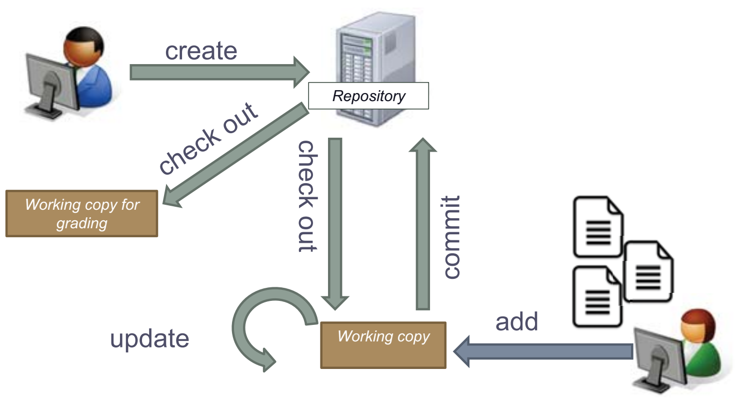
Section 5:
HW6 and Interfaces

Slides adapted from Alex Mariakakis,
with material from Krysta Yousoufian, Mike Ernst, Kellen Donohue

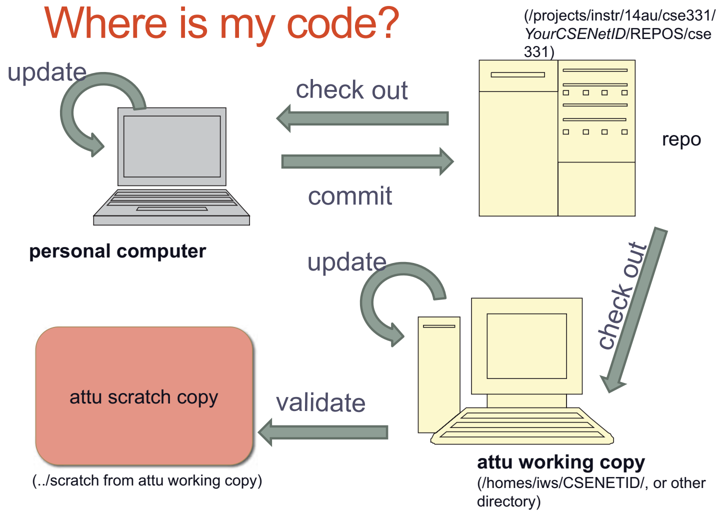
Agenda

- Version control and tools review
- BFS
- Interfaces
- Parsing Marvel Data

331 Version control



Where is my code?


(/projects/instr/14au/cse331/ *YourCSENetID*/REPOS/cse 331)
update
check out
commit
repo
personal computer
update
check out
attu scratch copy
validate
(../scratch from attu working copy)
attu working copy
(/homes/iws/CSENETID/, or other directory)

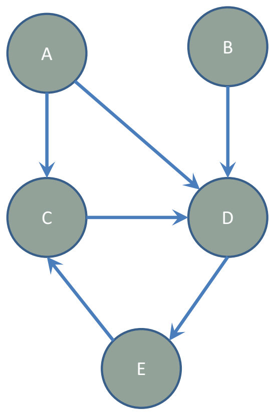
Where is my code?

- Main repo: /projects/instr/etc
  - Not human readable
  - You can't see files here
- Personal computer: any directory, via Subclipse or other
  - Working copy: add and edit files here
  - Must check in files for them to go to the repo
- attu working copy: /homes/iws/CSENETID/ or other
  - Just another working copy, same as personal computer
  - Must svn update to see changes from pc/repo
- validate copy: attu directory/src/…/scratch
  - NEW WORKING COPY CHECKED OUT FROM REPO
  - May NOT be the same as attu working copy if attu wasn't updated

Concepts vs tools, 331 vs general

- Version control: concept
  - Tools: svn, TortoiseSVN, Subclipse
- Ant: tool
  - Concept: build management
  - validate 331
- Remote access: concept
  - Tools: ssh, PuTTY, WinSCP
- Javadocs: tool
  - Concept: documentation
  - @param, @return, @throws general
  - @requires, @modifies, @effects 331

Graphs



Can I reach B from A?

Breadth-First Search (BFS)

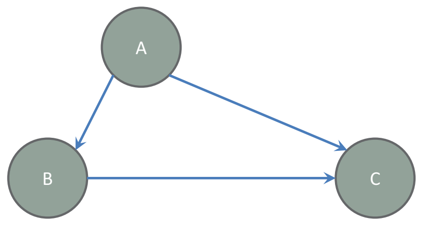
- Often used for discovering connectivity
- Calculates the shortest path if and only if all edges have same positive or no weight
- Depth-first search (DFS) is commonly mentioned with BFS
  - BFS looks “wide”, DFS looks “deep”
  - Can also be used for discovery, but not the shortest path

BFS Pseudocode

```
public boolean find(Node start, Node end) {
      put start node in a queue
      while (queue is not empty) {
            pop node N off queue
            if (N is goal)
                  return true;
            else {
                  for each node O that is child of N
                        push O onto queue
            }
      }
      return false;
}
```

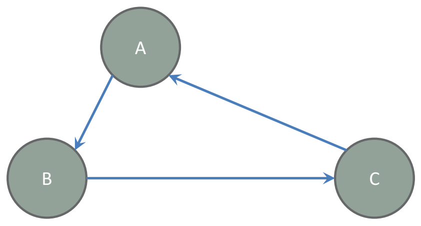
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <B>
Q: <B, C>
DONE



Breadth-First Search with Cycle

Q: <>
Q: <A>
Q: <>
Q: <B>
Q: <>
Q: <C>
Q: <>
Q: <A>
NEVER
DONE



BFS Pseudocode

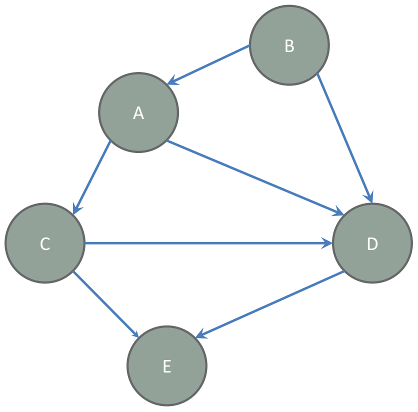
```
public boolean find(Node start, Node end) {
      put start node in a queue
      while (queue is not empty) {
            pop node N off queue
            if (N is goal)
                  return true;
            else {
                  for each node O that is child of N
                        push O onto queue
            }
      }
      return false;
}
```

Mark the node as visited!

What if there’s a cycle?
What if there’s no path between start and end?

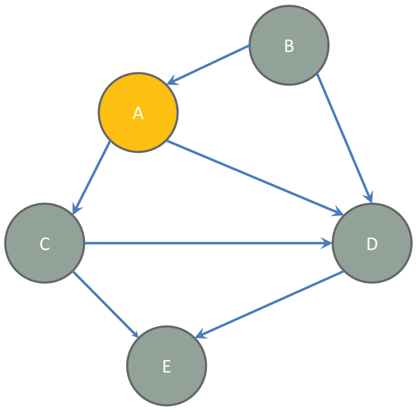
Breadth-First Search

Q: <>



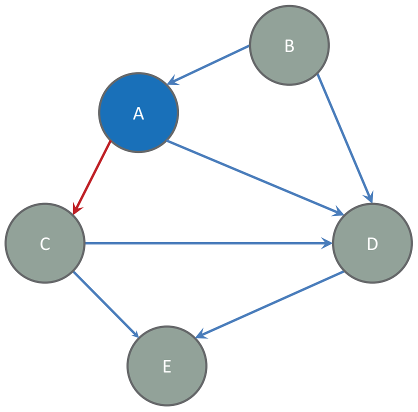
Breadth-First Search

Q: <>
Q: <A>



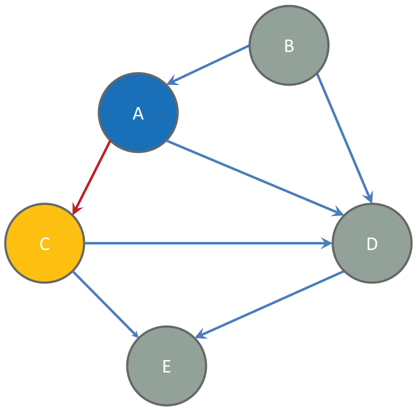
Breadth-First Search

Q: <>
Q: <A>
Q: <>



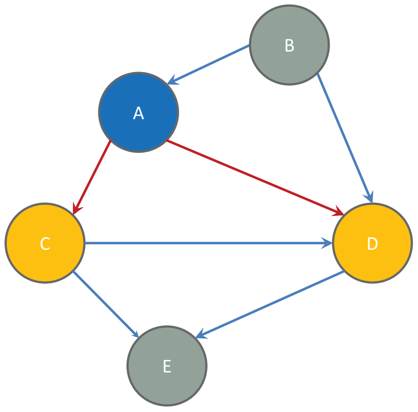
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>



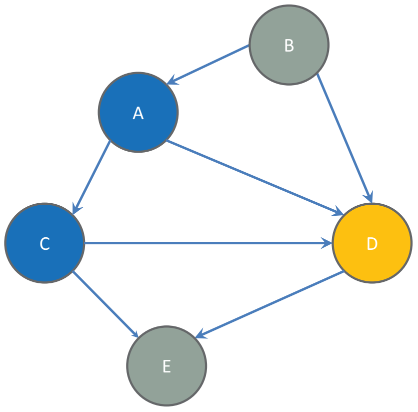
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C ,D>



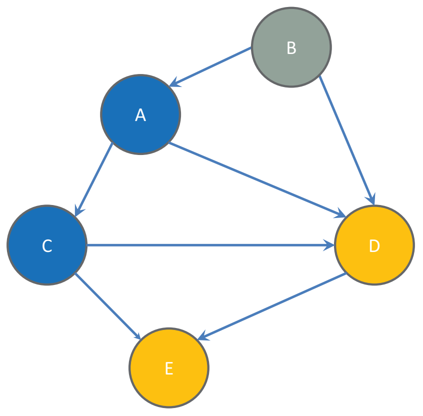
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C ,D>
Q: <D>



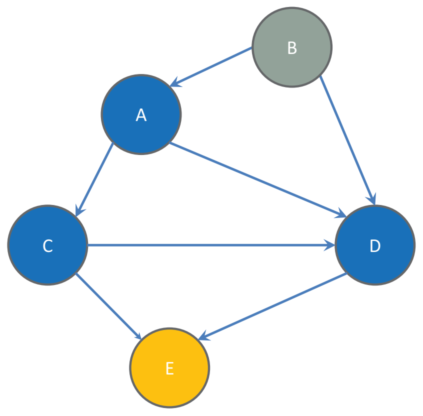
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C ,D>
Q: <D>
Q: <D, E>



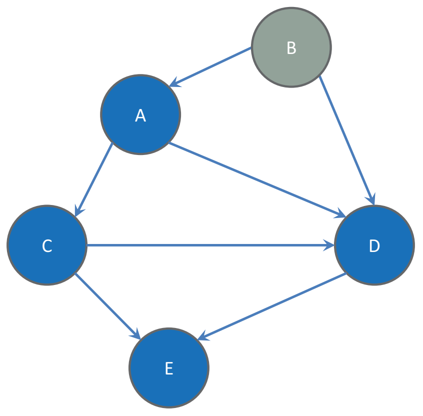
Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C ,D>
Q: <D>
Q: <D, E>
Q: <E>

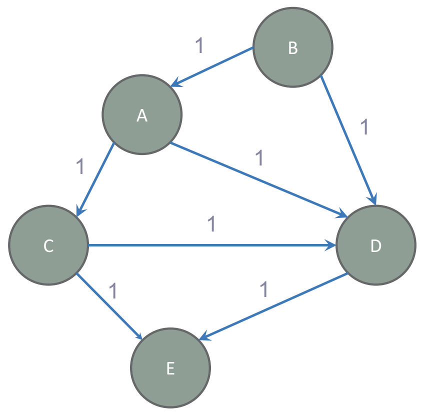


Breadth-First Search

Q: <>
Q: <A>
Q: <>
Q: <C>
Q: <C ,D>
Q: <D>
Q: <D, E>
Q: <E>
DONE



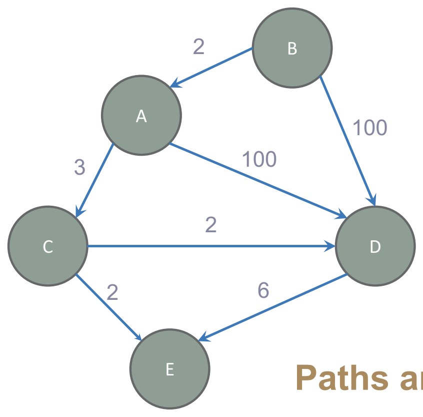
Shortest Paths with BFS



From Node B

Destination	Path	Cost
A	<B,A>	1
B	<B>	0
C	<B,A,C>	2
D	<B,D>	1
E	<B,D,E>	2

Shortest Paths with Weights



From Node B

Destination	Path	Cost
A	<B,A>	2
B	<B>	0
C	<B,A,C>	5
D	<B,A,C,D>	7
E	<B,A,C,E>	7

Paths are not the same!

Classes, Interfaces, and Types

- The fundamental unit of programming in Java is a class
- Classes can extend other classes and implement interfaces
- Interfaces can extend other interfaces

Classes, Objects, and Java

- Everything is an instance of a class
  - Defines data and methods
- Every class extends exactly one other class
  - Object if no explicit superclass
  - Inherits superclass fields
- Every class also defines a type
  - Foo defines type Foo
  - Foo inherits all inherited types
- Java classes contain both specification and implementation!

Interfaces

- Pure type declaration

```
    public interface Comparable {
        int compareTo(Object other);
    }
```

- Can contain:
  - Method specifications (implicitly `public abstract`)
  - Named constants (implicitly `public final static`)
- Does not contain implementation
- Cannot create instances of interfaces

Implementing Interfaces

- A class can implement one or more interfaces

```
class Kitten implements Pettable, Huggable
```

- The implementing class and its instances have the interface type(s) as well as the class type(s)
- The class must provide or inherit an implementation of all methods defined by the interface(s)
  - Not true for abstract classes

Using Interface Types

- An interface defines a type, so we can declare variables and parameters of that type
- A variable with an interface type can refer to an object of any class implementing that type

```
List<String> x = new ArrayList<String>();
void sort(List myList) {…}
```

Guidelines for Interfaces

- Provide interfaces for significant types and abstractions
- Write code using interface types like Map instead of HashMap and TreeMap wherever possible
  - Allows code to work with different implementations later on
- Both interfaces and classes are appropriate in various circumstances

Demo

Parsing the Marvel data