Does this code compile?

Assuming that `GuardDog` inherits from `Dog`, that its constructor takes a `String` and an `int`, and that it has a `growl` method, does the following code compile? Why or why not?

```java
Dog d = new GuardDog("fluffy", 1);
d.growl();
```

Firstname Lastname

T. 12 / 04

(Your response)

interface

what a piece of code can do

type

describe a set of supported operations

subtype

add more operations to an existing type

implementation

how a piece of code works

class

implement a type’s operations

subclass

re-use/modify an existing implementation

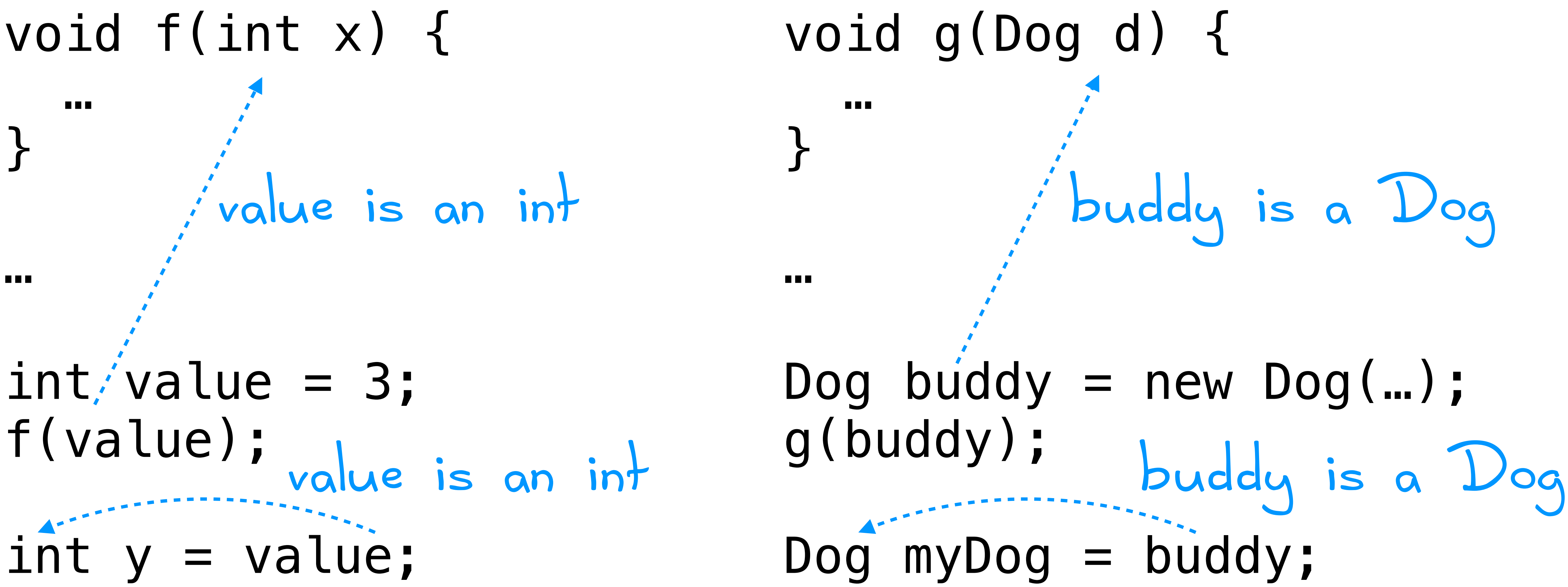
inheritance

usually extends interface *and* implementation

Declared type

When we declare a variable to be of a particular type, the value of that variable must always be an instance of that type.

If a variable has a type, then the value of that variable can be used anywhere that type is expected.



```
void f(int x) {
  …
}

…

int value = 3;
f(value);

int y = value;
```

value is an int

value is an int

```
void g(Dog d) {
  …
}

…

Dog buddy = new Dog(…);
g(buddy);

Dog myDog = buddy;
```

buddy is a Dog

buddy is a Dog

Subtyping: the "is-a" relationship

Implementing an interface establishes an is-a relationship.

Extending an interface establishes an is-a relationship.

Extending a class establishes an is-a relationship.

If we have the following declaration

`Type variable;`

Dog buddy;

then:

- `variable`'s declared type is `Type`.
- `variable` "is a" `Type`.
- If `Type` is an interface, then `variable` "is" all the interfaces that `Type` transitively extends.
- If `Type` is a class, then variable "is" all the classes that `Type` transitively extends *and* "is" all the interfaces that `Type` transitively implements

buddy's d.t. is Dog

buddy is a Dog

buddy is a Pet and an Animal

Subtyping as **substitutability**

When we declare a variable to be of a particular type, we say that the value of that variable should always be an instance of that type **or one of its subtypes**.

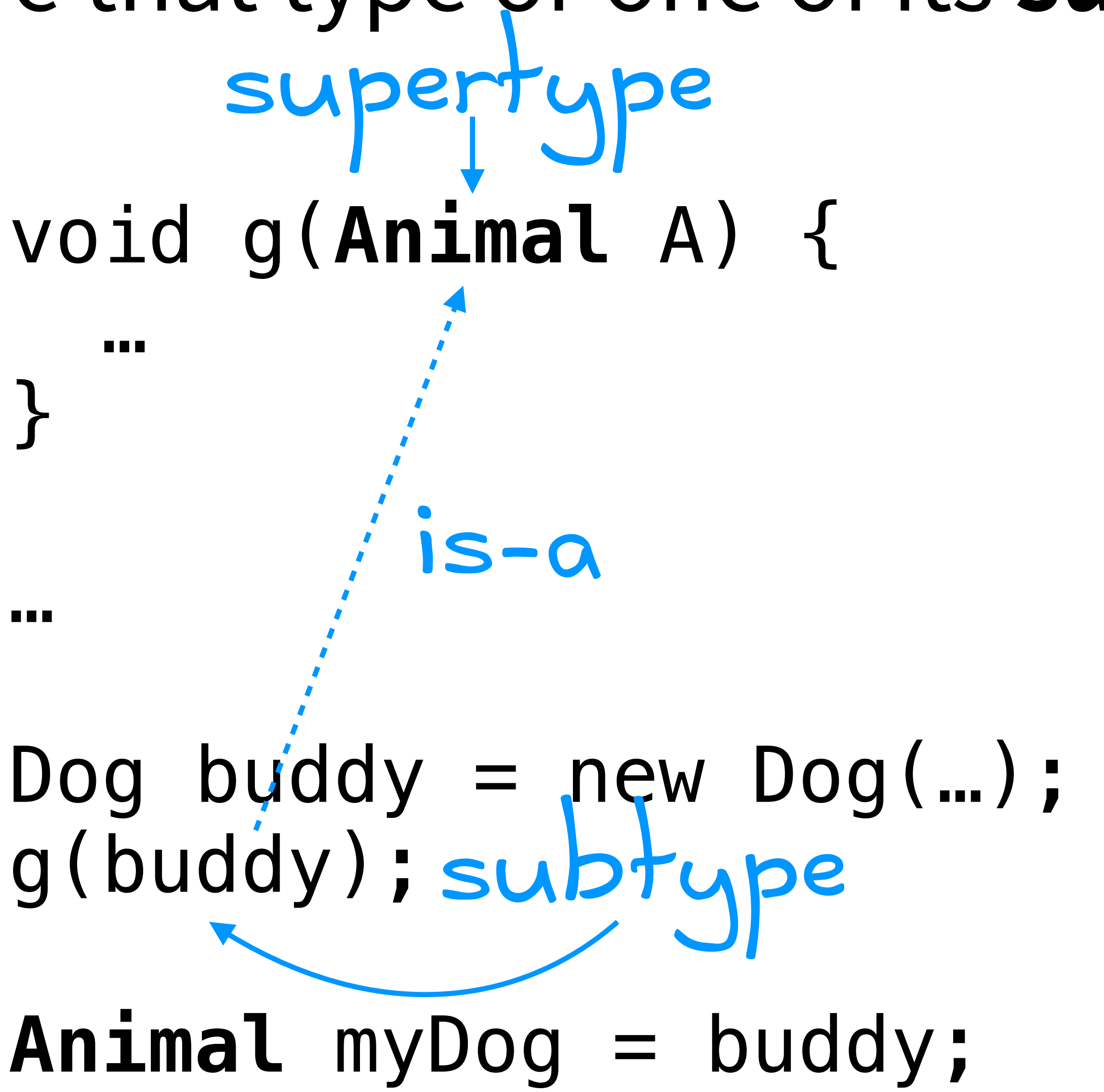
If a variable has a type, then the value of that variable can be used anywhere that type or one of its **supertypes** is expected.

```
void g(Animal A) {
  …
}


…

Dog buddy = new Dog(…);
g(buddy);

Animal myDog = buddy;
```

Subtyping as **substitutability**

When we declare a variable to be of a particular type, we say that the value of that variable should always be an instance of that type **or one of its subtypes**.

If a variable has a type, then the value of that variable can be used anywhere that type or one of its **supertypes** is expected.



```
void g(Animal A) {
   …
}

…

Dog buddy = new Dog(…);
g(buddy);

Animal myDog = buddy;
```

Subtyping as **substitutability**

When we declare a variable to be of a particular type, we say that the value of that variable should always be an instance of that type **or one of its subtypes**.

If a variable has a type, then the value of that variable can be used anywhere that type or one of its **supertypes** is expected.

```
void g(Animal A) {
   …
}


…

Dog buddy = new Dog(…);
g(buddy);

Animal myDog = buddy;
```

is-a

supertype →

← subtype

Declared type *vs* actual type

The type checker looks at the **declared type** (not the value) to see if method calls are legal.

`x.getSpots()` is legal only if the declared type of `x` guarantees there's a `getSpots` method.

When code runs, Java looks at the **actual object** (not the claimed type) to choose the right method to run.

`animal.speak()` does different things, depending on what kind of object `animal` is currently referencing.

Inheritance Puzzles

Does this type check in Java?

```
Cat c = new Cat("Nala", 14);
```

✓

Inheritance Puzzles

Does this type check in Java?

```
Cat c = new Cat("Nala", 14);
c.speak();
```

✓

Meow

Inheritance Puzzles

Does this type check in Java?

```java
Animal a = new Cat("Nala", 14);
```

✓

Inheritance Puzzles

Does this type check in Java?

```java
Animal a = new Cat("Nala", 14);
a.speak();
```

✓

Meow

Inheritance Puzzles

Does this type check in Java?

```
Dog d = new Dalmatian("Pango", 3, 101);
```

✓

Inheritance Puzzles

Does this type check in Java?

```java
Dalmatian d = new Dog("Pango", 101);
```

✗

Inheritance Puzzles

Does this type check in Java?

```java
Dog d = new Dalmatian("Pango", 3, 101);
Dalmatian dm = d;
```

✗

Inheritance Puzzles

Does this type check in Java?

```java
GuardDog gd = new GuardDog("fluffy", 1);
gd.growl();
```

✓

Inheritance Puzzles

Does this type check in Java?

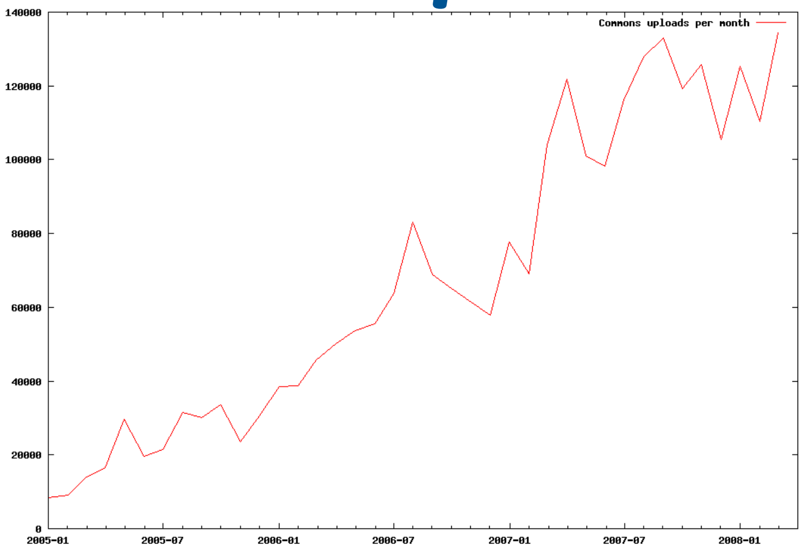
```java
Dog d = new GuardDog("fluffy", 1);
d.growl();
```

✗

Courtesy of Prof. Bassman

Graphs

nope

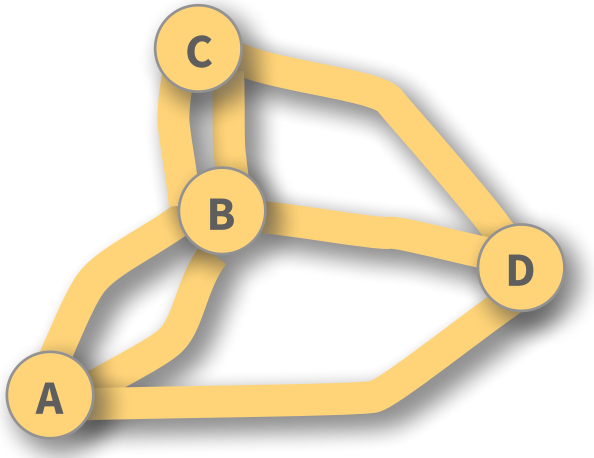


upload.wikimedia.org/wikipedia/commons/4/45/Wikimedia_Commons_monthly_uploads_graph.png

nope

upload.wikimedia.org/wikipedia/commons/9/93/Giraffe_head_1a_(7110736311).jpg

yep!



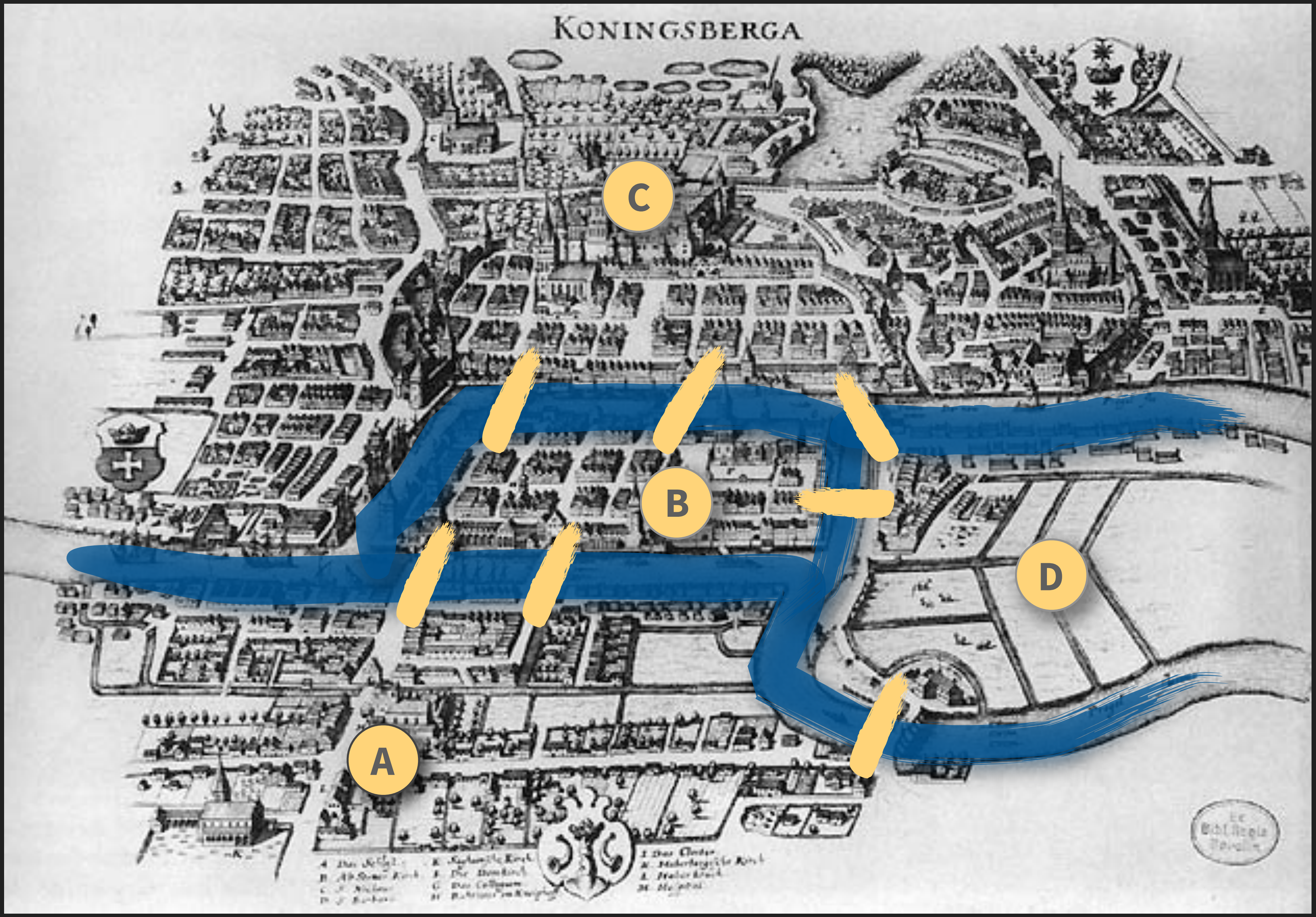
The Seven Bridges of Königsberg

Can you:

start at point A ,
cross every bridge only once,
and return to point A ?

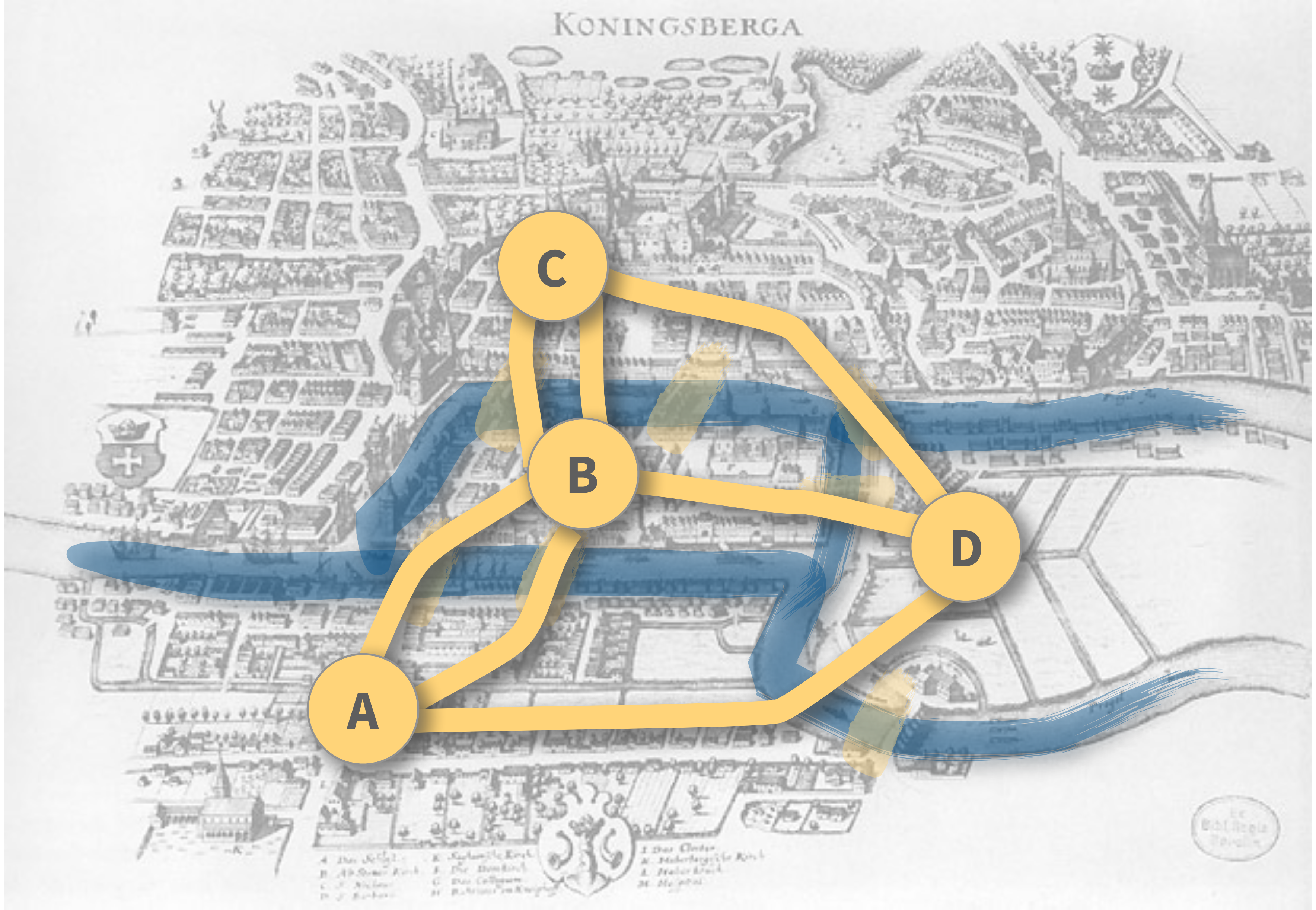
Leonard Euler

ommons.wikimedia.org/wiki/File:Leonard_Euler.jpg



Koenigsberg, Map by Merian-Erben 1652
commons.wikimedia.org/wiki/File:Image-Koenigsberg,_Map_by_Merian-Erben_1652.jpg

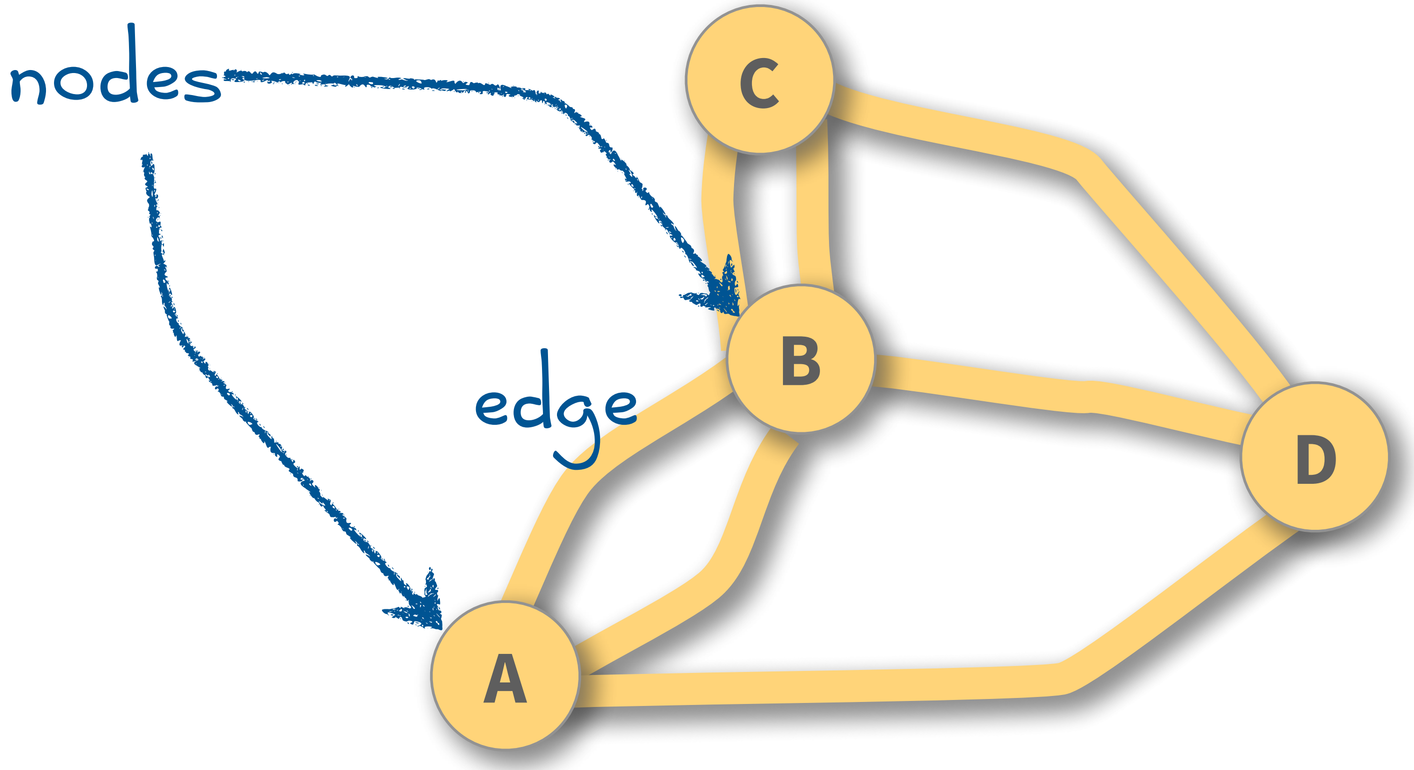
We need a model



Koenigsberg, Map by Merian-Erben 1652
commons.wikimedia.org/wiki/File:Image-Koenigsberg,_Map_by_Merian-Erben_1652.jpg

Graphs!

A set of **nodes/vertices** (places), and a set of **edges** (links)

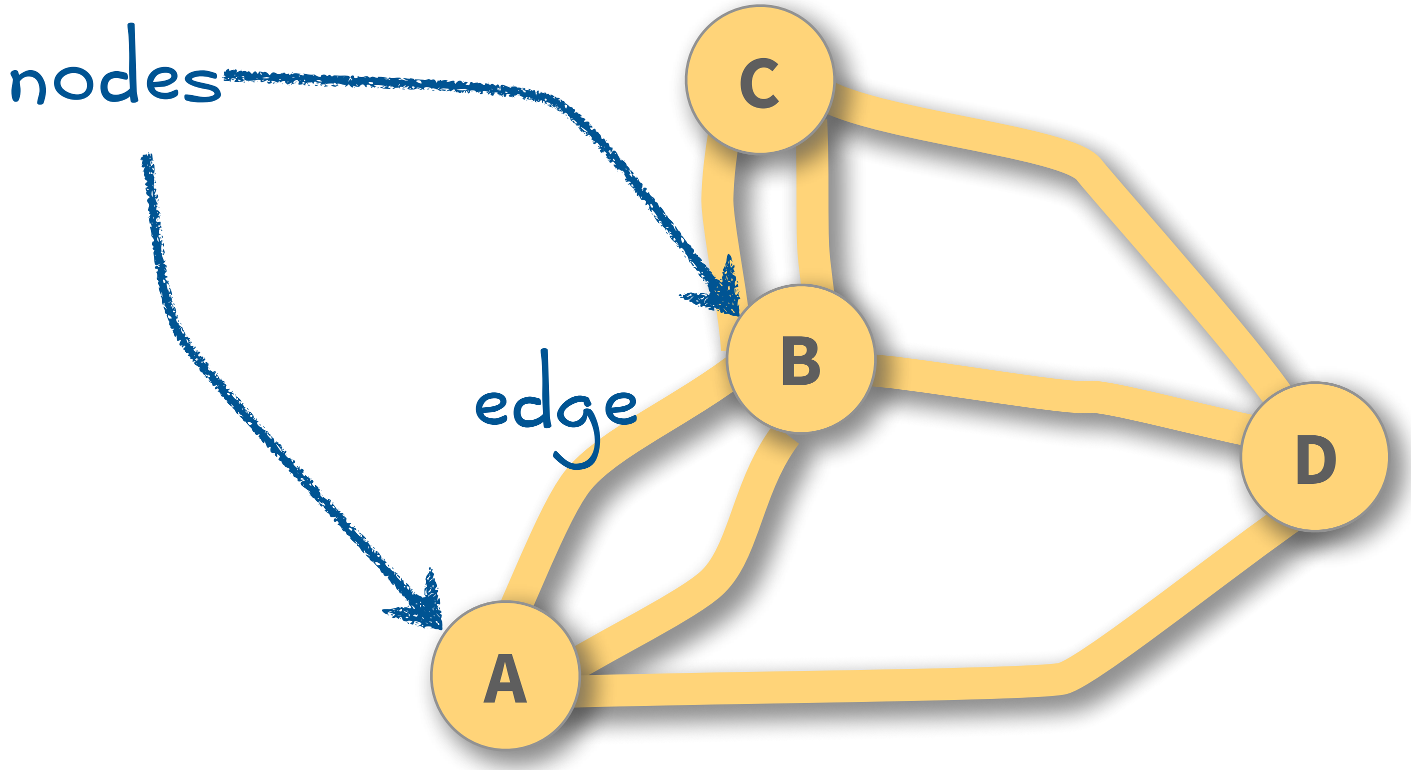


Graphs

represent relationships

Like what?

- A node is …
- An edge is…

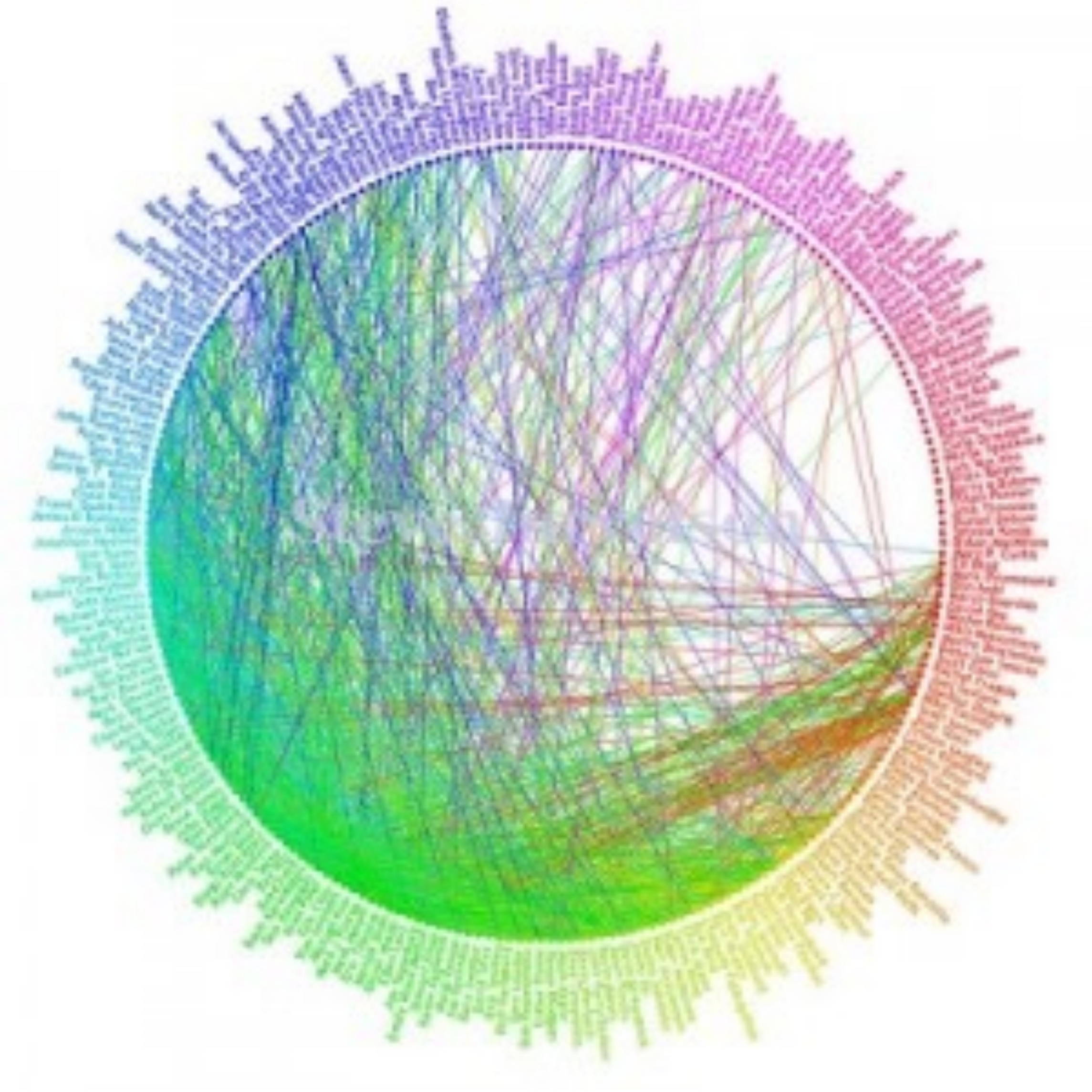


Graphs

represent relationships

Like Facebook

- A node is a Facebook user
- An edge is a “friendship”



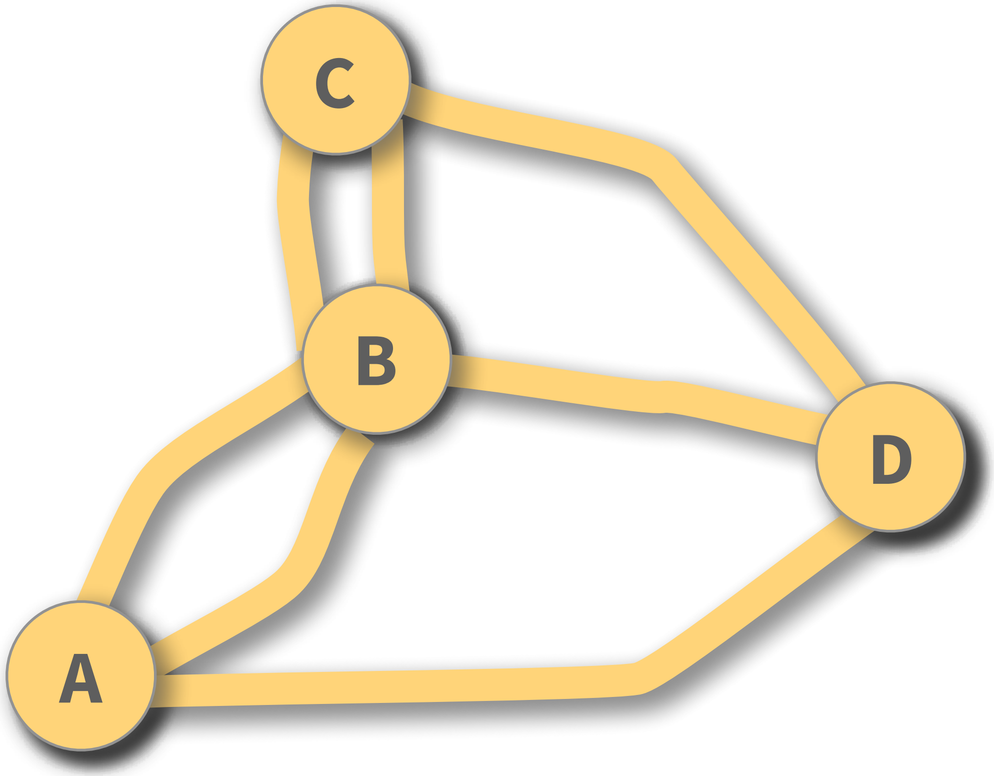
www.mcdougallinteractive.com/wp-content/uploads/2013/02/facebook-graph-search-400x400-300x300.jpg

Undirected graph

← important vocabulary!

We can "traverse the edge" in both directions.

The relationship is "mutual".

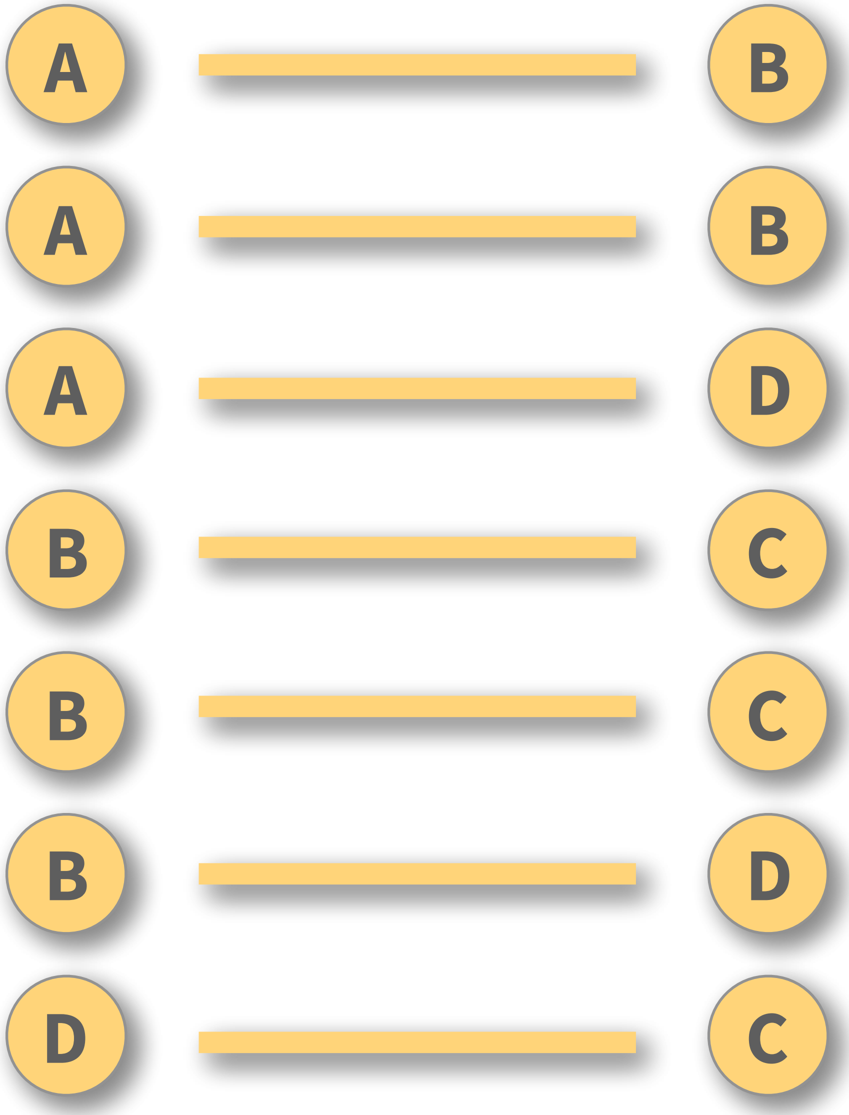


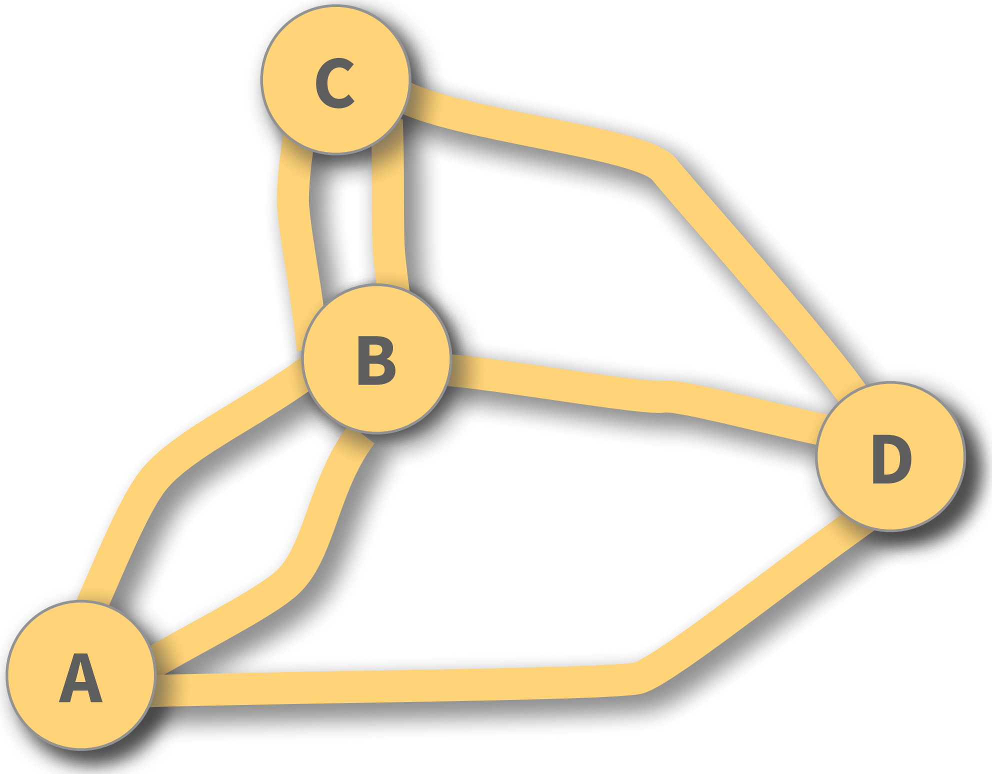
Undirected graph

← important vocabulary!

We can "traverse the edge" in both directions.

The relationship is "mutual".





Graphs

represent relationships

Like Twitter

- A node is a Twitter user
- An edge is a “follow”

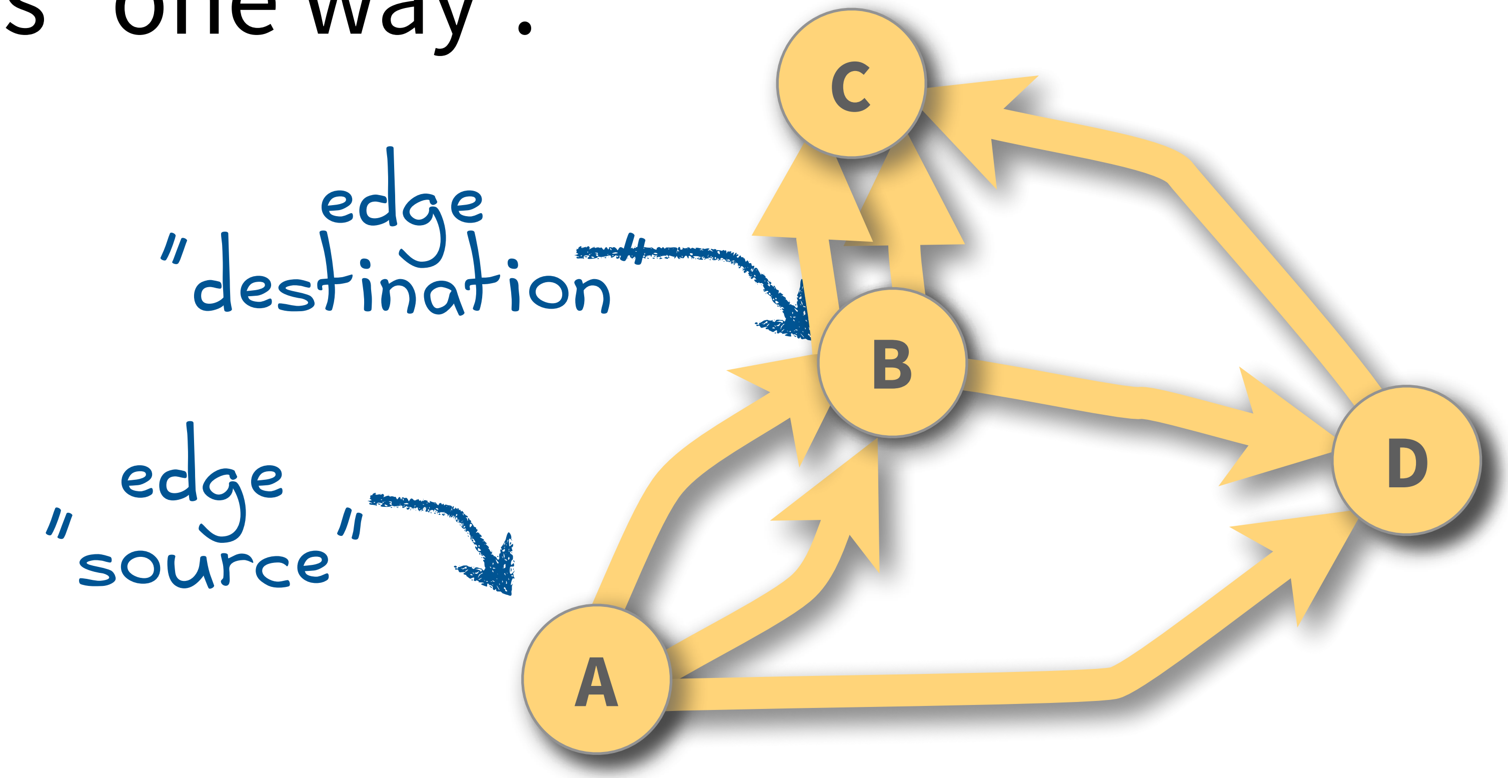
williamjturkel.files.wordpress.com/2011/08/fig-5-niche-twitter-followers-20110421.jpg

Directed graph

← important vocabulary!

We can "traverse the edge" in one direction.

The relationship is "one way".



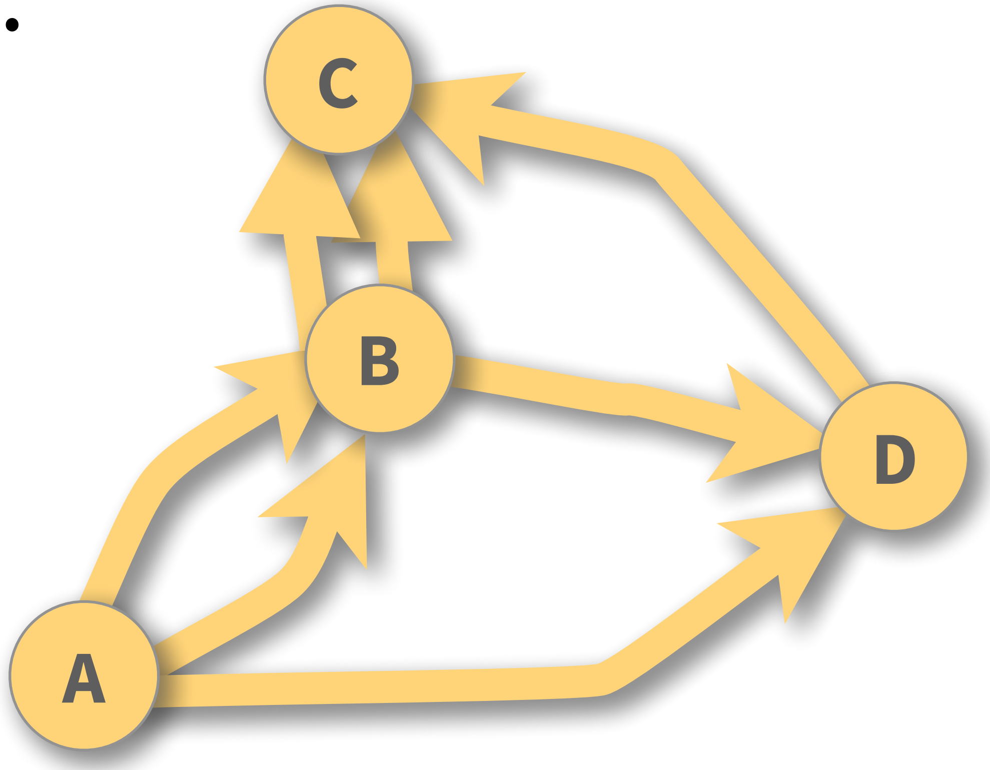
Directed graph

← important vocabulary!

We can "traverse the edge" in one direction.

The relationship is "one way".

edge "source"		edge "destination"
A	→	B
A	→	B
A	→	D
B	→	C
B	→	C
B	→	D
D	→	C

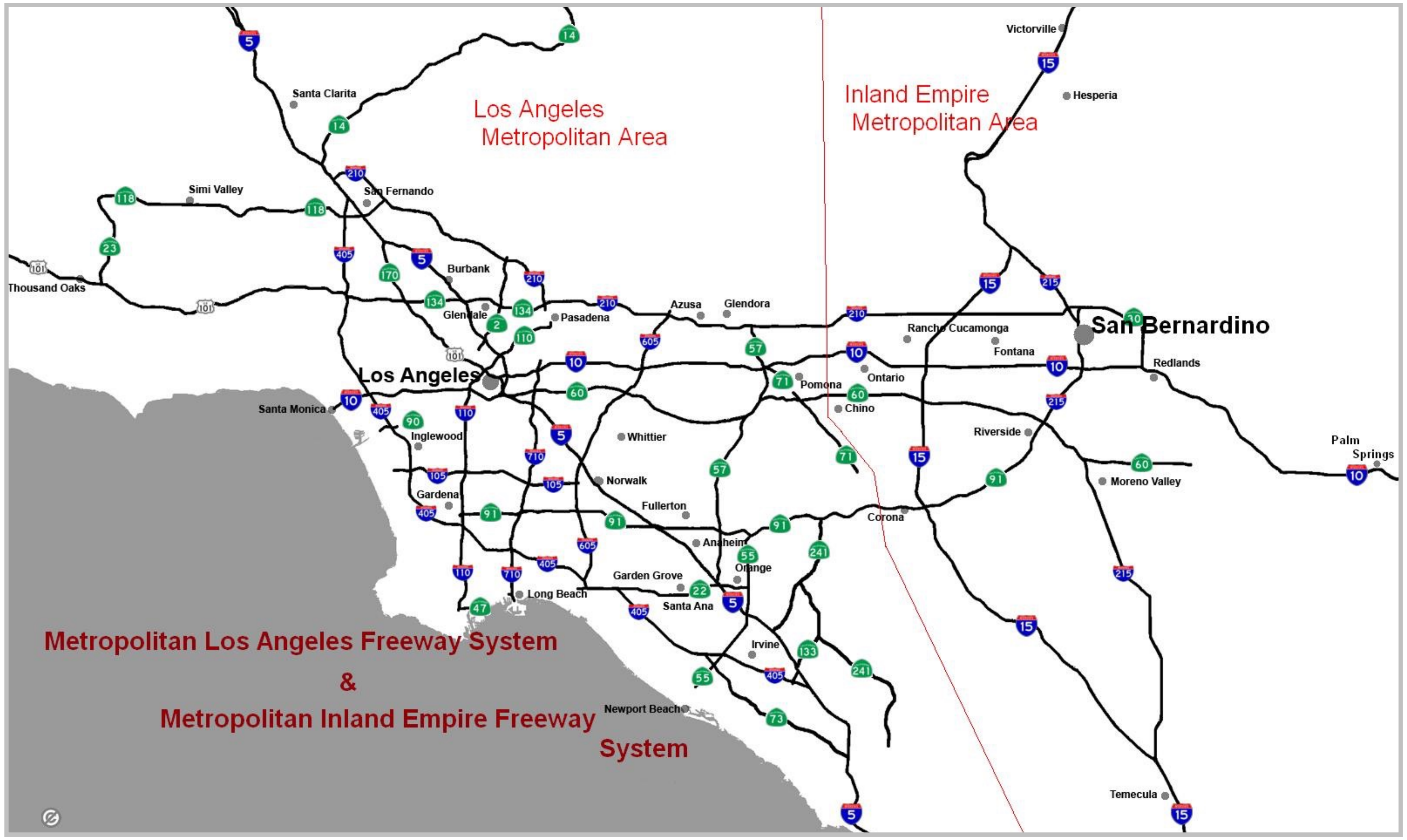


Graphs

represent relationships

Like highways

- A node is a city
- An edge is a highway from one city to another

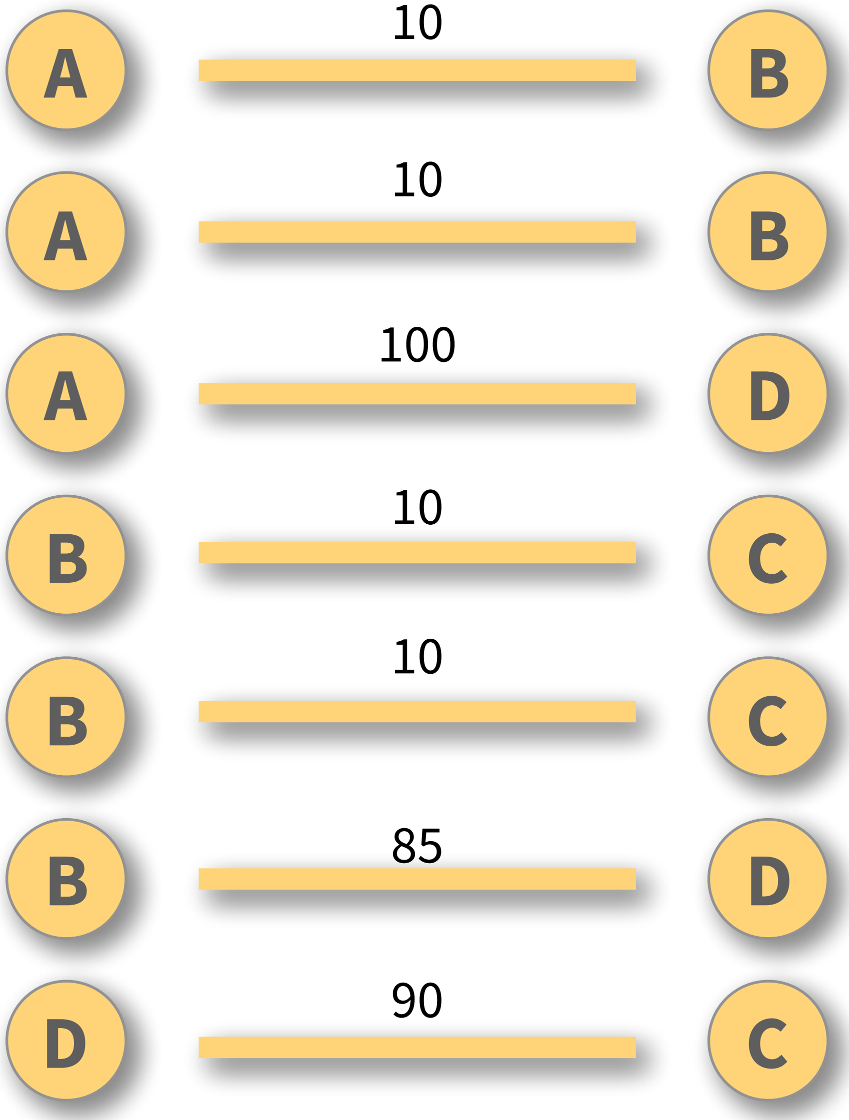


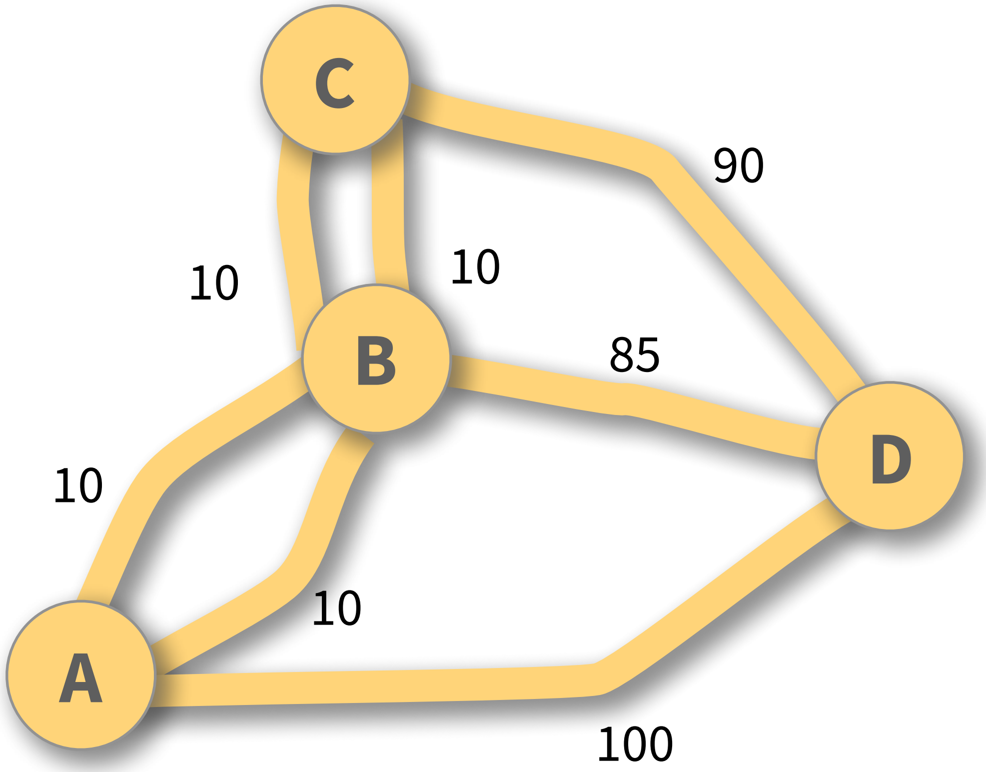
upload.wikimedia.org/wikipedia/commons/4/45/LA–IE_Freeway_System.jpg

Weighted graph

← important vocabulary!

Information (usually "cost") associated with each edge





Graphs

represent relationships

Like flights

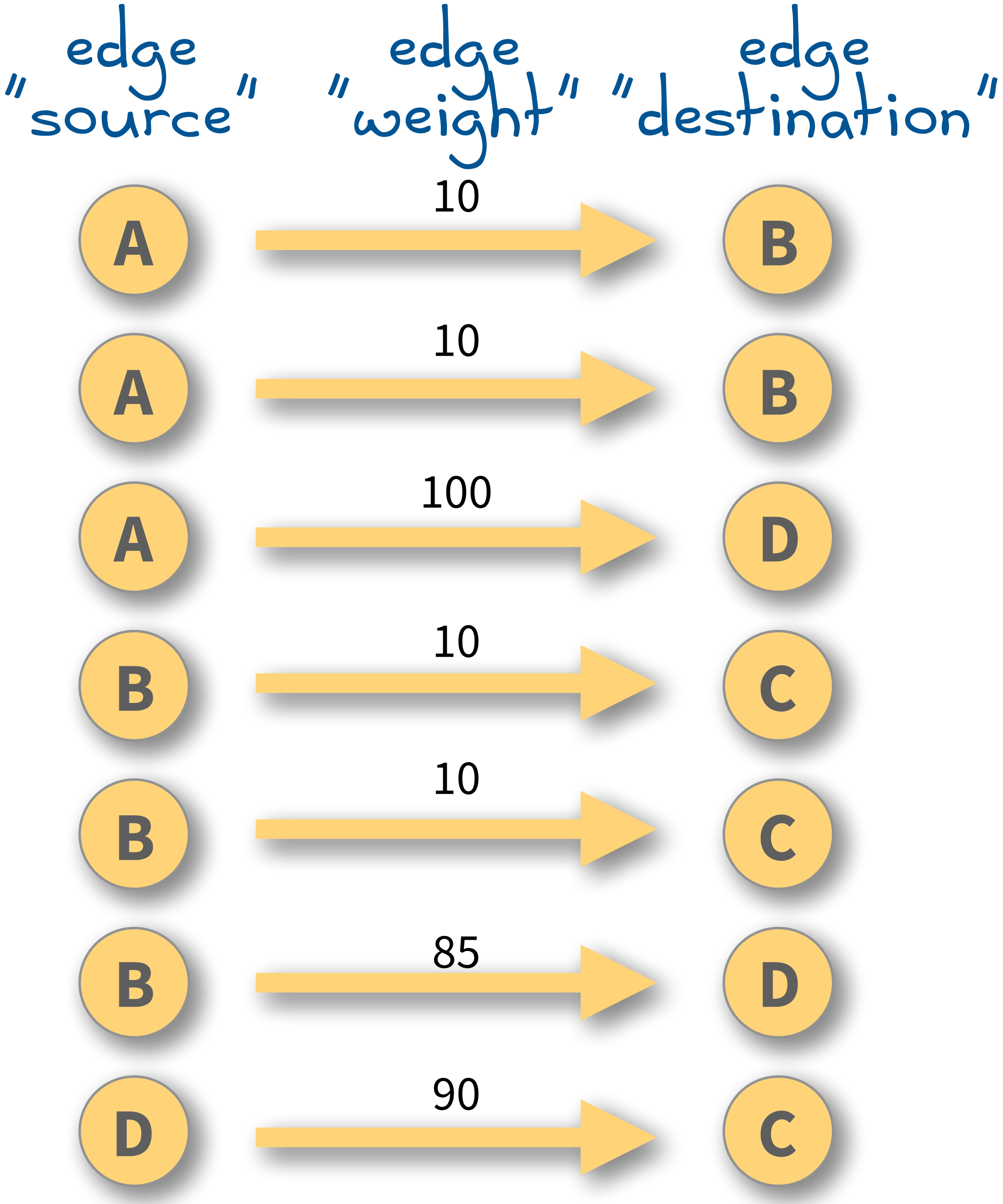
- A node is a city
- An edge is a flight from one city to another

www.expressjet.com/wp-content/uploads/2012/05/ExpressJet_UnitedSystem(February13).jpg

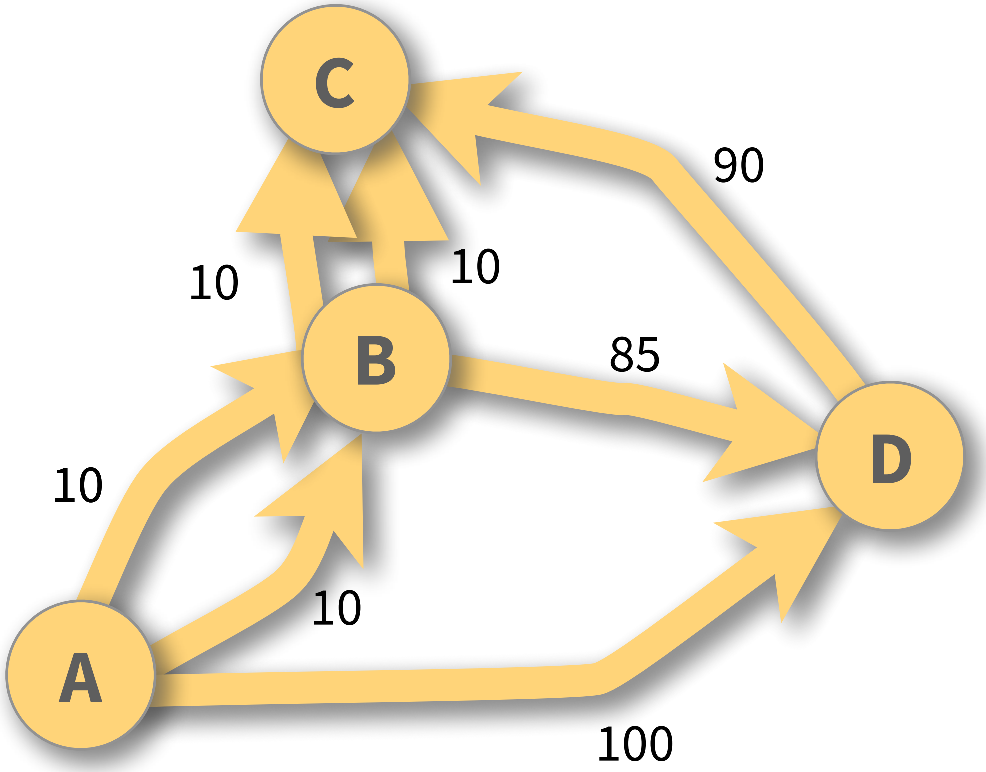
Directed weighted graph

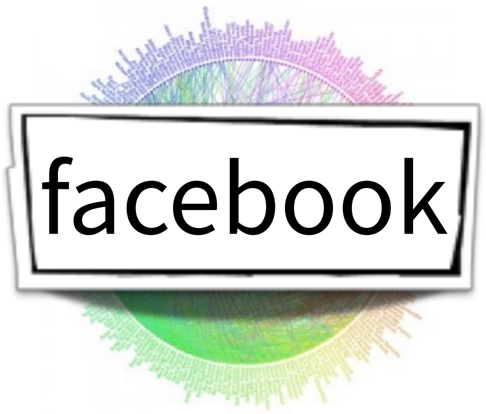
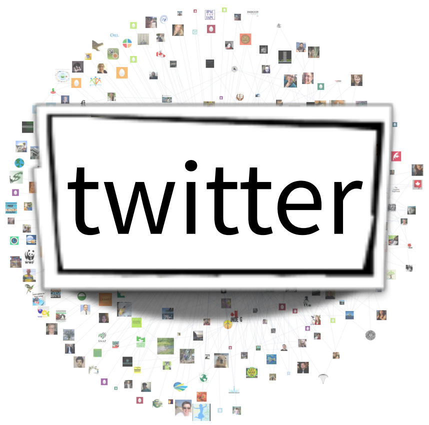
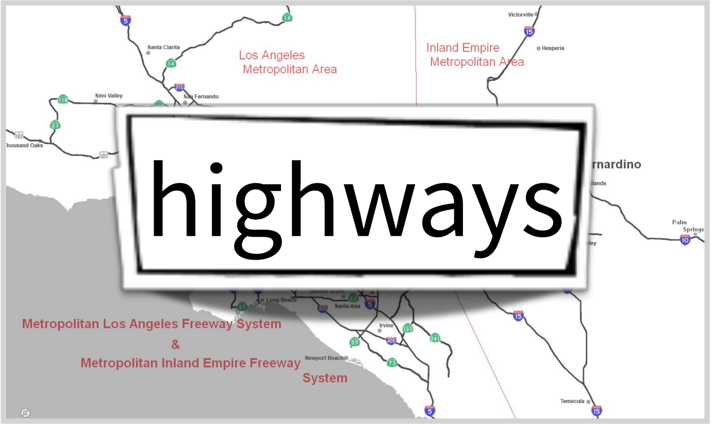
← important vocabulary!

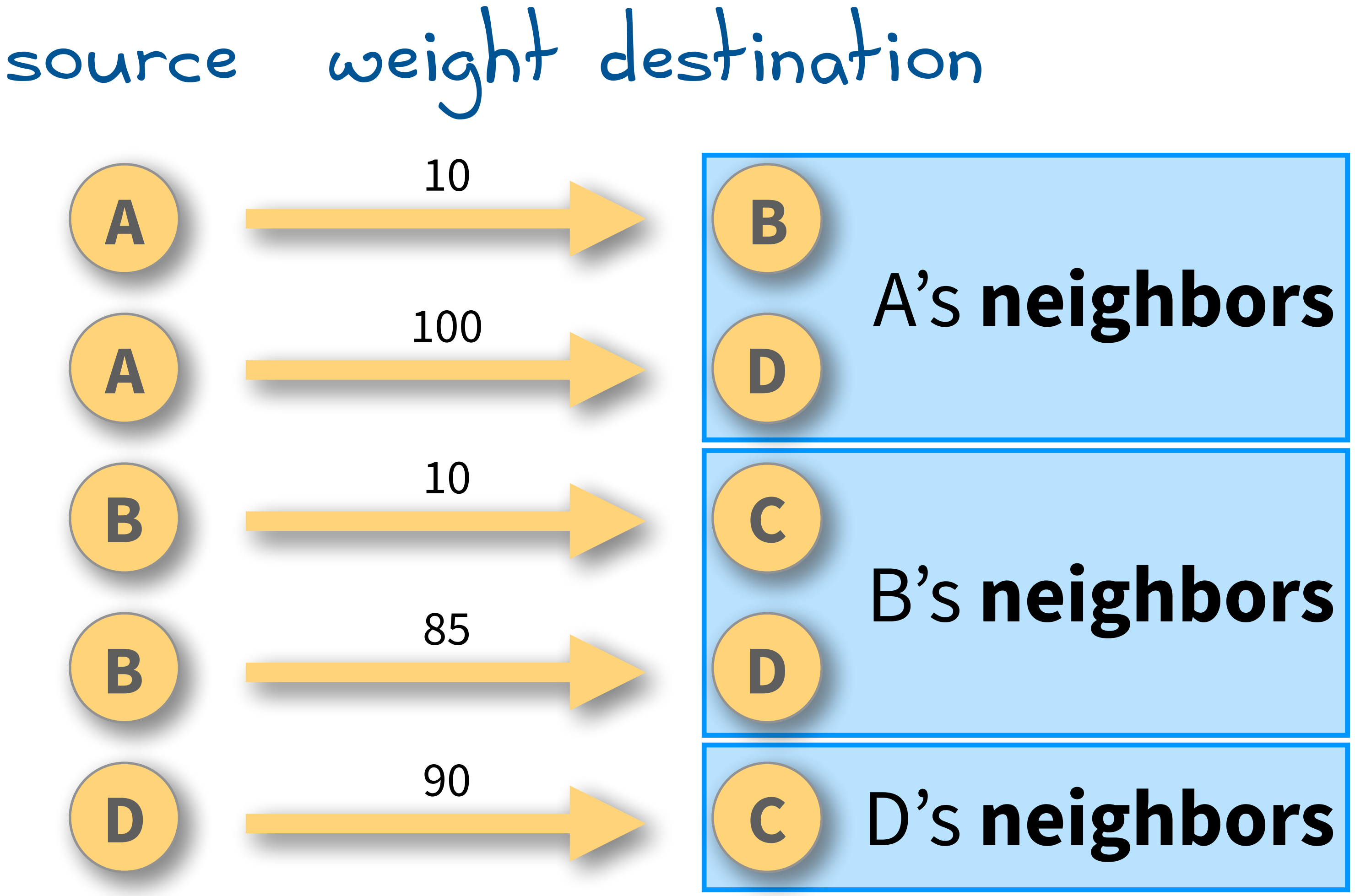
Information (usually “cost”) associated with each edge



edge "source"	edge "weight"	edge "destination"
A	10	B
A	10	B
A	100	D
B	10	C
B	10	C
B	85	D
D	90	C



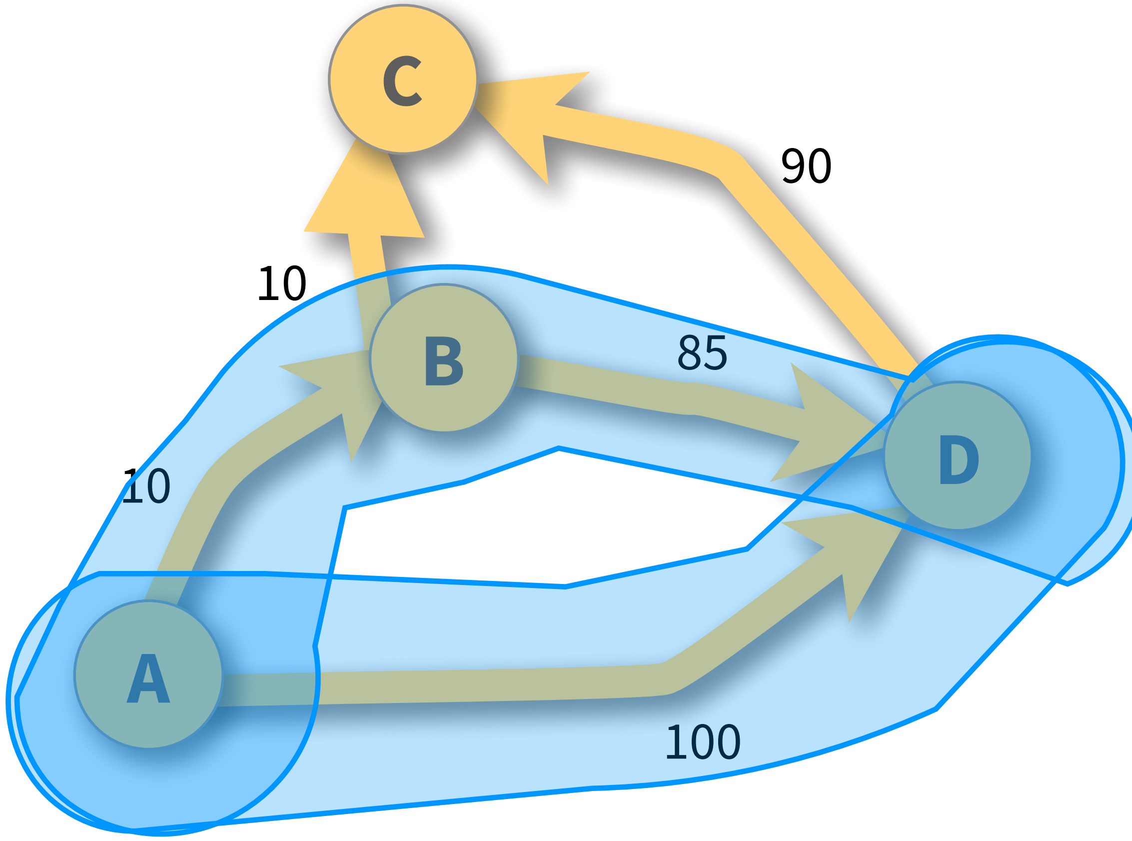
	Undirected	**Directed**
Unweighted	 facebook	 twitter
Weighted	 highways weights = distance, time, cost	 flights weights = distance, time, cost



A’s **neighbors**

B’s **neighbors**

D’s **neighbors**



D has one **adjacent** edge.
C is **adjacent** to D.

important vocabulary!

There are two **paths** from A to D.

C is **reachable** from A.

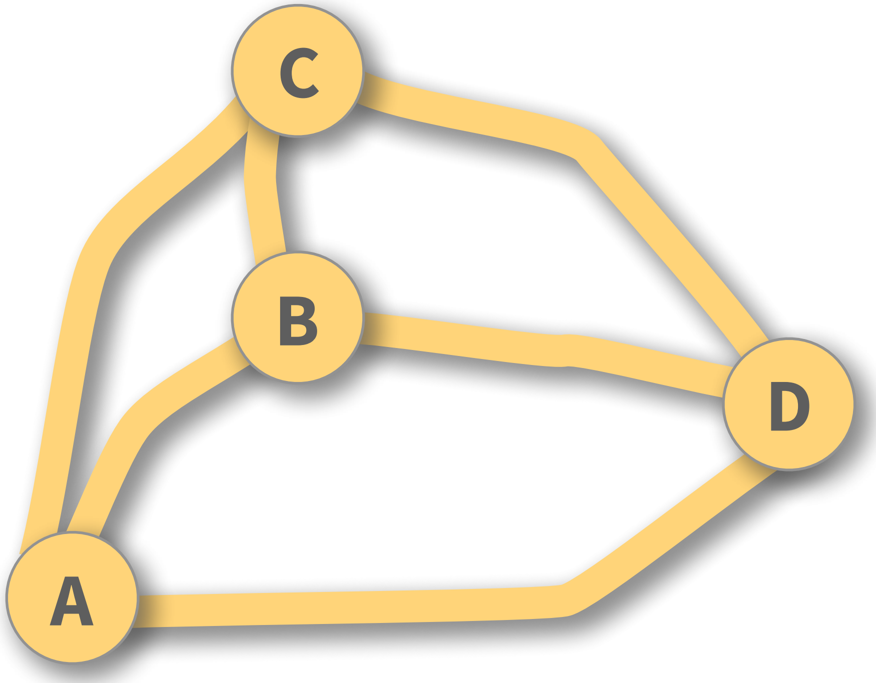
Complete graph

← important vocabulary!

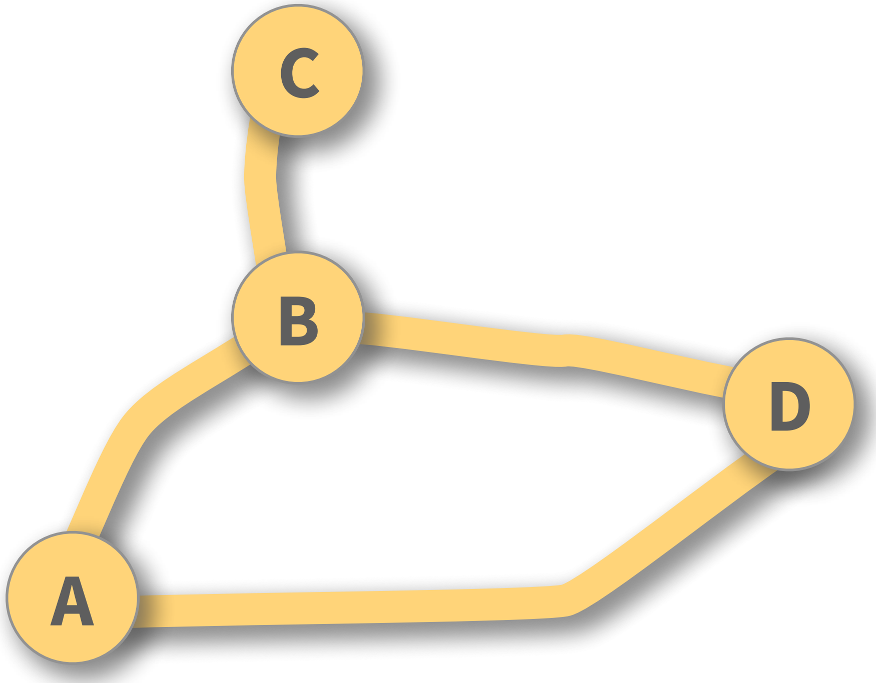
There is an edge between each pair of nodes.

In other words, each node is adjacent to every other node.

To be true in a directed graph, the edges must go in both directions.



complete



not complete

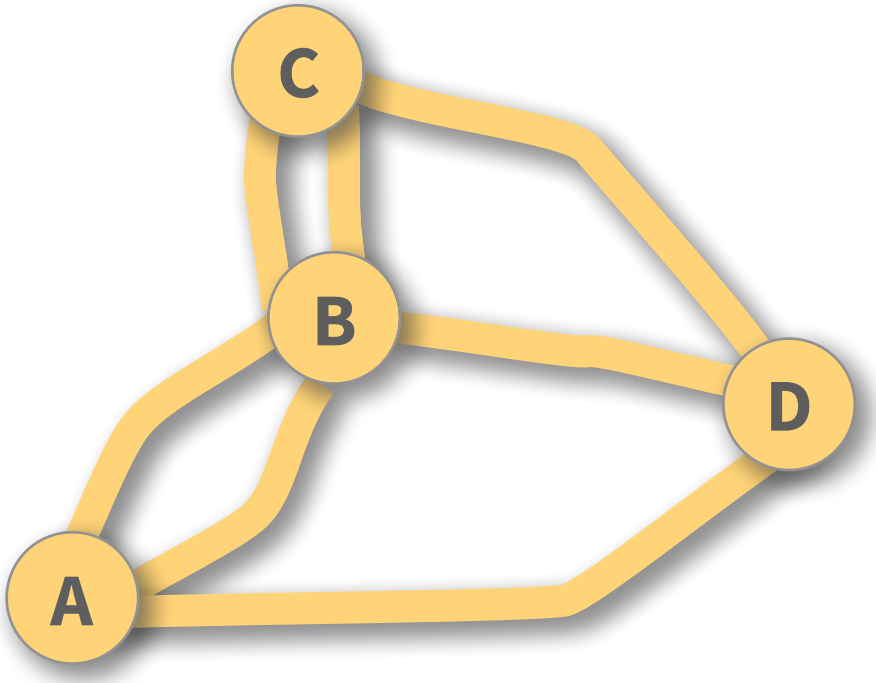
Connected graph

← important vocabula

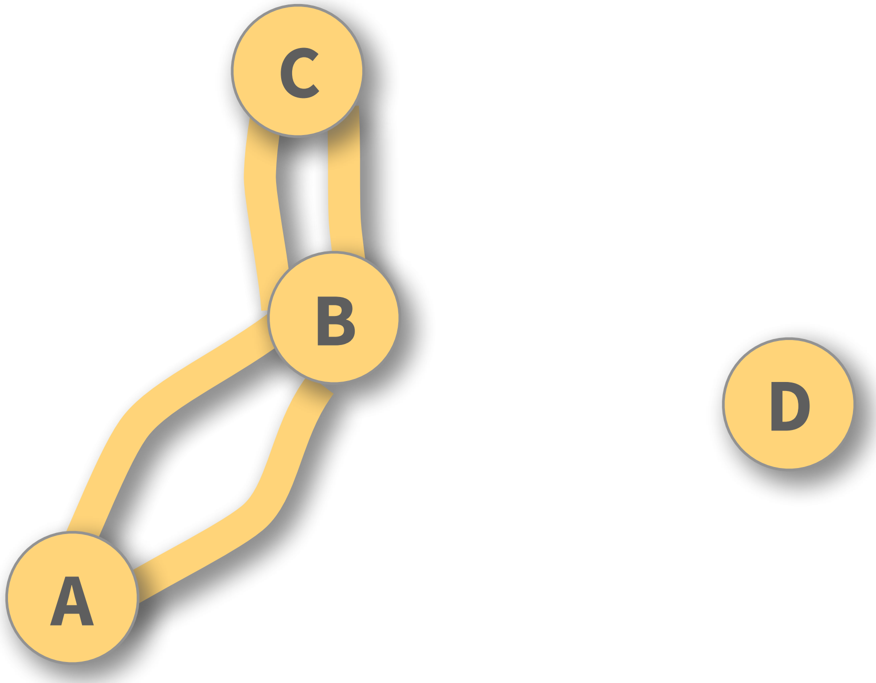
There is a path between each pair of nodes.

In other words, each node is reachable from every other node.

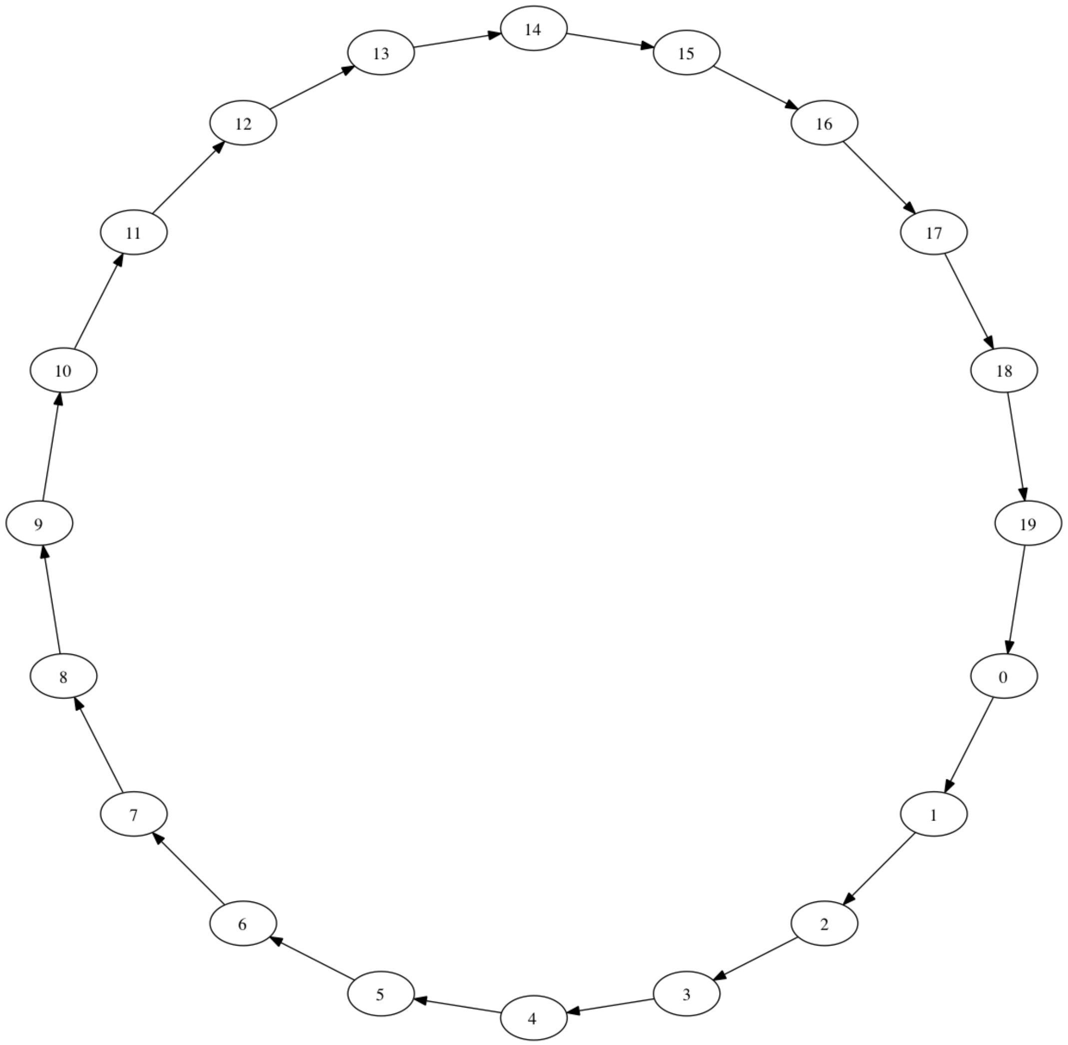
If this is true in a directed graph, the graph is "strongly connected".

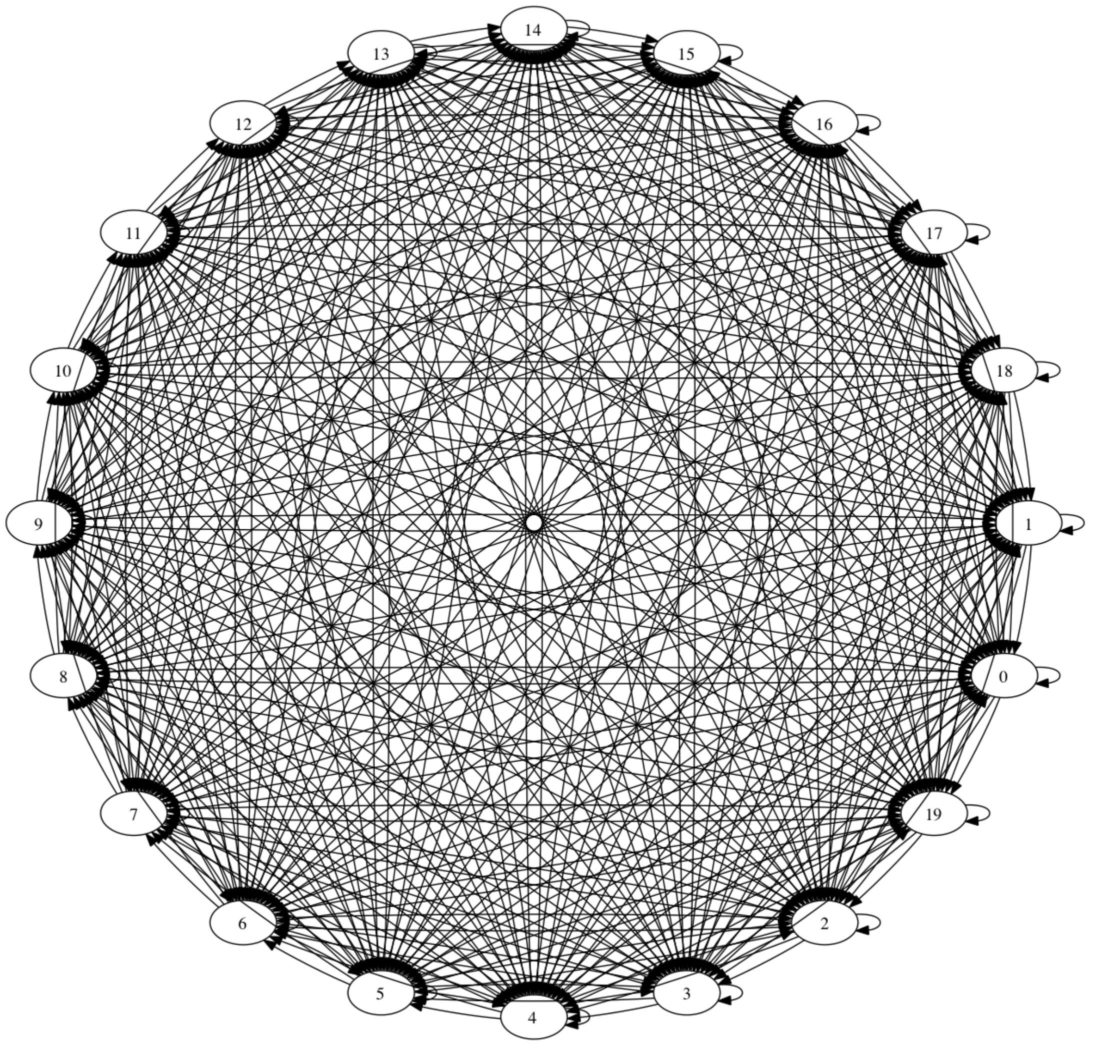


connected



not connected





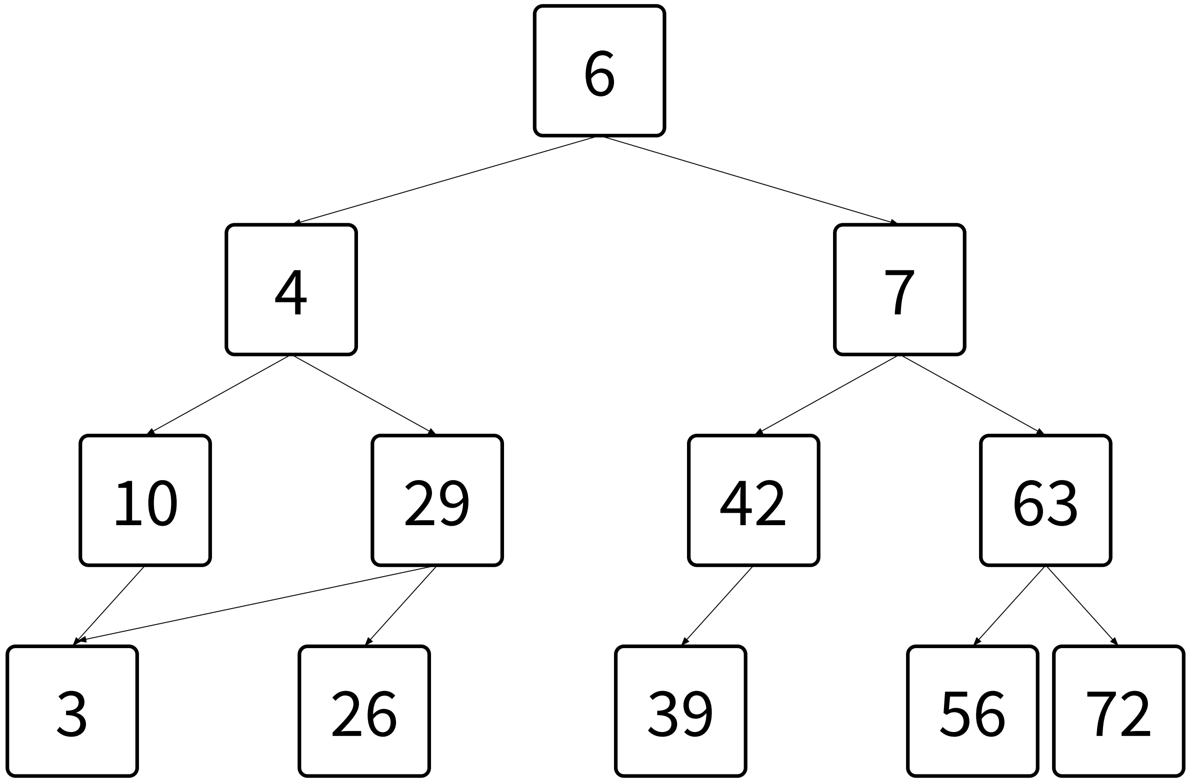
a **sparse** graph has few edges

a **dense** graph has many edges

important vocabulary!

Cycle

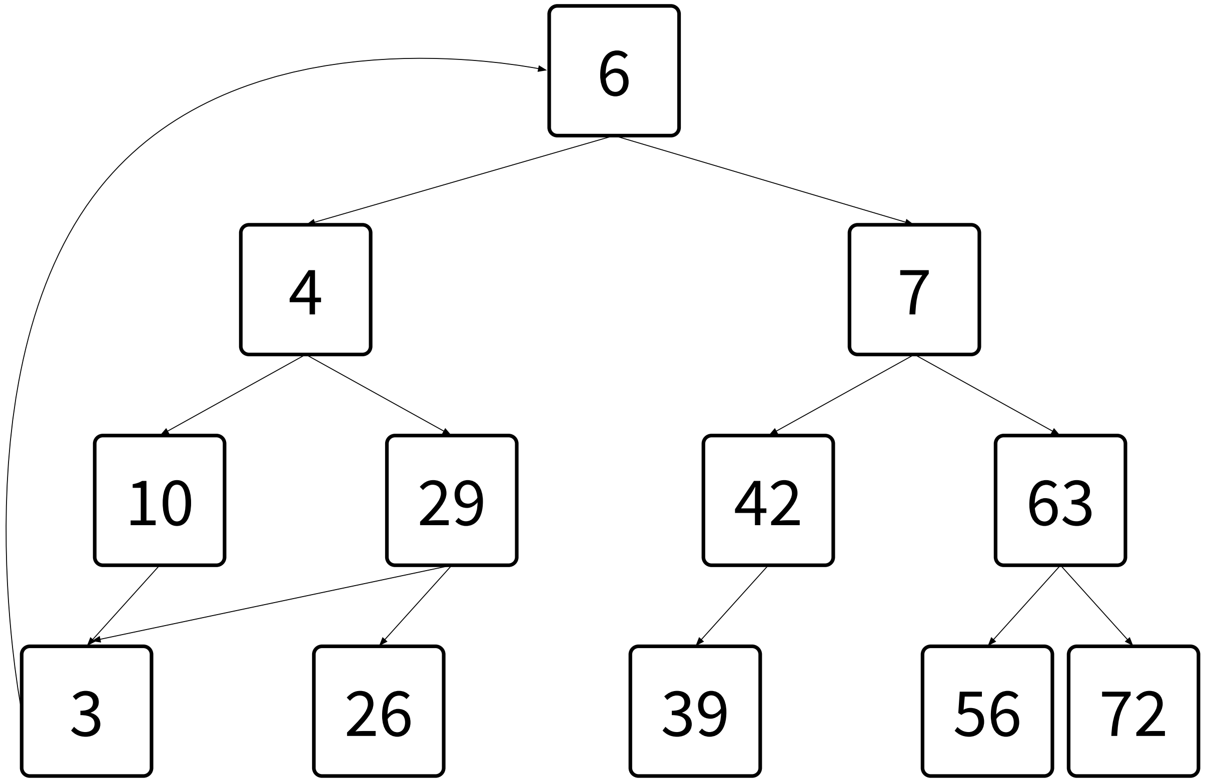
← important vocabulary!



acyclic

there are

a finite number of paths to a node



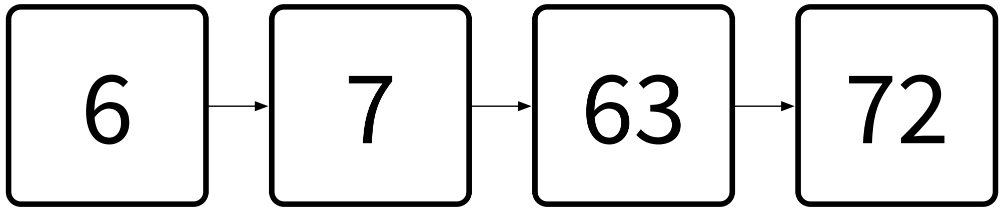
cyclic

there may be

an infinite number of paths to a node

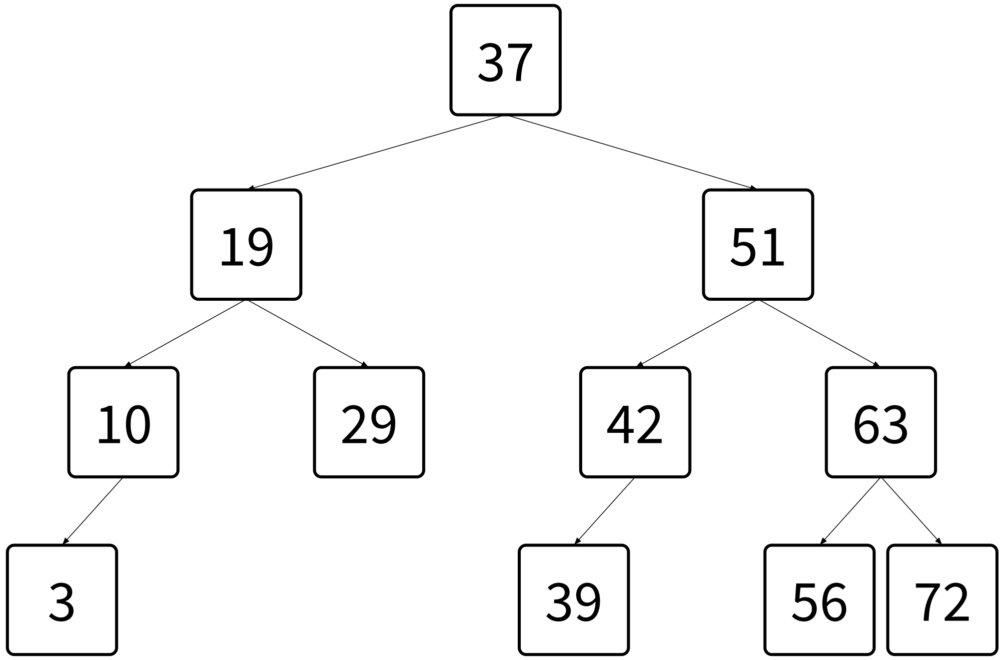
We’ve seen graphs before!

a linked list is a graph; a tree is a graph



linked list

connected, directed, acyclic graph



tree

connected, directed, acyclic graph