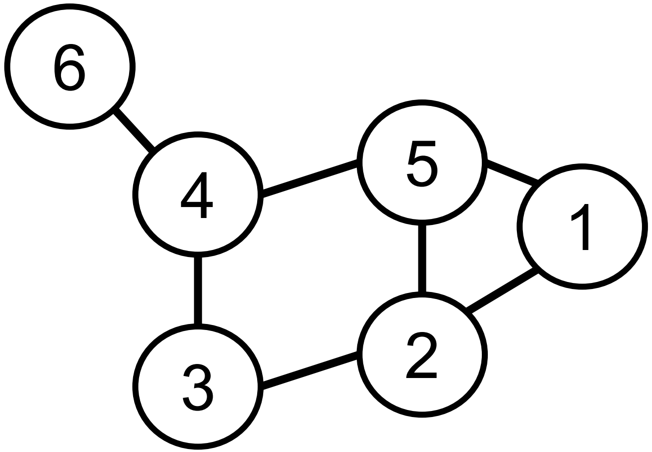
Introduction to Graph Theory

What is a graph?

What is a graph?



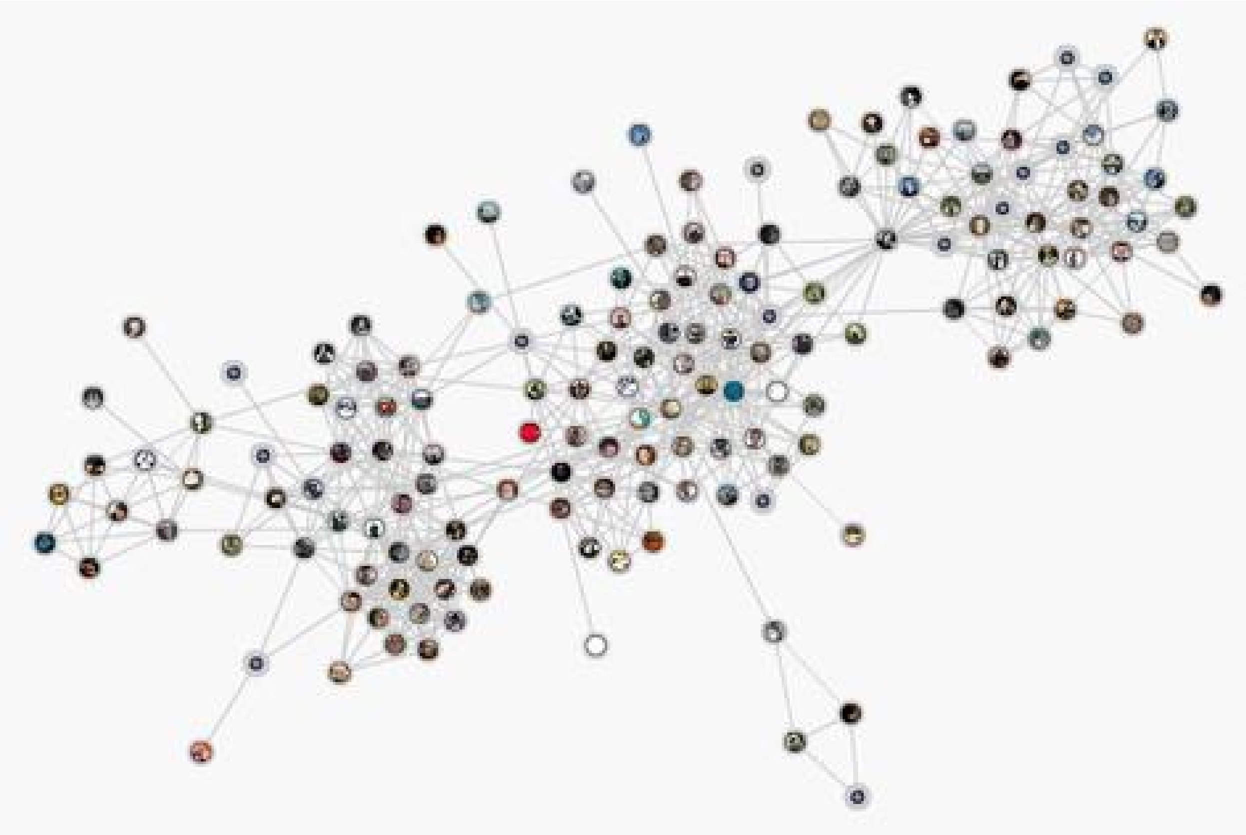
A graph is a network. An abstraction of relationships between data points.

Data points are ***nodes*** (or ***vertices***) in the graph, and the connections between them are ***edges***.

Maybe the nodes are people and the edges represent friendship. Here's a whole book on analyzing ***social networks*** using graphs.

Graphs can represent many, many other kinds of data as well.

Example: Facebook



Example: [Bible verses]

Example: [Game of Thrones]

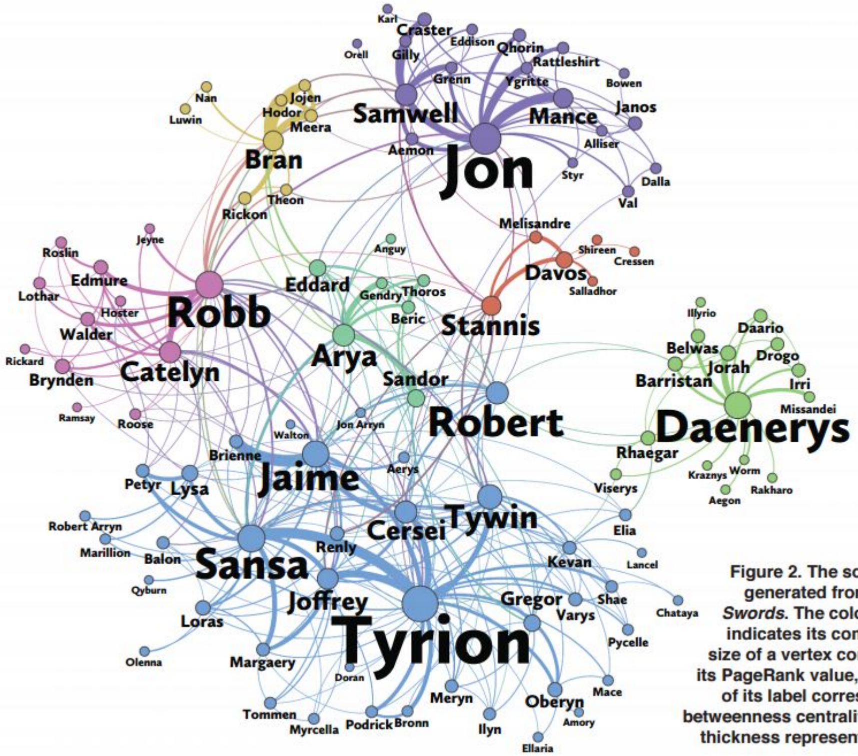
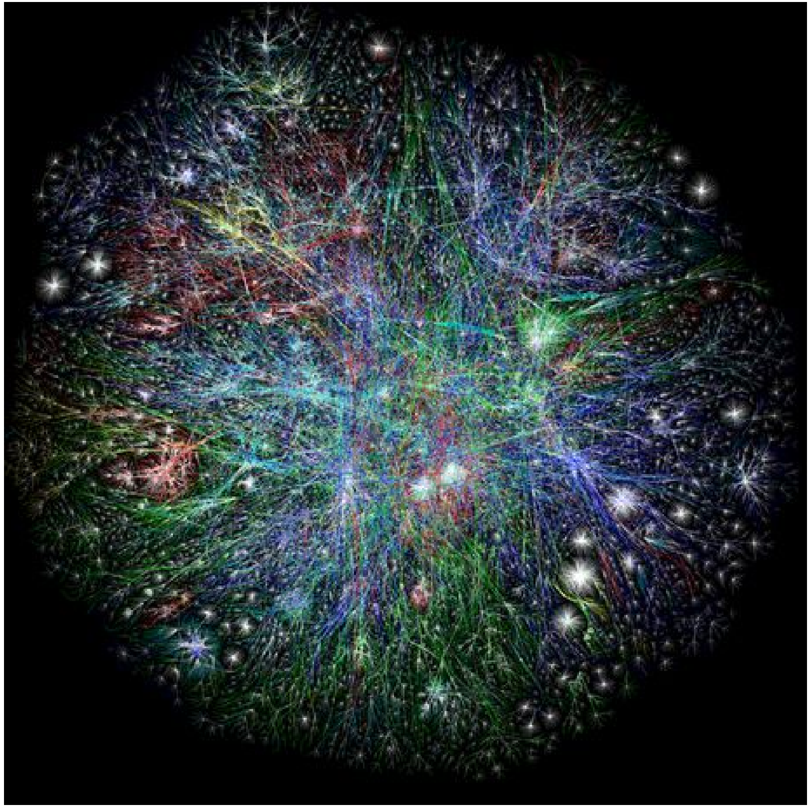


Figure 2. The social network generated from *A Storm of Swords*. The color of a vertex indicates its community. The size of a vertex corresponds to its PageRank value, and the size of its label corresponds to its betweenness centrality. An edge's thickness represents its weight.

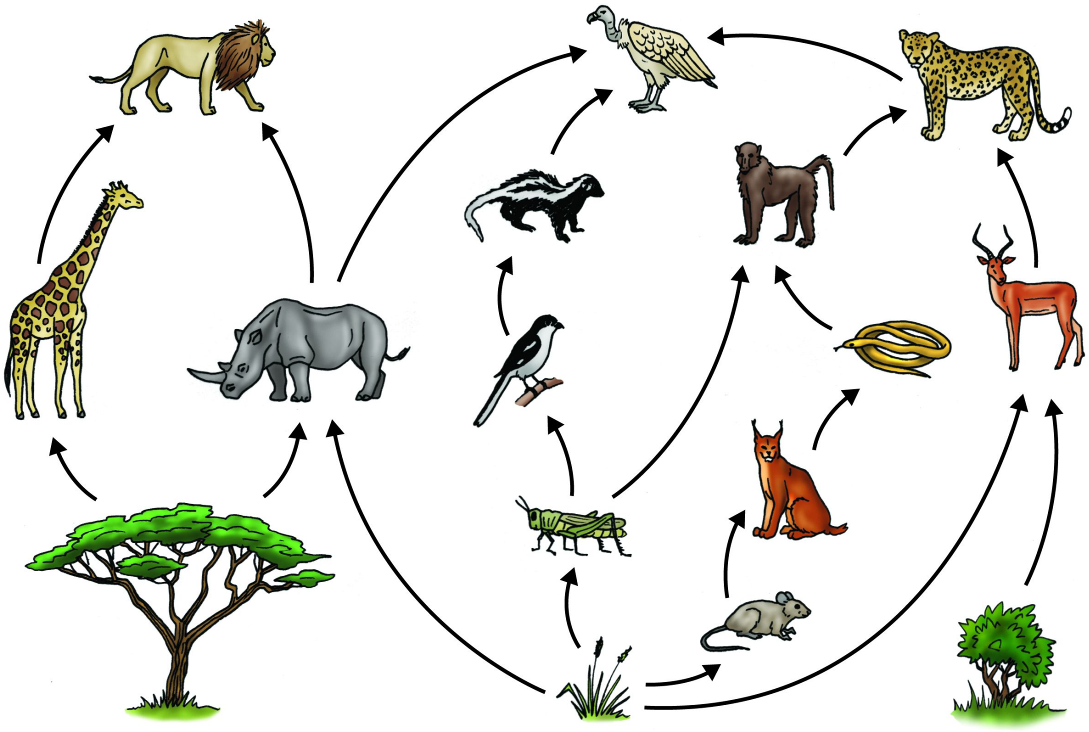
Example: London Subway

Example: London Subway

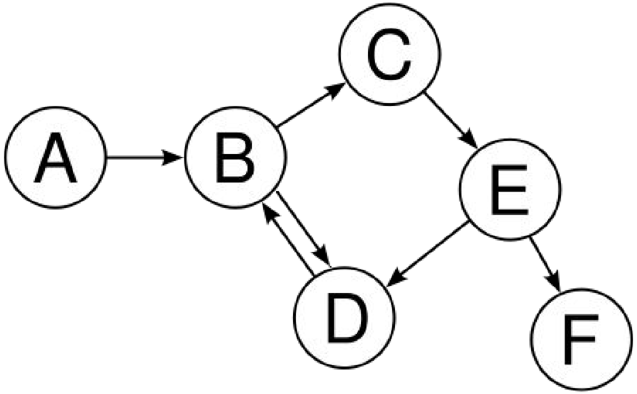
Example: The Internet

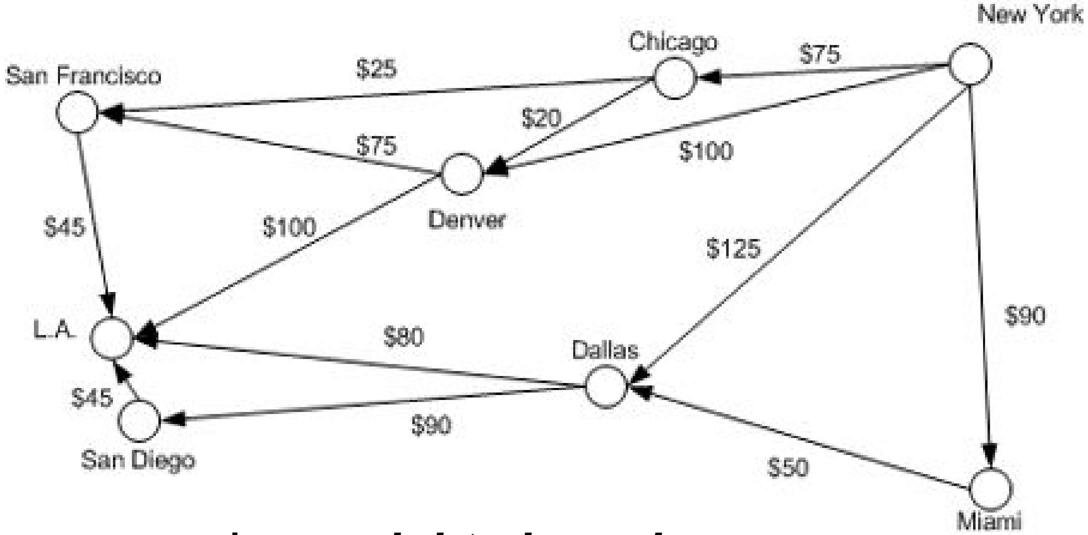


Example: [Food web]



What kind of relationships can we represent?





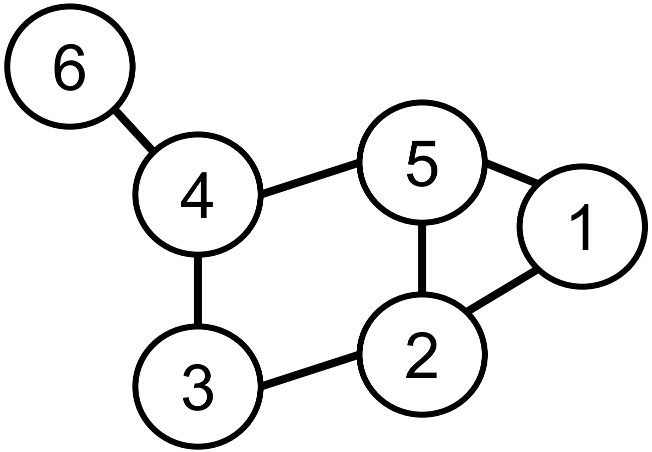
In a ***directed graph***, edges represent one-way relationships (e.g. Twitter followers, phone calls)

In a ***weighted graph***, edges also have a number (usually some kind of ***cost***) associated with them.

Pair discussion:

Can you think of something that could be represented by an ***undirected weighted*** graph?

Graph terminology



Graphs have *vertices* (*nodes*) and *edges*

`V = {1, 2, 3, 4, 5, 6}`

`E = {(1, 2), (1, 5), (2, 5), (2, 3), (3, 4), (4, 5), (4, 6)}`

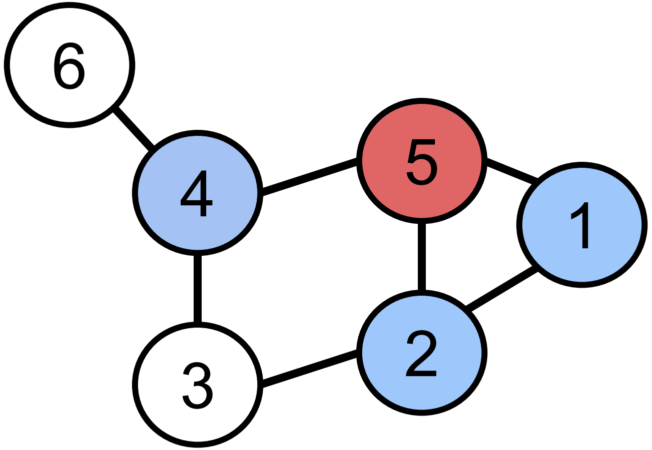
`G = (V, E)`

`|V| = order`

`|E| = size`

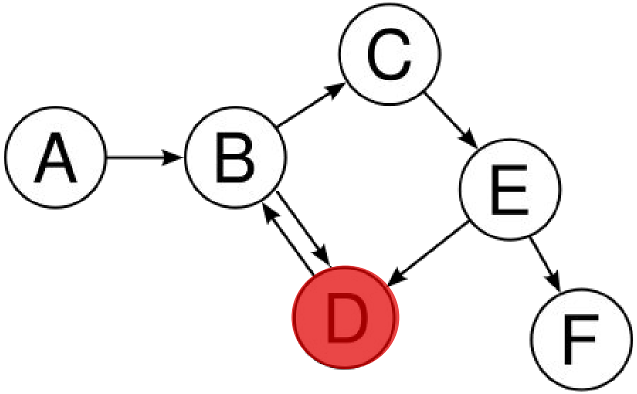
Graph terminology

- **Neighbors** of node N: nodes directly connected to N
- **Degree** of node N: number of neighbors of N



Graph terminology

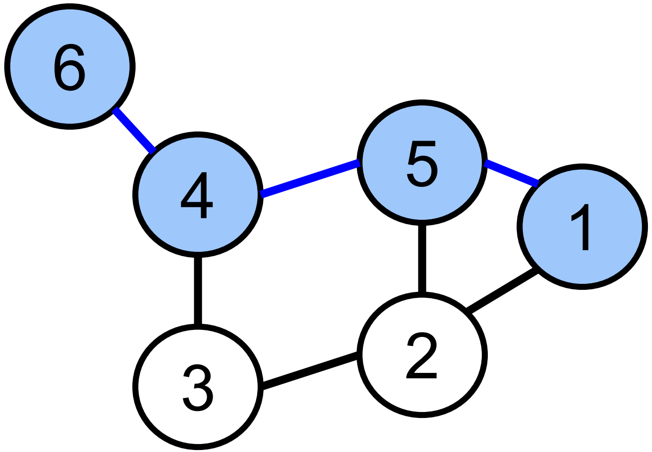
- Directed graphs have **in-degree** (number of incoming edges) and **out-degree** (number of outgoing edges)



Node **D** has an in-degree of 2 and an out-degree of 1

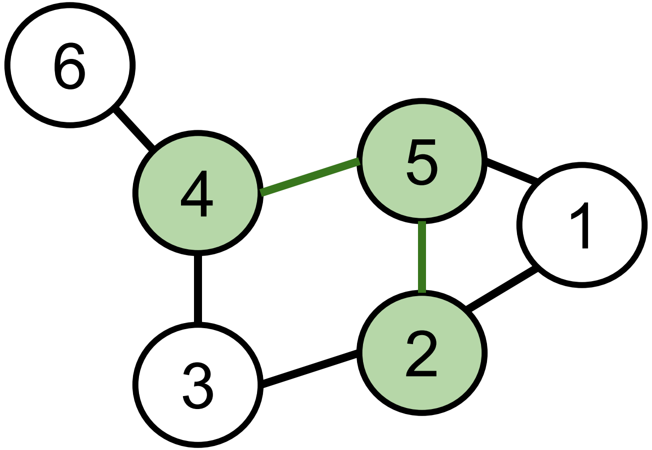
Graph terminology

- **Path** from N to M: series of unique nodes and edges that connect N to M



Graph terminology

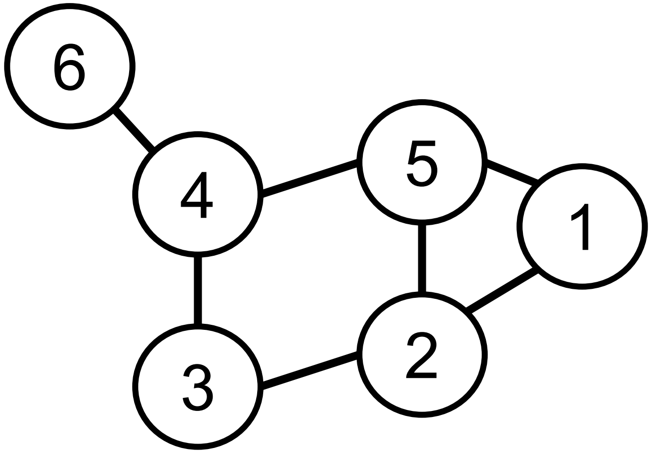
- **Subgraph**: subset of nodes and their edges



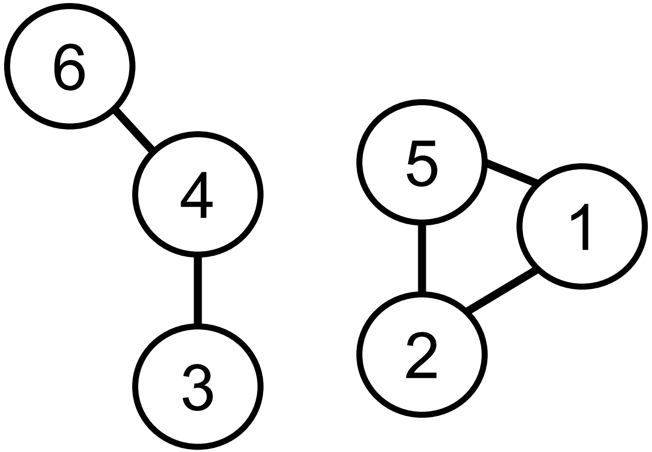
The green nodes & edges are a subgraph of the full graph

Graph terminology

- **Connected graph**: a graph with a path from every node to every other node



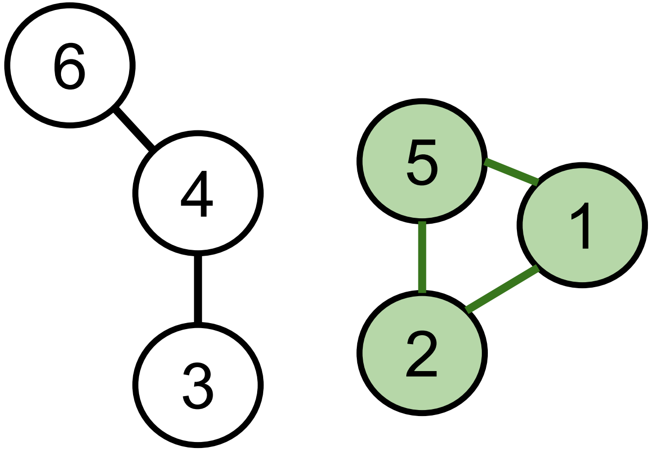
Connected.



Not connected.

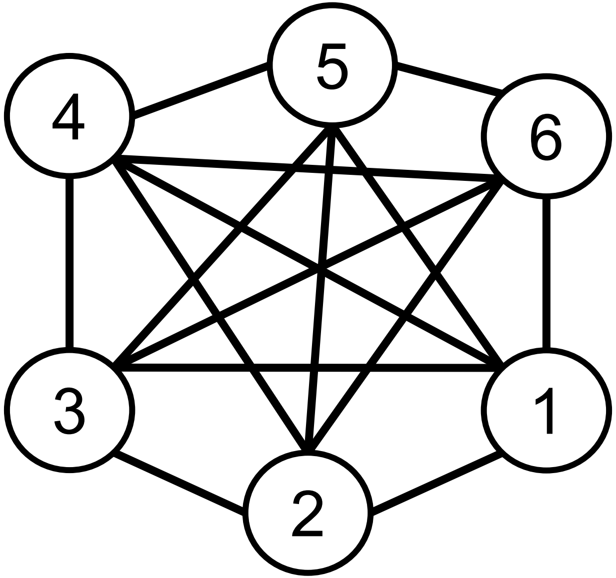
Graph terminology

- **Connected component**: a subgraph that is connected



Graph terminology

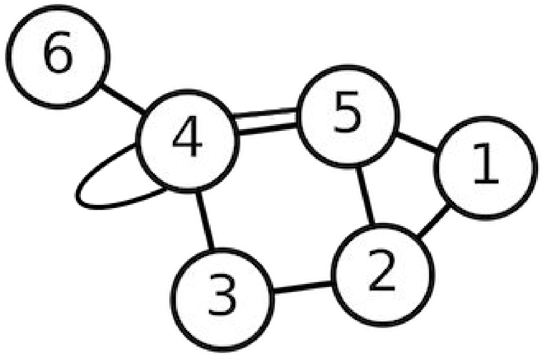
- **Complete graph**: a graph with an edge from every node to every other node

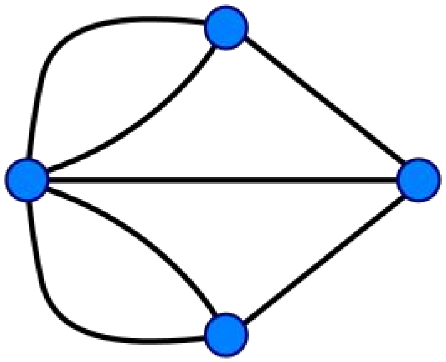


Graph terminology

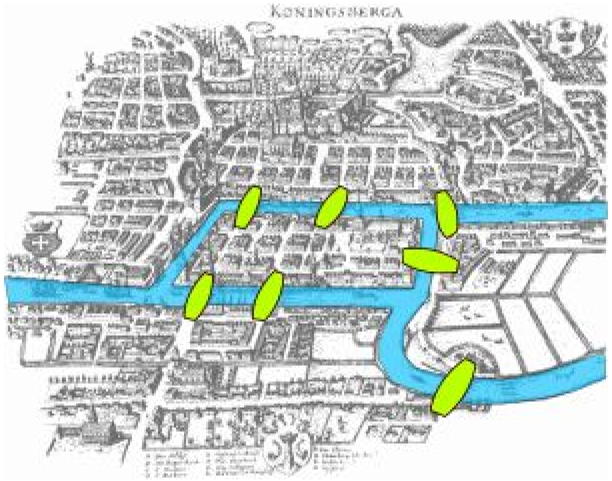
A ***simple graph*** has no loops (no edges connecting a node to itself) and no more than one edge directly connecting two nodes.

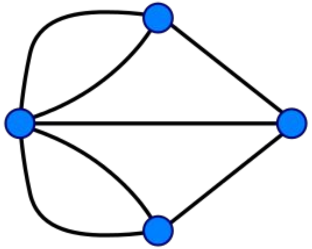
The examples below are ***non-simple graphs***





The Seven Bridges of Königsberg





Fun With Immanuel and Leonhard

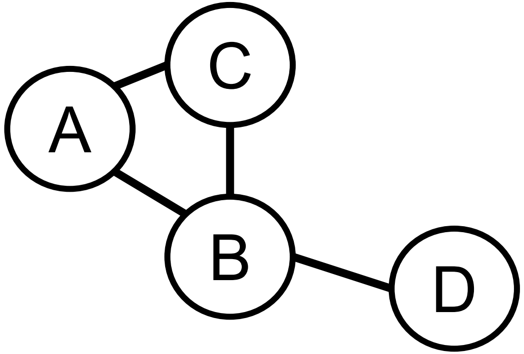
See Kant walk.

Walk, Kant, walk!

“Come home, Kant,” cried Euler

Hence, the birth of graph theory.

How do we represent graphs in a computer?



Edge list

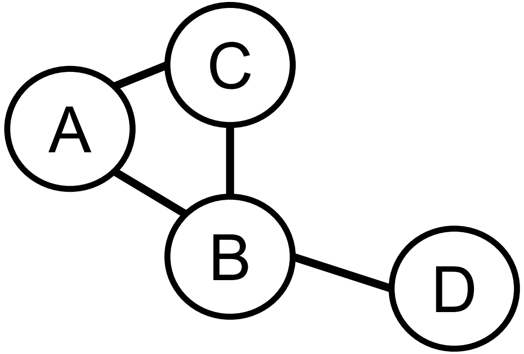
```
[(A,B), (A,C), (B,C), (B,D)]
```

Adjacency list

```
{A: [B, C],
 B: [A, C, D],
 C: [A, B],
 D: [B]}
```

Pop quiz, hotshot: how much space does an adjacency list take up?

How do we represent graphs in a computer?



Edge list

```
[(A,B), (A,C), (B,C), (B,D)]
```

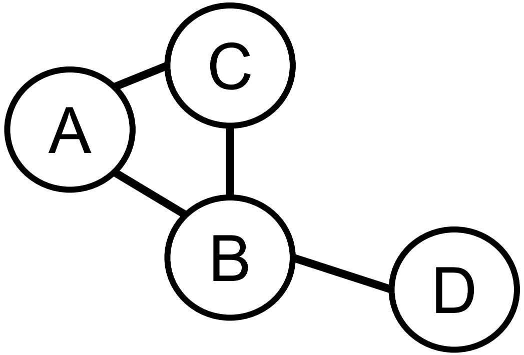
Adjacency list

```
{A: [B, C],
 B: [A, C, D],
 C: [A, B],
 D: [B]}
```

Pop quiz, hotshot: how much space does an adjacency list take up? **O(|V| + |E|)**

How do we represent graphs in a computer?

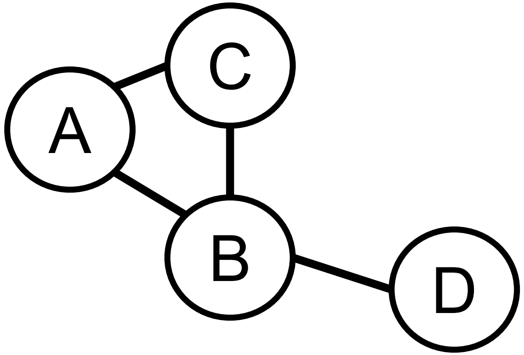
Adjacency matrix (unweighted graph)



	A	*B*	*C*	*D*
A	0	1	1	0
B	1	0	1	1
C	1	1	0	0
D	0	1	0	0

How much space does an adjacency matrix take up?

How do we represent graphs in a computer?

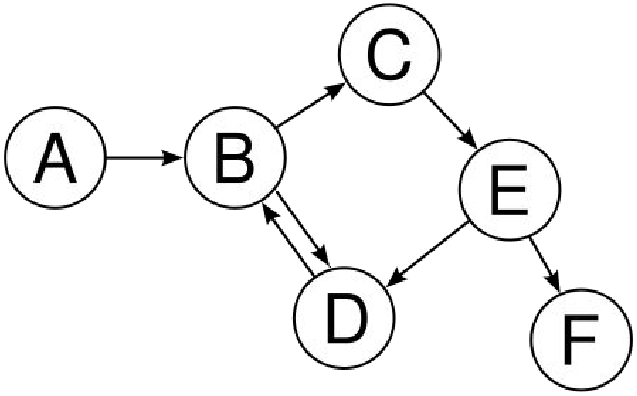


Adjacency matrix (unweighted graph)

	A	*B*	*C*	*D*
A	0	1	1	0
B	1	0	1	1
C	1	1	0	0
D	0	1	0	0

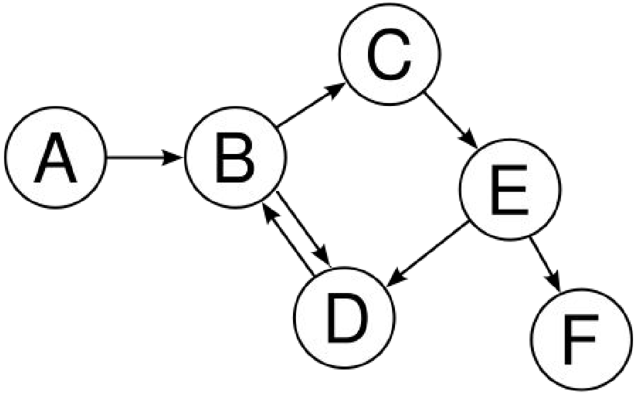
How much space does an adjacency matrix take up? O(|V|^2)

Exercise: Adjacency matrix for a ***directed*** graph



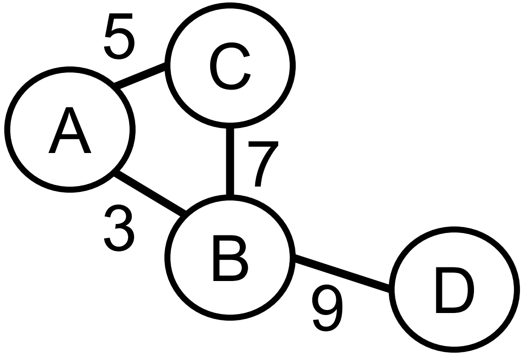
	A	B	C	D	E	F
A	0	_	_	_	_	_
B	_	0	_	_	_	_
C	_	_	0	_	_	_
D	_	_	_	0	_	_
E	_	_	_	_	0	_
F	_	_	_	_	_	0

Exercise: Adjacency matrix for a ***directed*** graph



	A	B	C	D	E	F
A	0	1	0	0	0	0
B	0	0	1	1	0	0
C	0	0	0	0	1	0
D	0	1	0	0	0	0
E	0	0	0	1	0	1
F	0	0	0	0	0	0

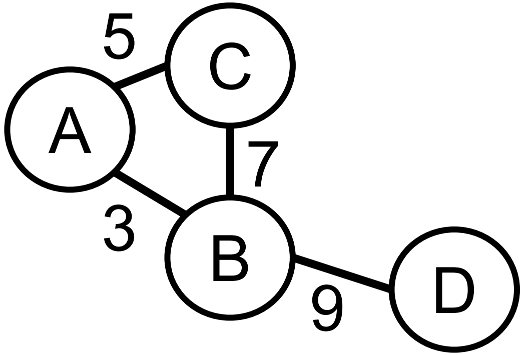
How about an adjacency matrix for a ***weighted*** graph?



Adjacency matrix (weighted graph)

	A	*B*	*C*	*D*
A	_	3	5	_
B	3	_	7	9
C	5	7	_	_
D	_	9	_	_

How about an adjacency matrix for a ***weighted*** graph?



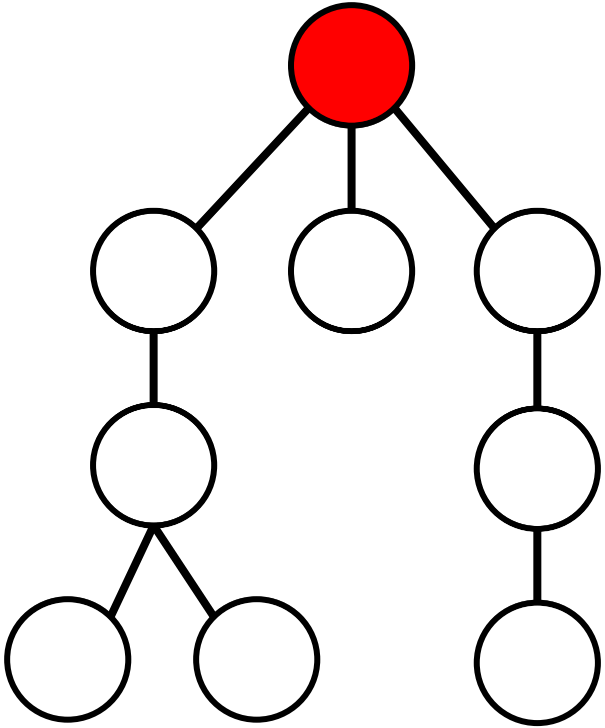
Adjacency matrix (weighted graph)

$$
\begin{array}{ccccc}
 & A & B & C & D \\
A & 0 & 3 & 5 & \infty \\
B & 3 & 0 & 7 & 9 \\
C & 5 & 7 & 0 & \infty \\
D & \infty & 9 & \infty & 0
\end{array}
$$

So uh what do we do with the graphs?

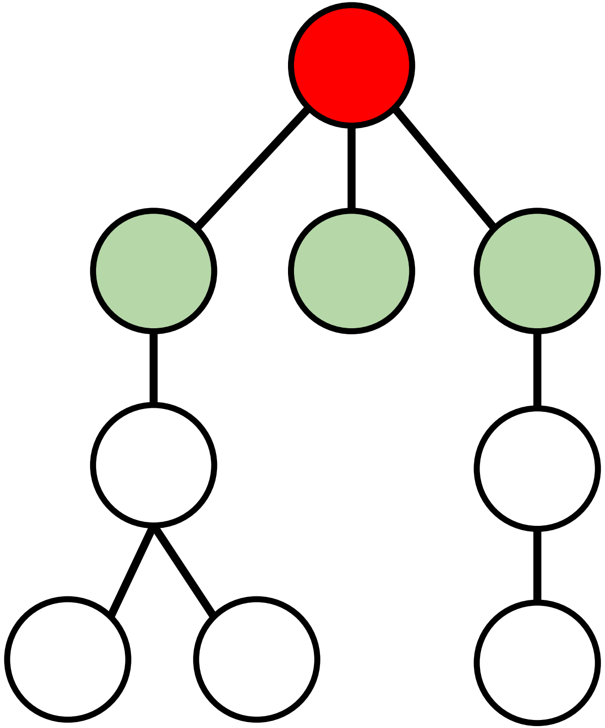
- Community detection (afternoon)
- Node importance (afternoon)
- Traverse the graph (now!)
  - Pick a starting node
  - Find information about the graph local to the starting node (neighbors, neighbors of neighbors, etc.)
  - Find paths to a target node (e.g. Kevin Bacon)

Traversing Graphs: Breadth First Search



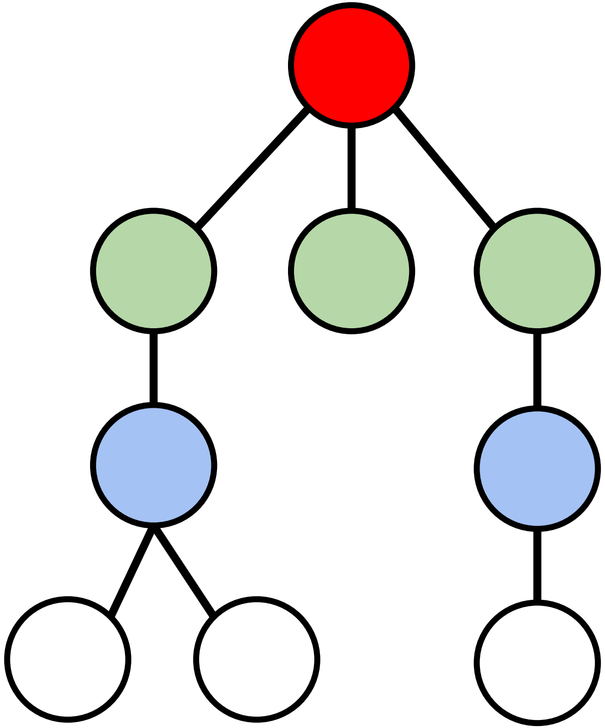
- Starting node

Traversing Graphs: Breadth First Search



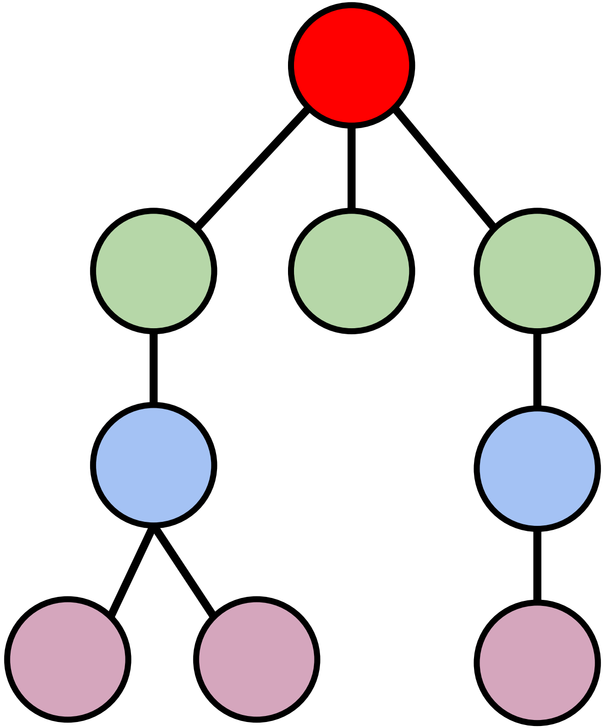
- Starting node
- First set of visited nodes

Traversing Graphs: Breadth First Search



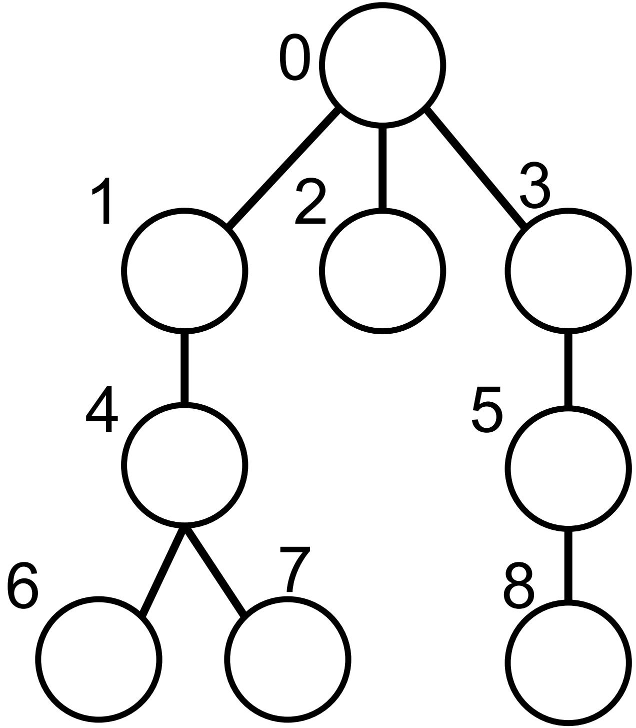
- Starting node
- First set of visited nodes
- Second set of visited nodes

Traversing Graphs: Breadth First Search



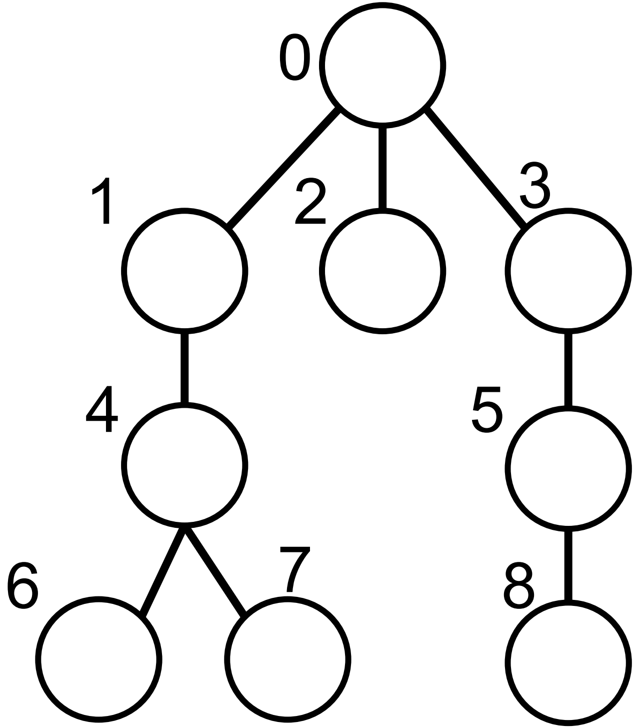
- Starting node
- First set of visited nodes
- Second set of visited nodes
- Third set of visited nodes

Traversing Graphs: Breadth First Search



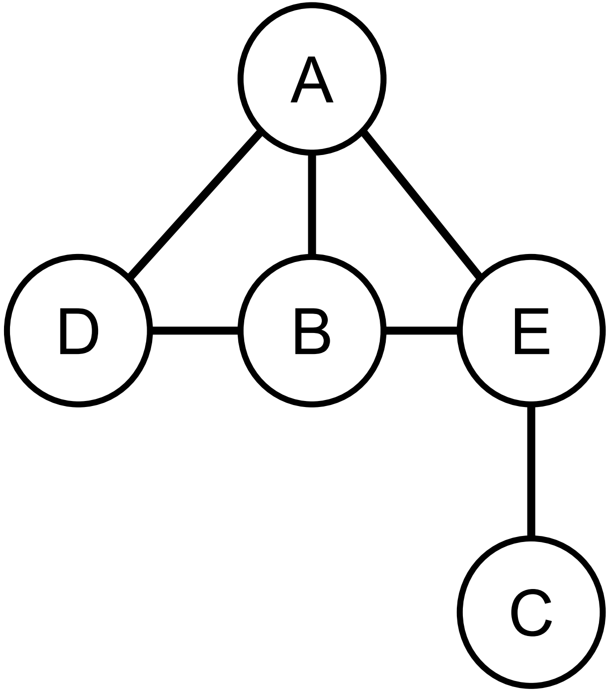
- Start at node 0
- Put all neighbors of 0 into a list Q of nodes to visit
- Remove each neighbor from Q on a first-in-first-out (FIFO) basis
  - Add it to the set of visited nodes V
  - Add all of its neighbors to the end of the list Q

Traversing Graphs: Breadth First Search



- Given a start and end node, the first path found by BFS is guaranteed to be the shortest path
- Q could contain many, many neighbors. BFS is memory intensive.

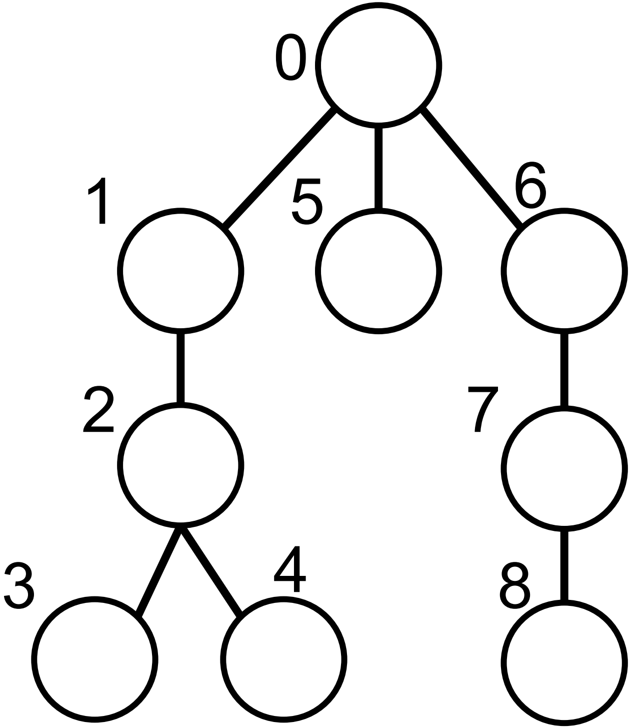
Example: Find the shortest path length from A to C



BFS Pseudocode:

- Create empty queue Q
- Create empty set V (visited nodes)
- Add the tuple (A,0) to Q
- While Q is not empty:
  - Remove first element from the queue, it will be the tuple (N, d) representing node N a distance d from A
  - if N is the desired end node: return (N, d)
  - if N is not in V;
    - add N to V
    - add every neighbor of N to Q with distance d+1

Traversing Graphs: Depth First Search



- Pick a starting node (0)
- Pick a neighbor node, add it to your set of visited nodes V
- Pick a neighbor node of that neighbor node, add it to V
- Repeat until you find no neighbors that you haven’t visited, then backtrack until you find a node with a fresh neighbor
- Repeat until you have visited every node

Appendix: time complexity

Adjacency matrix
(unweighted graph)

$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 1 & 1 & 0 \\ B & 1 & 0 & 1 & 1 \\ C & 1 & 1 & 0 & 0 \\ D & 0 & 1 & 0 & 0 \end{array}$$

Adjacency list

```
{A: [(B,3), (C,5)],
 B: [(A,3), (C,7), (D,9)],
 C: [(A,5), (B,7)],
 D: [(B,9)]}
```

How many steps does it take to perform the following operations?
“Is A a neighbor of B”
“How many neighbors of A?”
“Add a node”

Appendix: time complexity

Adjacency matrix
(unweighted graph)

	A	*B*	*C*	*D*
A	0	1	1	0
B	1	0	1	1
C	1	1	0	0
D	0	1	0	0

Adjacency list

```
{A: [(B,3), (C,5)],
 B: [(A,3), (C,7), (D,9)],
 C: [(A,5), (B,7)],
 D: [(B,9)]}
```

How many steps does it take to perform the following operations?

	Adjacency matrix	Adjacency list
"Is A a neighbor of B"	**O(1)**	**O(|V|)**
"How many neighbors of A?"	**O(|V|)**	**O(|# of neighbors|)**
"Add a node"	**O(|V|)**	**O(|# of new edges|)**

Appendix: Graph terminology

- **Neighbors** of node N: nodes directly connected to N
- **Degree** of node N: number of neighbors of N
  - Directed graphs have *in-degree* and *out-degree*
- **Path** from N to M: series of unique nodes and edges that connect N to M
- **Complete graph**: a graph with an edge from every node to every other node
- **Connected graph**: a graph with a path from every node to every other node
- **Subgraph**: subset of nodes and their edges
- **Connected component**: a subgraph that is connected