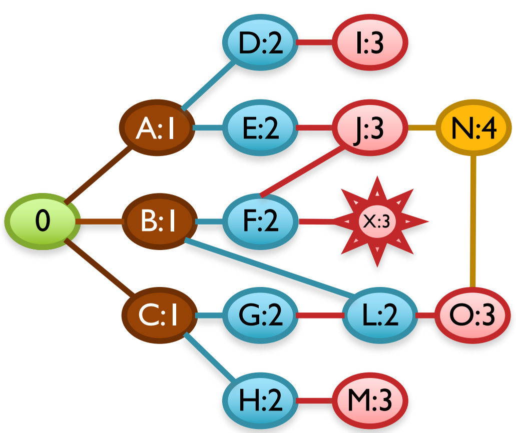
0

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

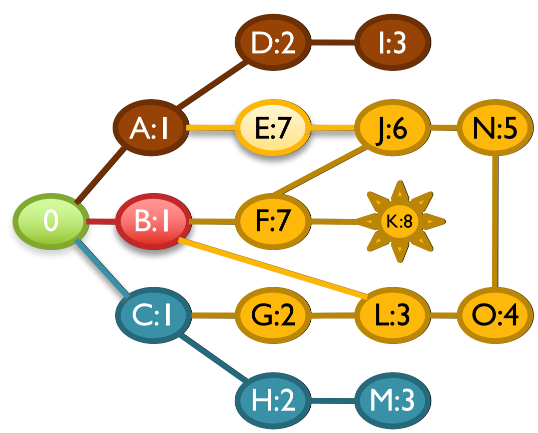
A,B,C

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



BFS: Queue

BF

What is the runtime complexity?

// i

sta

What is the space complexity?

list

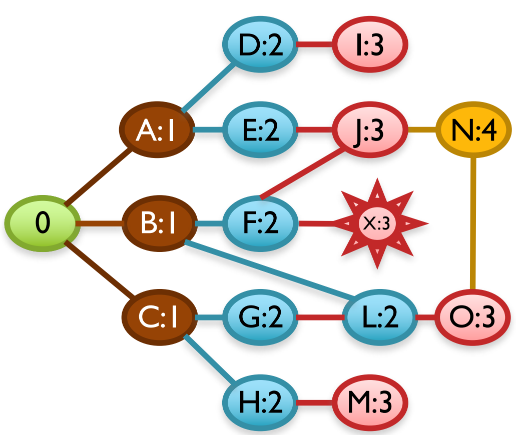
B,C,D,E
C,D,E,F,L

```
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)
```

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

DF

What is the runtime complexity?

// i

sta

C

What is the space complexity?

list.

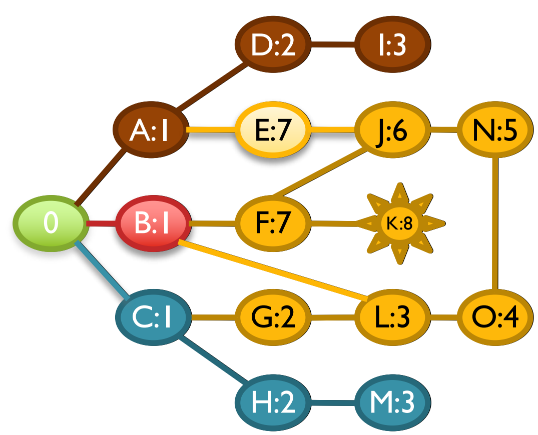
```
while (!list.empty())
  cur = list.end()
  if (cur == stop)
    print cur.dist;
  else
    foreach child in cur.children
      if (child.dist == -1)
        child.dist = cur.dist+1
        list.addEnd(child)
```

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

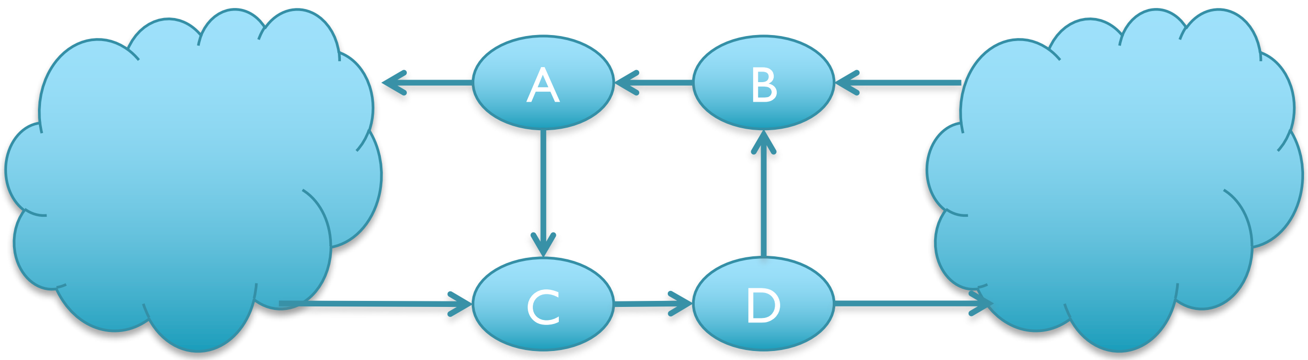
A,B,G
A,B,L
A,B,O
A,B,N
A,B,J
A,B,E,F
A,B,E,K
A,B,E

A,B

A
D
I



Graph Interface 6

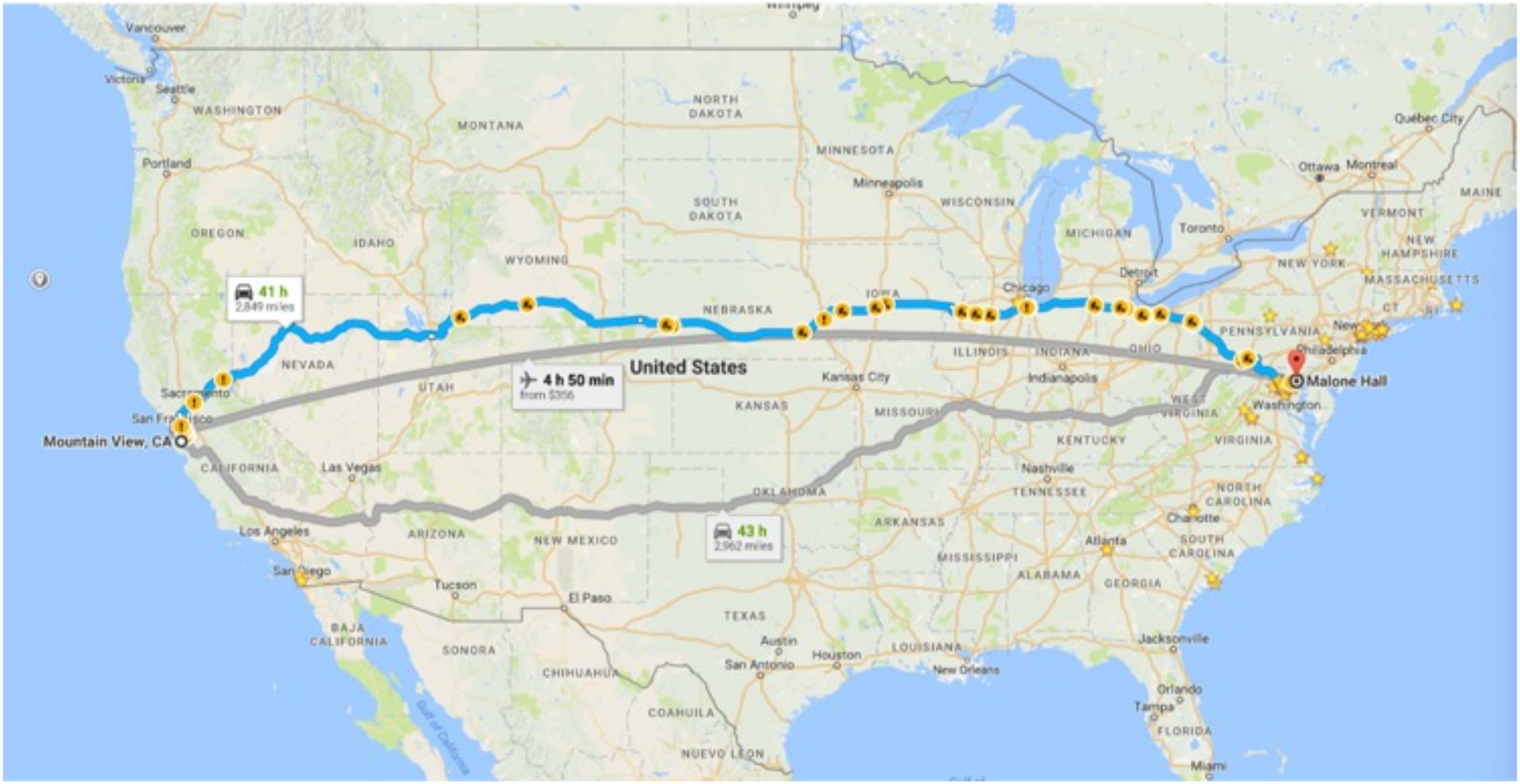



```
public interface Graph<V,E> {
    ...
    Object label(Vertex<V> v);
    Object label(Edge<E> e);
    void label(Vertex<V> v, Object label);
    void label(Edge<E> e, Object label);
    void clearLabels();
    ...
}
```

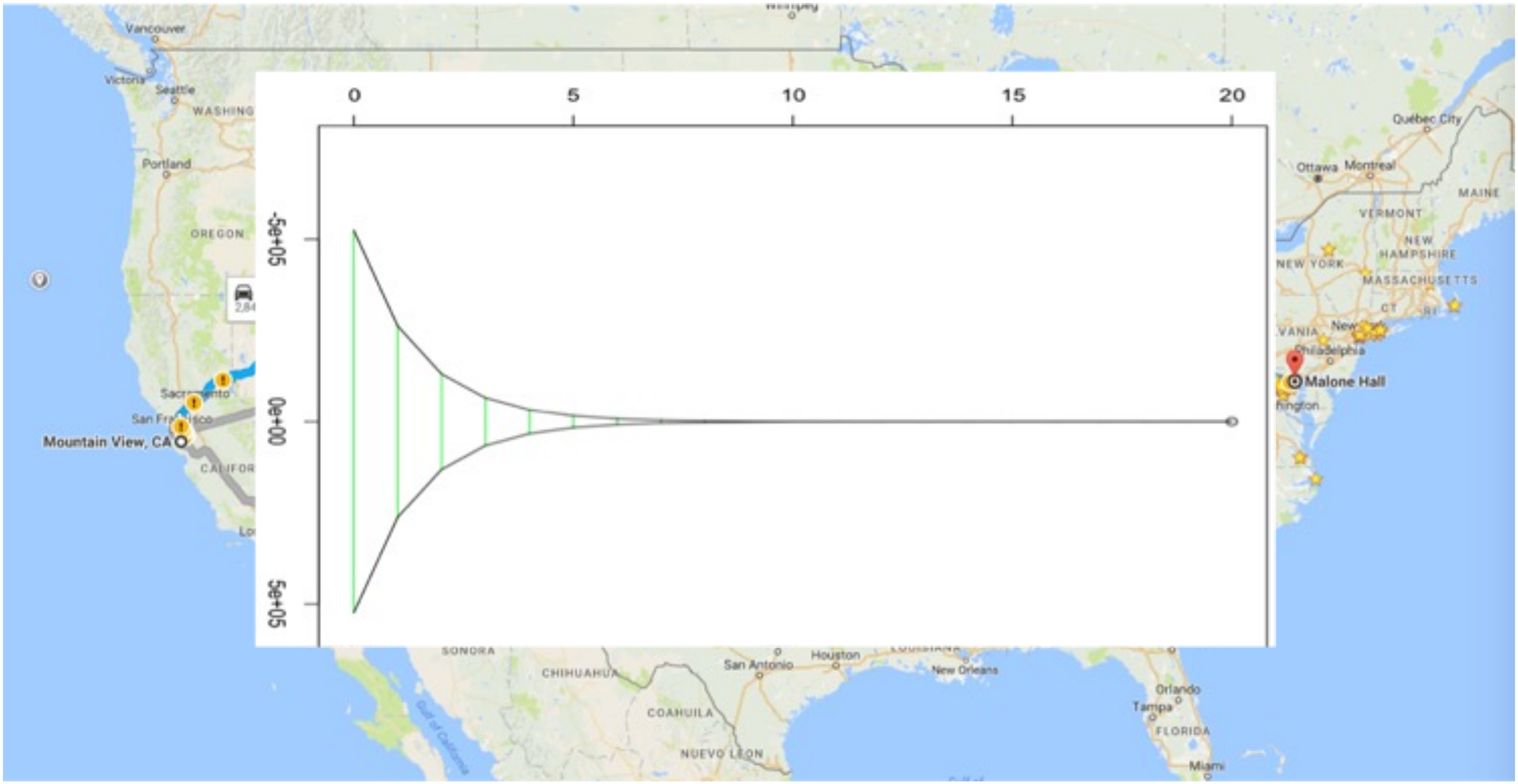
Very flexible, but client will have to cast Object to correct type

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

Breath First Searching

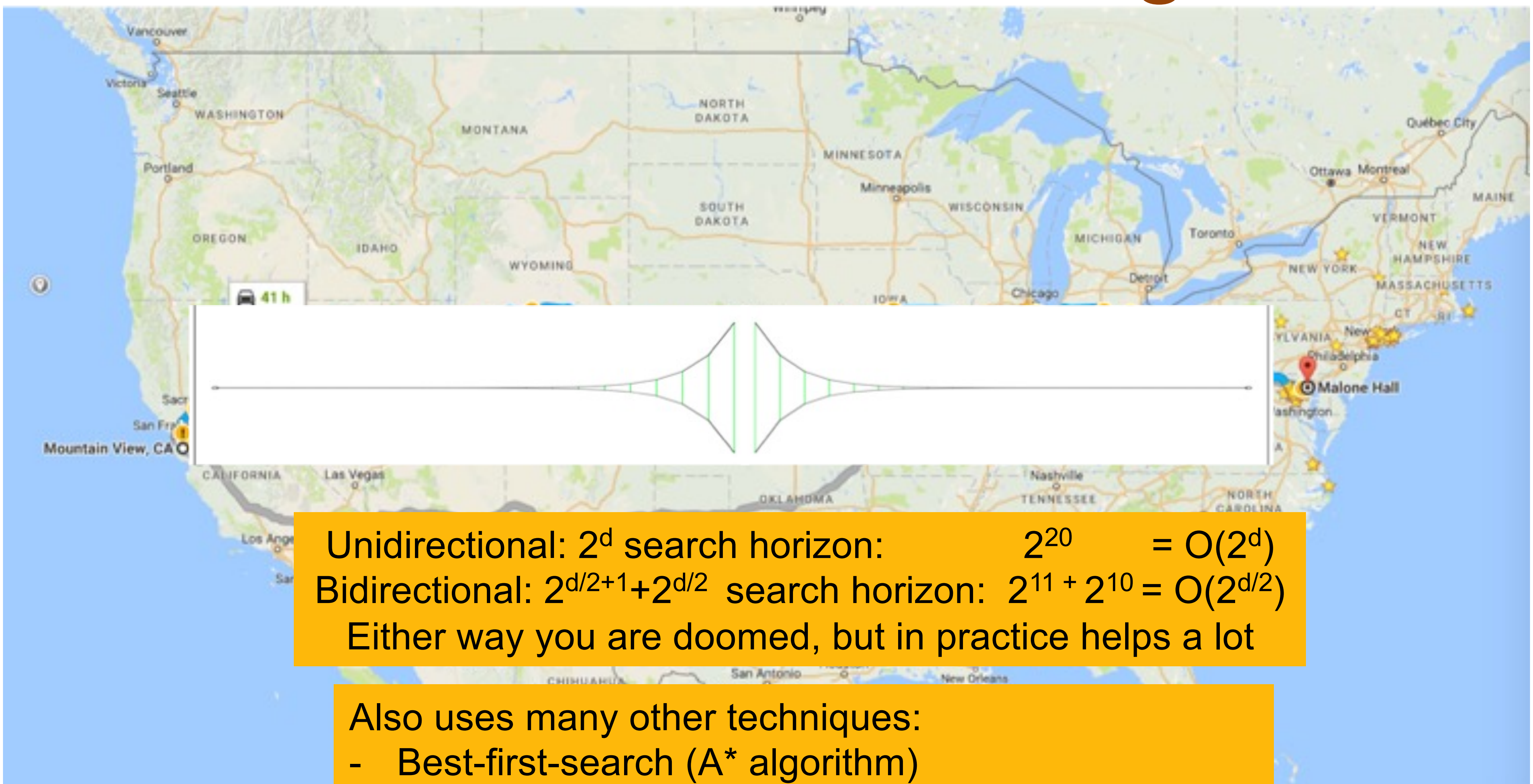


Breath First Searching



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…

Agenda

1. ***Questions on HW4***
2. ***Recap on Graphs***
3. ***Machine Code Optimization***

Machine Code Optimization

Intel Instruction Set

Instruction	Meaning	Notes	Opcode
AAA	ASCII adjust AL after addition	used with unpacked binary coded decimal	0x37
AAD	ASCII adjust AX before division	8086/8088 datasheet documents only base 10 version of the AAD instruction (opcode 0xD5 0x0A), but any other base will work. Later Intel's documentation has the generic form too. NEC V20 and V30 (and possibly other NEC V-series CPUs) always use base 10, and ignore the argument, causing a number of incompatibilities	0xD5
AAM	ASCII adjust AX after multiplication	Only base 10 version (Operand is 0xA) is documented, see notes for AAD	0xD4
AAS	ASCII adjust AL after subtraction		0x3f
ADC	Add with carry	`destination := destination + source + carry_flag`	0x10...0x15, 0x80/2..., 0x83/2
ADD	Add	(1) `r/m += r/imm;` (2) `r += m/imm;`	0x00...0x05, 0x80/0..., 0x83/0
AND	Logical AND	(1) `r/m &= r/imm;` (2) `r &= m/imm;`	0x20...0x25, 0x80/4..., 0x83/4
CALL	Call procedure	`push eip; eip points to the instruction directly after the call`	0x9A, 0xE8, 0xFF/2, 0xFF/3
CBW	Convert byte to word		0x98
CLC	Clear carry flag	`CF = 0;`	0xF8
CLD	Clear direction flag	`DF = 0;`	0xFC
CLI	Clear interrupt flag	`IF = 0;`	0xFA
CMC	Complement carry flag		0xF5
CMP	Compare operands		0x38...0x3D, 0x80/7..., 0x83/7
CMPSB	Compare bytes in memory		0xA6
CMPSW	Compare words		0xA7
CWD	Convert word to doubleword		0x99
DAA	Decimal adjust AL after addition	(used with packed binary coded decimal)	0x27
DAS			0x2F
DEC	Decrement by 1		0x48, 0xFE/1, 0xFF/1

Note: modern processors have hundreds of instructions

Intro to machine code

```c
unsigned int fib(unsigned int n)
{
    if (n <= 0)
        return 0;
    else if (n <= 2)
        return 1;
    else {
        unsigned int a,b,c;
        a = 1;
        b = 1;
        while (1) {
            c = a + b;
            if (n <= 3) return c;
            a = b;
            b = c;
            n--;
        }
    }
}
```

compiler →

```asm
fib:
    mov edx, [esp+8]
    cmp edx, 0
    ja @f
    mov eax, 0
    ret

    @@:
    cmp edx, 2
    ja @f
    mov eax, 1
    ret

    @@:
    push ebx
    mov ebx, 1
    mov ecx, 1

    @@:
        lea eax, [ebx+ecx]
        cmp edx, 3
        jbe @f
        mov ebx, ecx
        mov ecx, eax
        dec edx
    jmp @b

    @@:
    pop ebx
    ret
```

← assembler

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
C9010000 008D0419 83FA0376 078BD98B
B84AEBF1 5BC3
```

Java bytecode instructions

Mnemonic	Opcode *(in hexadecimal)*	Opcode (in binary)	Other bytes	Stack [before]→[after]	Description
aaload	32	0011 0010		arrayref, index → value	load onto the stack a reference from an array
aastore	53	0101 0011		arrayref, index, value →	store into a reference in an array
aconst_null	01	0000 0001		→ null	push a *null* reference onto the stack
aload	19	0001 1001	1: index	→ objectref	load a reference onto the stack from a local variable *#index*
aload_0	2a	0010 1010		→ objectref	load a reference onto the stack from local variable 0
aload_1	2b	0010 1011		→ objectref	load a reference onto the stack from local variable 1
aload_2	2c	0010 1100		→ objectref	load a reference onto the stack from local variable 2
aload_3	2d	0010 1101		→ objectref	load a reference onto the stack from local variable 3
anewarray	bd	1011 1101	2: indexbyte1, indexbyte2	count → arrayref	create a new array of references of length *count* and component type identified by the class reference *index* (`indexbyte1 << 8 + indexbyte2`) in the constant pool
areturn	b0	1011 0000		objectref → [empty]	return a reference from a method
arraylength	be	1011 1110		arrayref → length	get the length of an array
astore	3a	0011 1010	1: index	objectref →	store a reference into a local variable *#index*
astore_0	4b	0100 1011		objectref →	store a reference into local variable 0
astore_1	4c	0100 1100		objectref →	store a reference into local variable 1
astore_2	4d	0100 1101		objectref →	store a reference into local variable 2

https://en.wikipedia.org/wiki/Java_bytecode_instruction_listings

Java bytecode instructions

```
outer:
for (int i = 2; i < 1000; i++) {
    for (int j = 2; j < i; j++) {
        if (i % j == 0)
            continue outer;
    }
    System.out.println (i);
}
```

```
8B542408 83FA0077 06B80000 0000C383
FA027706 B8010000 00C353BB 01000000
C9010000 008D0419 83FA0376 078BD98B
B84AEBF1 5BC3
```

```
0:   iconst_2
1:   istore_1
2:   iload_1
3:   sipush  1000
6:   if_icmpge       44
9:   iconst_2
10:  istore_2
11:  iload_2
12:  iload_1
13:  if_icmpge       31
16:  iload_1
17:  iload_2
18:  irem
19:  ifne    25
22:  goto    38
25:  iinc    2, 1
28:  goto    11
31:  getstatic       #84;
34:  iload_1
35:  invokevirtual   #85;
38:  iinc    1, 1
41:  goto    2
44:  return
```

https://en.wikipedia.org/wiki/Java_bytecode

Reverse Engineering Bytecode

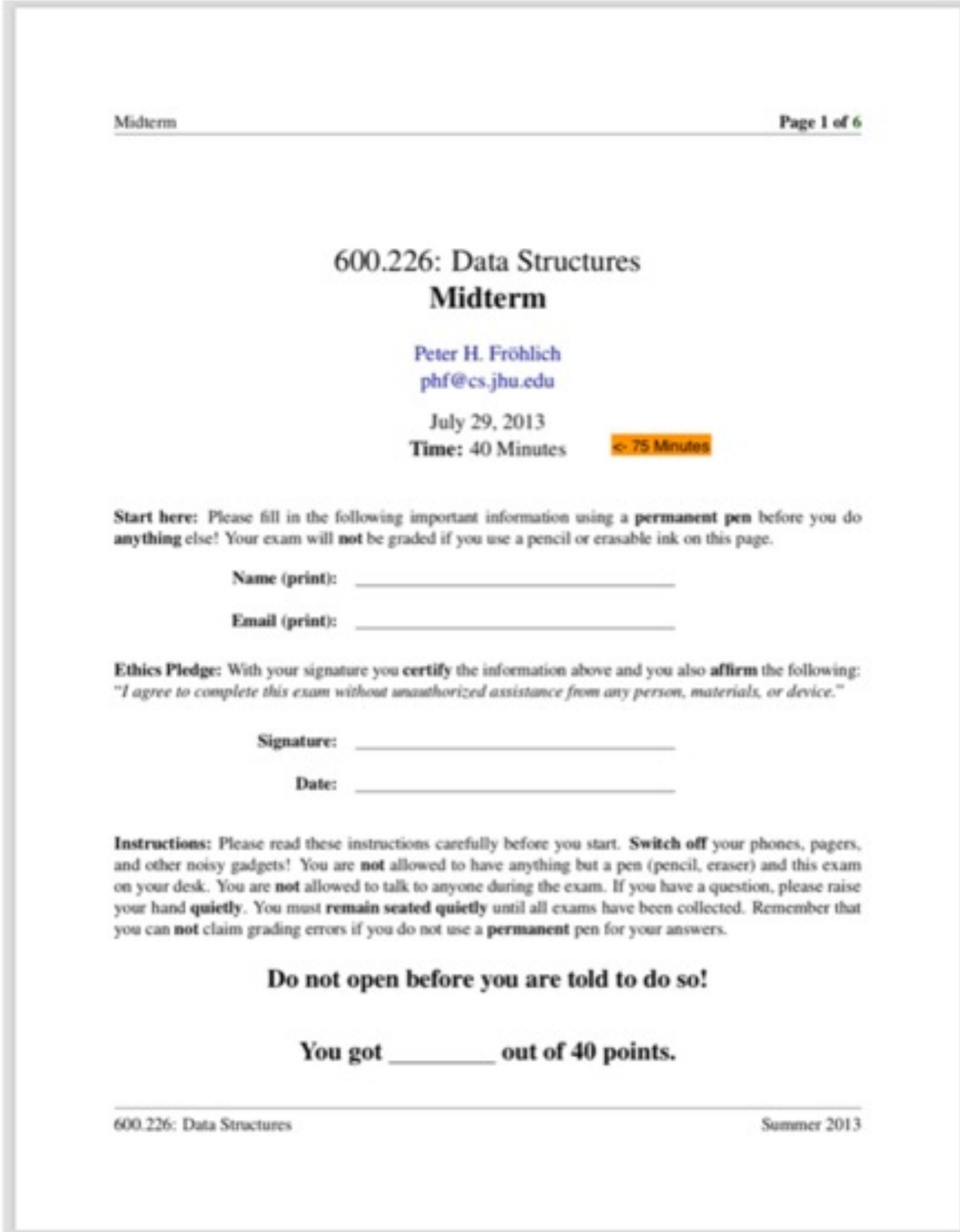
```
$ od -x Mystery.class
0000000      feca    beba    0000    3400    3c00    000a    0013    061c
0000020      0040    0000    0000    0000    4006    003f    0000    0000
0000040      0a00    1d00    1e00    0009    001f    0720    2100    000a
0000060      0008    081c    2200    000a    0008    0a23    0800    2400
0000100      0008    0a25    0800    2600    0008    0a27    0800    2800
0000120      000a    0029    072a    2b00    0007    012c    0600    693c
0000140      696e    3e74    0001    2803    5629    0001    4304    646f
0000160      0165    0f00    694c    656e    754e    626d    7265    6154
0000200      6c62    0165    0400    616d    6e69    0001    2816    4c5b
0000220      616a    6176    6c2f    6e61    2f67    7453    6972    676e
0000240      293b    0156    0a00    6f53    7275    6563    6946    656c
0000260      0001    520e    7665    6569    4977    746e    6a2e    7661
0000300      0c61    1400    1500    0007    0c2d    2e00    2f00    0007
0000320      0c30    3100    3200    0001    6a17    7661    2f61    616c
0000340      676e    532f    7274    6e69    4267    6975    646c    7265
0000360      0001    6403    203a    000c    0033    0c34    3300    3500
0000400      0001    2004    3a69    0c20    3300    3600    0001    2006
0000420      2b69    3a31    0c20    3700    3800    0007    0c39    3a00
0000440      3b00    0001    5209    7665    6569    4977    746e    0001
0000460      6a10    7661    2f61    616c    676e    4f2f    6a62    6365
0000500      0174    0e00    616a    6176    6c2f    6e61    2f67    614d
0000520      6874    0001    7003    776f    0001    2805    4444    4429
0000540      0001    6a10    7661    2f61    616c    676e    532f    7379
0000560      6574    016d    0300    756f    0174    1500    6a4c    7661
0000600      2f61    6f69    502f    6972    746e    7453    6572    6d61
0000620      013b    0600    7061    6570    646e    0001    282d    6a4c
0000640      7661    2f61    616c    676e    532f    7274    6e69    3b67
0000660      4c29    616a    6176    6c2f    6e61    2f67    7453    6972
0000700      676e    7542    6c69    6564    3b72    0001    281c    2944
0000720      6a4c    7661    2f61    616c    676e    532f    7274    6e69
0000740      4267    6975    646c    7265    013b    1c00    4928    4c29
```

Reverse Engineering Bytecode

```
$ od -x Mystery.class
0000000    feca    beba    0000    3400    3c00    000a    0013    061c
0000020    0040    0000    0000    0000    4006    003f    0000    0000
0000040    0a00    1d00    1e00    0009    001f    0720    2100    000a
0000060    0008    081c    2200    000a    0008    0a23    0800    2400
0000100    0008    0a25    0800    2600    0008    0a27    0800    2800
0000120    000a    0029    072a    2b00    0007    012c    0600    693c
0000140    696e    3e74    0001    2803    5629    0001    4304    646f
0000160    0165    0f00    694c    656e    754e    626d    7265    6154
0000                                                            4c5b
0000                                                            676e
0000                                                            656c
0000                                                            7661
0000                                                            0007
0000                                                            616c
0000                                                            7265
0000                                                            3500
0000                                                            2006
0000                                                            3a00
0000                                                            0001
0000                                                            6365
0000                                                            614d
0000                                                            4429
0000                                                            7379
0000560    6574    016d    0300    756f    0174    1500    6a4c    7661
0000600    2f61    6f69    502f    6972    746e    7453    6572    6d61
0000620    013b    0600    7061    6570    646e    0001    282d    6a4c
0000640    7661    2f61    616c    676e    532f    7274    6e69    3b67
0000660    4c29    616a    6176    6c2f    6e61    2f67    7453    6972
0000700    676e    7542    6c69    6564    3b72    0001    281c    2944
0000720    6a4c    7661    2f61    616c    676e    532f    7274    6e69
0000740    4267    6975    646c    7265    013b    1c00    4928    4c29
```

Just Kidding ☺

Midterm Review



600.226: Data Structures

Midterm

Peter H. Fröhlich
phf@cs.jhu.edu

July 29, 2013

Time: 40 Minutes <- 75 Minutes

Start here: Please fill in the following important information using a **permanent pen** before you do **anything** else! Your exam will **not** be graded if you use a pencil or erasable ink on this page.

Name (print): ____________________

Email (print): ____________________

Ethics Pledge: With your signature you **certify** the information above and you also **affirm** the following: *"I agree to complete this exam without unauthorized assistance from any person, materials, or device."*

Signature: ____________________

Date: ____________________

Instructions: Please read these instructions carefully before you start. **Switch off** your phones, pagers, and other noisy gadgets! You are **not** allowed to have anything but a pen (pencil, eraser) and this exam on your desk. You are **not** allowed to talk to anyone during the exam. If you have a question, please raise your hand **quietly**. You must **remain seated quietly** until all exams have been collected. Remember that you can **not** claim grading errors if you do not use a **permanent** pen for your answers.

Do not open before you are told to do so!

You got ________ out of 40 points.

Primitive Data Types

The 8 primitive data types supported by the Java programming language are:

1. ***byte:*** 8-bit signed two's complement integer: [-128, 127]
2. ***short:*** 16-bit signed two's complement integer: [-32,768, 32,767]
3. ***int:*** 32-bit signed two's complement integer: [-2^{31}, 2^{31}-1]
4. ***long:*** 64-bit two's complement integer: [-2^{63}, 2^{63}-1]
5. ***float:*** Single-precision 32-bit IEEE 754 floating point. Good for saving memory in large arrays of values.
6. ***double:*** Double-precision 64-bit IEEE 754 floating point. Default choice for decimal values
7. ***boolean:*** Two possible values: true and false. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
8. ***char:*** The char data type is a single 16-bit Unicode character.

Everything else is an Object

Note there is an Object version of each primitive data type and Java will try to convert back and forth when you need it:

int ⇔ Integer, float ⇔ Float, etc

Classes & Objects

All java code must be in some class (and inside some package)
• If no package name is listed, code goes into unnamed package
• This helps organize code, and avoids naming conflicts: if your code defines a method "print", and my code defines a method "print" specifying the class (and package) will clarify which one you mean

However, we don't always want nor need an object to call a method:

```
class MathStuff {
    public int max3(int a, int b, int c) { … }
}

MathStuff stuff = new MathStuff();
int biggest = stuff.max3(42,14,99);
```

Use the ***static*** keyword to tell the compiler that it is okay to call this method directly (without an object):

```
class MathStuff {
    public static int max3(int a, int b, int c) { … }
}

int biggest = MathStuff.max3(42,14,99);
```

Variables

Instance Variables (Non-Static Fields)

- Values are unique to each instance of a class (to each object)

Class Variables (Static Fields)

- Tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated.

Local Variables

- Similar to how an object stores its state in fields, a method can store its temporary state in local variables. Local variables are only visible to the methods in which they are declared

Parameters

- Similar to local variables, although are passed in from other calling methods.

Final Variables

- Final means the value can only be set once

Variables

Instance Variables (Non-Static Fields)

- Values are unique to each instance of a class (to each object)

Class Variables (Static Fields)

- Tells the compiler that there is exactly one copy of this variable in existence, regardless of how many times the class has been instantiated.

Local Variables

- Similar to how an object stores its state in fields, a method can store its temporary state in local variables. Local variables are only visible to the methods in which they are declared

Parameters

- Similar to local variables, although are passed in from other calling methods.

Final Variables

- Final

How should you store the currentSpeed in a Bicycle class?

How should you store Pi in a Math class?

Controlling Access

Use access level modifiers to restrict access to methods and member variables (enforced by the compiler and JRE!)

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Access levels affect you in two ways:

- When you use classes that come from another source, such as the classes in the Java platform, access levels determine which members of those classes your own classes can use.
- When you write a class, you need to decide what access level every member variable and every method in your class should have.

https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Controlling Access

Use access level modifiers to restrict access to methods and member variables (enforced by the compiler and JRE!)

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Access levels affect you in two ways:

- When you use classes that come from another source, such as the classes … embers of those classes your own classes can use.
- When … ery member variable and every method in your class should have.

Should you make all your methods and fields public?

No way! Try to make access as restricted as possible!

https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Introduction to Java Interfaces

Objects define their interaction with the outside world through the methods that they expose. Methods form the object's interface with the outside world; the buttons on the front of your television set, for example, are the interface between you and the electrical wiring on the other side of its plastic casing. You press the "power" button to turn the television on and off. […] ***An interface is a group of related methods with empty bodies.***

https://docs.oracle.com/javase/tutorial/java/concepts/interface.html


```
interface Counter {
  int value();
  void up();
  void down();
}
```

```
specification:
    Counter has an integer value
    '+' button increments by 1
    '-' button decrements by 1
```

Specification can be a separate documents or in the javadoc comments

Nested Classes

The Java programming language allows you to define a class within another class – a nested class.

```
class OuterClass {
    ...
    class NestedClass {
        ...
    }
}
```

- ***It is a way of logically grouping classes that are only used in one place:*** If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.

- ***It increases encapsulation:*** Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.

- ***It can lead to more readable and maintainable code:*** Nesting small classes within top-level classes places the code closer to where it is used.

https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

Nested and Inner Classes

Nested classes are divided into two categories: static and non-static.

- Nested classes that are declared static are called ***static nested classes***.
- Non-static nested classes are called ***inner classes.***

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

A static nested class is associated with its outer class. And like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class: it can use them only through an object reference.

An inner class is associated with an instance of its enclosing class and has direct access to that object's methods and fields. Also, because an inner class is associated with an instance, it cannot define any static members itself.

Nested and Inner Classes

Nested classes are divided into two categories: static and non-static.

- Nested classes that are declared static are called ***static nested classes***.
- Non-static nested classes are called ***inner classes.***

```
class OuterClass {
```

A nested class is a member of its enclosing class.

- As a member of the OuterClass, a nested class can be declared private, public, protected, or package private.

Static nested classes do not have access to other members of the enclosing class.

- Like static methods, you would have to pass in an object reference

Non-static nested classes (inner classes) have access to other members of the enclosing class, even if they are declared private.

- An instance of InnerClass can exist only within an instance of OuterClass

direct access to that object's methods and fields. Also, because an inner class is associated with an instance, it cannot define any static members itself.

Java Abstract Classes

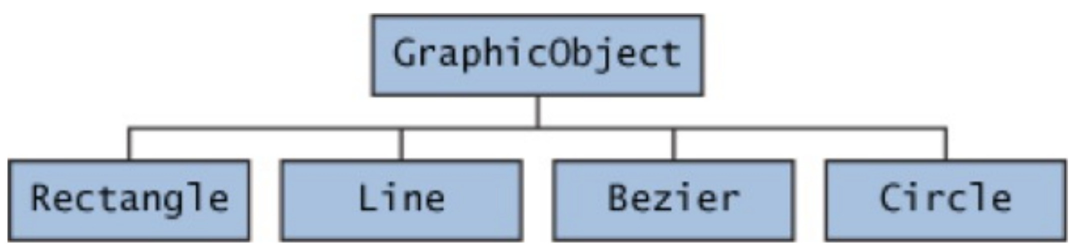
An abstract class is a class that is declared abstract—it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

An abstract method is a method that is declared without an implementation (without braces, and followed by a semicolon), like this:

```
abstract void moveTo(double deltaX, double deltaY);
```

If a class includes abstract methods, then the class itself must be declared abstract, as in:

```
public abstract class GraphicObject {
   // declare fields
   // declare nonabstract and abstact methods
   void setPen(Pen p) { this.pen = p }
   abstract void draw();
}
```



https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html

Abstract Classes

Consider using <u>abstract classes</u> if any of these statements apply to your situation:

- You want to share code among several closely related classes.
- You expect that classes that extend your abstract class have many common methods or fields, or require access modifiers other than public (such as protected and private).
- You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.

Consider using <u>interfaces</u> if any of these statements apply to your situation:

- You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and Cloneable are implemented by many unrelated classes.
- You want to specify the behavior of a particular data type, but not concerned about who implements its behavior.
- You want to take advantage of multiple inheritance of type.

https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html

Java Generics

Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods. Much like the more familiar formal parameters used in method declarations***, type parameters provide a way for you to re-use the same code with different inputs***. The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types.

Code that uses generics has many benefits over non-generic code:

- ***Stronger type checks at compile time.*** A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.

- ***Elimination of casts.*** The following code snippet without generics requires casting:

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

  When re-written to use generics, the code does not require casting:

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);   // no cast
```

- ***Enabling programmers to implement generic algorithms.*** By using generics, programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read.

https://docs.oracle.com/javase/tutorial/java/generics/why.html

Java Generics

Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces and methods. Much like the more familiar formal parameters used in method declarations, ***type parameters provide a way for you to re-use the same code with different inputs***. The difference is that the inputs to formal parameters are values, while the inputs to type parameters are types.

Code that uses generics has many benefits over non-generic code:

- ***Stronger type checks at compile time.*** A Java compiler applies strong type checking to

To use Java generics effectively, you must consider the following restrictions:

- Cannot Instantiate Generic Types with Primitive Types
- Cannot Create Instances of Type Parameters
- Cannot Declare Static Fields Whose Types are Type Parameters
- Cannot Use Casts or `instanceof` With Parameterized Types
- Cannot Create Arrays of Parameterized Types
- Cannot Create, Catch, or Throw Objects of Parameterized Types
- Cannot Overload a Method Where the Formal Parameter Types of Each Overload Erase to the Same Raw Type

When re-written to use generics, the code does not require casting:

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);   // no cast
```

- ***Enabling programmers to implement generic algorithms.*** By using generics, programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read.

https://docs.oracle.com/javase/tutorial/java/generics/why.html

Midterm Topics

Topics

01.Intro (kd-tree)
02.Interfaces
03.ArraysGenericsExceptions
04.Lists
05.Iterators
06.Complexity
07.MoreComplexity
08.Sorting
09.Stacks
10.StacksJunit
11.Queues and Dequeues
12.Lists (Single/Double)
13.MoreLists
14.Trees & Tree Iteration
15.Graphs
16.GraphSearch

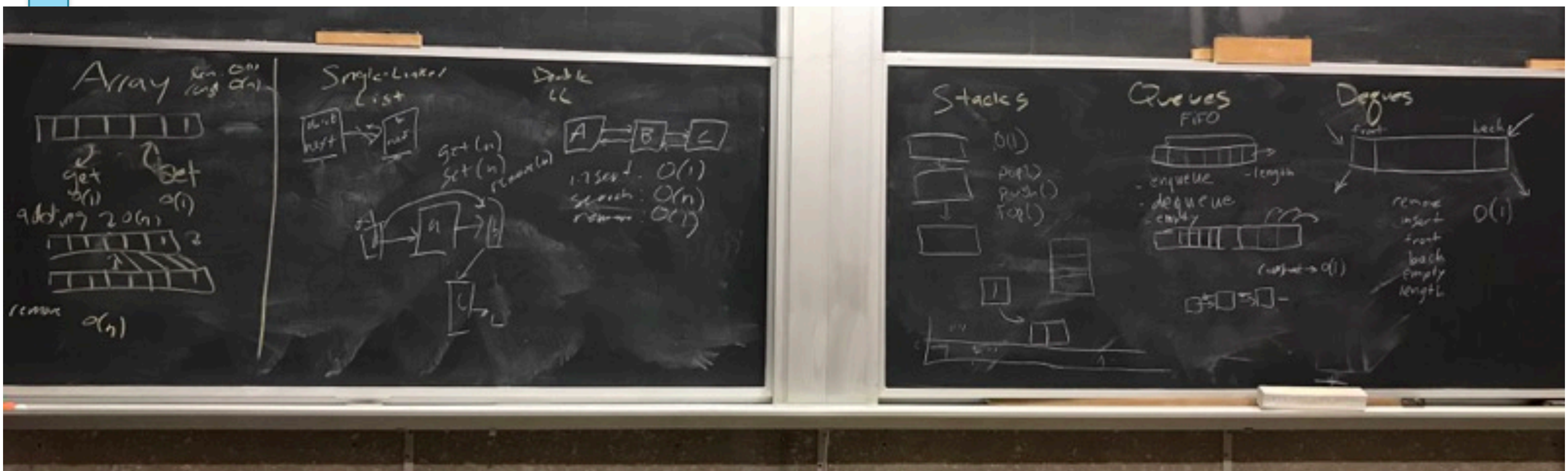
For each data structure discuss:

- Explain the interface
- Explain/Draw how it will be implemented
- Explain/Draw how to add/remove elements
- Iterate through the elements
- Explain the complexity of these

In addition:

- Can you discuss interfaces and ADTs
- Can you discuss computational complexity

Midterm Topics



Next Steps

1. Review for Midterm

2. Check on Piazza for tips & corrections!