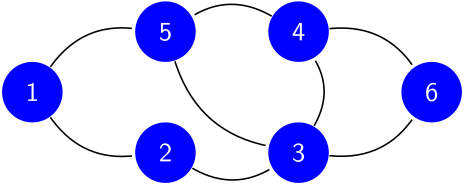
Graph Theory Review

Gonzalo Mateos
Dept. of ECE and Goergen Institute for Data Science and AI
University of Rochester
gmateosb@ece.rochester.edu
http://www.hajim.rochester.edu/ece/sites/gmateos/

January 23, 2025

Basic definitions and concepts

Graphs

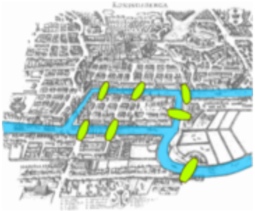


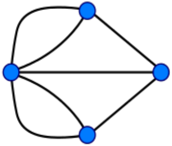
- Graph $G(V, E)$ $\Rightarrow$ A set V of vertices or nodes
  $\Rightarrow$ Connected by a set E of edges or links
  $\Rightarrow$ Elements of E are unordered pairs (u, v), $u, v \in V$
- In figure $\Rightarrow$ Vertices are $V = \{1, 2, 3, 4, 5, 6\}$
  $\Rightarrow$ Edges $E = \{(1, 2), (1, 5), (2, 3), (3, 4), \ldots$
  $(3, 5), (3, 6), (4, 5), (4, 6)\}$
- Often we will say graph G has order $N_v := |V|$, and size $N_e := |E|$

From networks to graphs

- Networks are complex systems of inter-connected components
- Graphs are mathematical representations of these systems

  $\Rightarrow$ Formal language we use to talk about networks





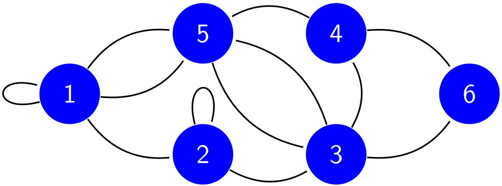
- Components: nodes, vertices $\quad V$
- Inter-connections: links, edges $\quad E$
- Systems: networks, graphs $\quad G(V, E)$

Vertices and edges in networks

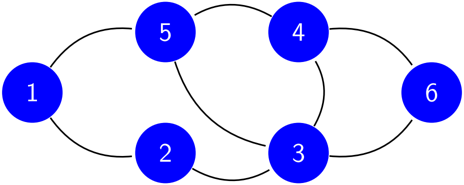
Network	Vertex	Edge
Internet	Computer/router	Cable or wireless link
Metabolic network	Metabolite	Metabolic reaction
WWW	Web page	Hyperlink
Food web	Species	Predation
Gene-regulatory network	Gene	Regulation of expression
Friendship network	Person	Friendship or acquaintance
Power grid	Substation	Transmission line
Affiliation network	Person and club	Membership
Protein interaction	Protein	Physical interaction
Citation network	Article/patent	Citation
Neural network	Neuron	Synapse
⋮	⋮	⋮

Simple and multi-graphs

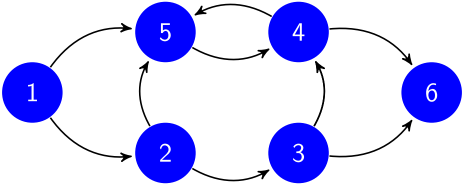
- In general, graphs may have self-loops and multi-edges
  - $\Rightarrow$ A graph with either is called a multi-graph



- Mostly work with simple graphs, with no self-loops or multi-edges



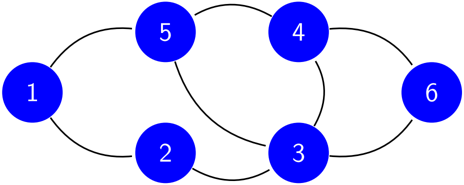
Directed graphs



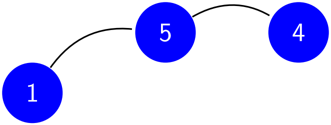
- In directed graphs, elements of E are ordered pairs (u, v), $u, v \in V$
  - $\Rightarrow$ Means (u, v) distinct from (v, u)
  - $\Rightarrow$ Directed edges are called arcs
- Directed graphs often called digraphs
  - $\Rightarrow$ By convention arc (u, v) points to v
  - $\Rightarrow$ If both $\{(u, v), (v, u)\} \subseteq E$, the arcs are said to be mutual
- Ex: who-calls-whom phone networks, Twitter follower networks

Subgraphs

- Consider a given graph $G(V, E)$

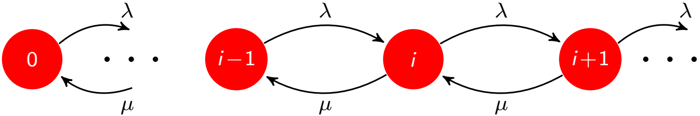


- **Def:** Graph $G'(V', E')$ is an induced subgraph of G if $V' \subseteq V$ and $E' \subseteq E$ is the collection of edges in G among that subset of vertices
- Ex: Graph induced by $V' = \{1, 4, 5\}$



Weighted graphs

- Oftentimes one labels vertices, edges or both with numerical values
  - $\Rightarrow$ Such graphs are called weighted graphs
- Useful in modeling are e.g., Markov chain transition diagrams
- Ex: Single server queuing system (M/M/1 queue)



- Labels could correspond to measurements of network processes
- Ex: Node is infected or not with influenza, IP traffic carried by a link

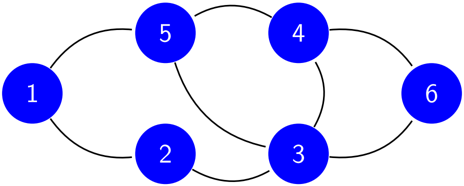
Typical network representations

Network	Graph representation
WWW	Directed multi-graph (with loops), unweighted
Facebook friendships	Undirected, unweighted
Citation network	Directed, unweighted, acyclic
Collaboration network	Undirected, unweighted
Mobile phone calls	Directed, weighted
Protein interaction	Undirected multi-graph (with loops), unweighted
⋮	⋮

- Note that multi-edges are often encoded as edge weights (counts)

Adjacency

- Useful to develop a language to discuss the connectivity of a graph
- A simple and local notion is that of adjacency
  - $\Rightarrow$ Vertices $u, v \in V$ are said adjacent if joined by an edge in E
  - $\Rightarrow$ Edges $e_1, e_2 \in E$ are adjacent if they share an endpoint in V

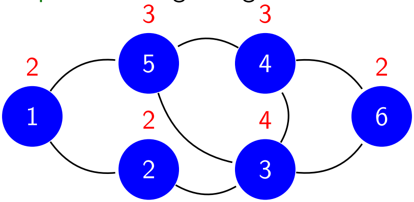


- In figure $\Rightarrow$ Vertices 1 and 5 are adjacent; 2 and 4 are not
  $\Rightarrow$ Edge $(1, 2)$ is adjacent to $(1, 5)$, but not to $(4, 6)$

Degree

- An edge (u, v) is incident with the vertices u and v
- **Def:** The degree d_v of vertex v is its number of incident edges

  $\Rightarrow$ Degree sequence arranges degrees in non-decreasing order



- In figure $\Rightarrow$ Vertex degrees shown in red, e.g., $d_1 = 2$ and $d_5 = 3$

  $\Rightarrow$ Graph's degree sequence is 2,2,2,3,3,4
- High-degree vertices likely influential, central, prominent. More soon

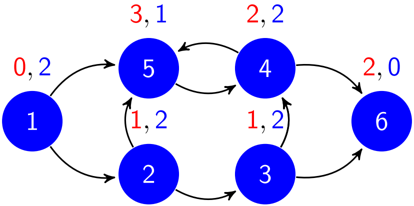
Properties and observations about degrees

- Degree values range from 0 to $N_v - 1$
- The sum of the degree sequence is twice the size of the graph

$$\sum_{v=1}^{N_v} d_v = 2|E| = 2N_e$$

$\Rightarrow$ The number of vertices with odd degree is even

- In digraphs, we have vertex in-degree d_v^{in} and out-degree d_v^{out}



- In figure $\Rightarrow$ Vertex in-degrees shown in red, out-degrees in blue
  $\Rightarrow$ For example, $d_1^{in} = 0, d_1^{out} = 2$ and $d_5^{in} = 3, d_5^{out} = 1$

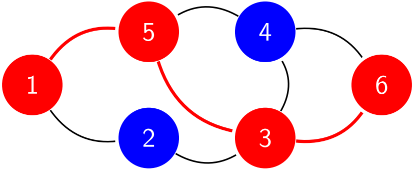
Movement in a graph and connectivity

Movement in a graph

- **Def:** A walk of length l from v_0 to v_l is an alternating sequence

$$\{v_0, e_1, v_1, \ldots, v_{l-1}, e_l, v_l\}, \text{ where } e_i \text{ is incident with } v_{i-1}, v_i$$

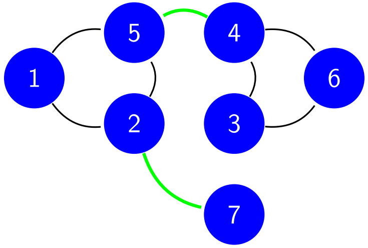
- A trail is a walk without repeated edges
- A path is a walk without repeated nodes (hence, also a trail)



- A walk or trail is closed when $v_0 = v_l$. A closed trail is a circuit
- A cycle is a closed walk with no repeated nodes except $v_0 = v_l$
- All these notions generalize naturally to directed graphs

Connectivity

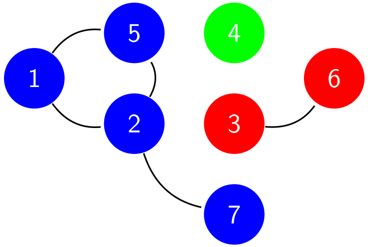
- Vertex v is reachable from u if there exists a $u - v$ walk
- **Def:** Graph is connected if every vertex is reachable from every other



- If bridge edges are removed, the graph becomes disconnected

Connected components

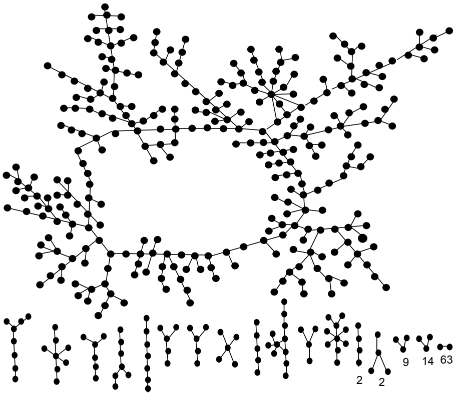
- **Def:** A component is a maximally connected subgraph
  $\Rightarrow$ Maximal means adding a vertex will ruin connectivity



- In figure $\Rightarrow$ Components are $\{1, 2, 5, 7\}$, $\{3, 6\}$ and $\{4\}$
  $\Rightarrow$ Subgraph $\{3, 4, 6\}$ not connected, $\{1, 2, 5\}$ not maximal
- Disconnected graphs have 2 or more components
  $\Rightarrow$ Largest component often called giant component

Giant connected components

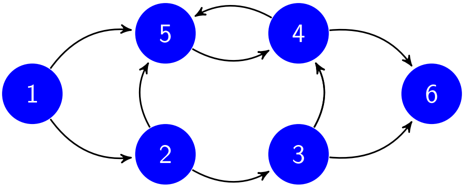
- Large real-world networks typically exhibit one giant component
- Ex: romantic relationships in a US high school [Bearman et al'04]



- Q: Why do we expect to find a single giant component?
- A: Well, it only takes one edge to merge two giant components

Connectivity of directed graphs

- Connectivity is more subtle with directed graphs. Two notions
- **Def:** Digraph is strongly connected if for every pair $u, v \in V$, u is reachable from v (via a directed walk) and vice versa
- **Def:** Digraph is weakly connected if connected after disregarding arc directions, i.e., the underlying undirected graph is connected

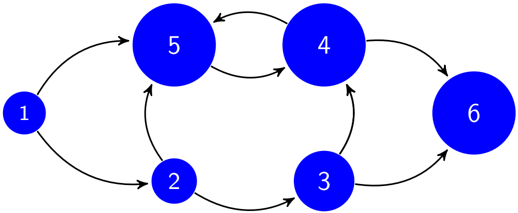


- Above graph is weakly connected but not strongly connected

  $\Rightarrow$ Strong connectivity obviously implies weak connectivity

How well connected nodes are?

- Q: Which node is the most connected?
- A: Node rankings to measure website relevance, social influence
- There are two important connectivity indicators
  - ⇒ How many links point to a node (outgoing links irrelevant)
  - ⇒ How important are the links that point to a node



- Idea exploited by Google's PageRank© to rank webpages

  ... by social scientists to study trust & reputation in social networks

  ... by ISI to rank scientific papers, journals ... More soon

Families of graphs

Basic definitions and concepts

Movement in a graph and connectivity

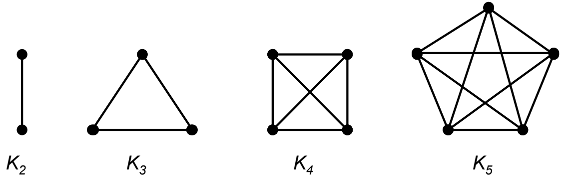
Families of graphs

Algebraic graph theory

Graph data structures and algorithms

Complete graphs and cliques

- A complete graph K_n of order n has all possible edges

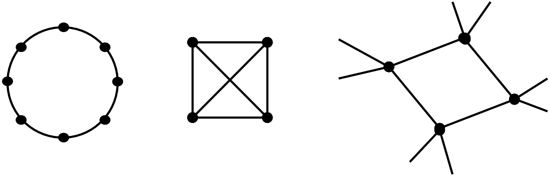


- Q: What is the size of K_n?
- A: Number of edges in K_n = Number of vertex pairs $= \binom{n}{2} = \frac{n(n-1)}{2}$
- Of interest in network analysis are cliques, i.e., complete subgraphs

  $\Rightarrow$ Extreme notions of cohesive subgroups, communities

Regular graphs

- A *d*-regular graph has vertices with equal degree d



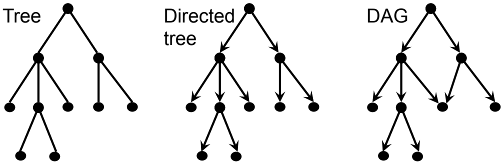
- Naturally, the complete graph K_n is $(n-1)$-regular

  $\Rightarrow$ Cycles are 2-regular (sub) graphs

- Regular graphs arise frequently in e.g.,
  - Physics and chemistry in the study of crystal structures
  - Geo-spatial settings as pixel adjacency models in image processing
  - Opinion formation, information cycles as regular subgraphs

Trees and directed acyclic graphs

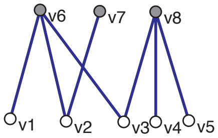
- A tree is a connected acyclic graph. An acyclic graph is a forest
- Ex: river network, information cascades in Twitter, citation network

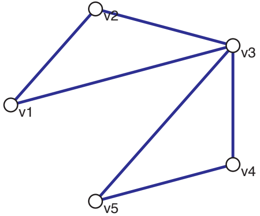


- A directed tree is a digraph whose underlying undirected graph is a tree
  - $\Rightarrow$ Root is only vertex with paths to all other vertices
- Vertex terminology: parent, children, ancestor, descendant, leaf
- Underlying graph of a directed acyclic graph (DAG) may not be a tree
  - $\Rightarrow$ DAGs have a near-tree structure, also useful for algorithms

Bipartite graphs

- A graph $G(V, E)$ is called bipartite when
  - $\Rightarrow$ V can be partitioned in two disjoint sets, say V_1 and V_2; and
  - $\Rightarrow$ Each edge in E has one endpoint in V_1, the other in V_2

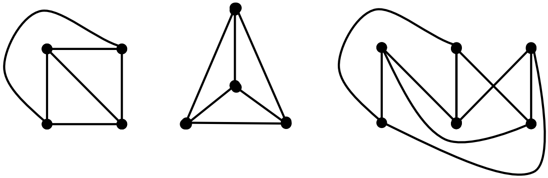




- Useful to represent e.g., membership or affiliation networks
  - $\Rightarrow$ Nodes in V_1 could be people, nodes in V_2 clubs
  - $\Rightarrow$ Induced graph $G(V_1, E_1)$ joins members of same club

Planar graphs

- A graph $G(V, E)$ is called planar if it can be drawn in the plane so that no two of its edges cross each other



- Planar graphs can be drawn in the plane using straight lines only
- Useful to represent or map networks with a spatial component

  $\Rightarrow$ Planar graphs are rare

  $\Rightarrow$ Some mapping tools minimize edge crossings

Algebraic graph theory

Adjacency matrix

- Algebraic graph theory deals with matrix representations of graphs
- Q: How can we capture the connectivity of $G(V, E)$ in a matrix?
- A: Binary, symmetric adjacency matrix $\mathbf{A} \in \{0, 1\}^{N_v \times N_v}$, with entries

$$A_{ij} = \begin{cases} 1, & \text{if } (i, j) \in E \\ 0, & \text{otherwise} \end{cases} .$$

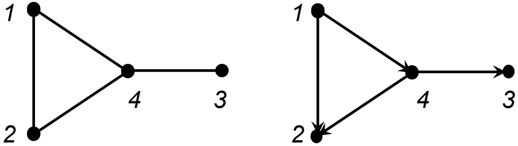
  $\Rightarrow$ Note that vertices are indexed with integers $1, \ldots, N_v$

  $\Rightarrow$ Binary and symmetric $\mathbf{A}$ for unweighted and undirected graph

- In words, $\mathbf{A}$ is one for those entries whose row-column indices denote vertices in V joined by an edge in E, and is zero otherwise

Adjacency matrix examples

- Examples for undirected graphs and digraphs



$$\mathbf{A}_u = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}, \quad \mathbf{A}_d = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

- If the graph is weighted, store the (i, j) weight instead of 1

Adjacency matrix properties

- Adjacency matrix useful to store graph structure. More soon

  $\Rightarrow$ Also, operations on $\mathbf{A}$ yield useful information about G
- Degrees: Row-wise sums give vertex degrees, i.e., $\sum_{j=1}^{N_v} A_{ij} = d_i$
- For digraphs $\mathbf{A}$ is not symmetric and row-, colum-wise sums differ

$$\sum_{j=1}^{N_v} A_{ij} = d_i^{out}, \qquad \sum_{i=1}^{N_v} A_{ij} = d_j^{in}$$

- Walks: Let $\mathbf{A}^r$ denote the r-th power of $\mathbf{A}$, with entries $A_{ij}^{(r)}$

  $\Rightarrow$ Then $A_{ij}^{(r)}$ yields the number of $i - j$ walks of length r in G
- Corollary: $\text{tr}(\mathbf{A}^2)/2 = N_e$ and $\text{tr}(\mathbf{A}^3)/6 = \#\triangle$ in G
- Spectrum: G is d-regular if and only if $\mathbf{1}$ is an eigenvector of $\mathbf{A}$, i.e.,

$$\mathbf{A}\mathbf{1} = d\mathbf{1}$$

Incidence matrix

- A graph can be also represented by its $N_v \times N_e$ incidence matrix $\mathbf{B}$

  $\Rightarrow$ $\mathbf{B}$ is in general not a square matrix, unless $N_v = N_e$

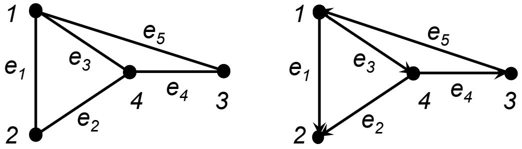
- For undirected graphs, the entries of $\mathbf{B}$ are

$$B_{ij} = \begin{cases} 1, & \text{if vertex } i \text{ incident to edge } j \\ 0, & \text{otherwise} \end{cases}.$$

- For digraphs we also encode the direction of the arc, namely

$$B_{ij} = \begin{cases} 1, & \text{if edge } j \text{ is } (k, i) \\ -1, & \text{if edge } j \text{ is } (i, k) \\ 0, & \text{otherwise} \end{cases}.$$

- Examples for undirected graphs and digraphs



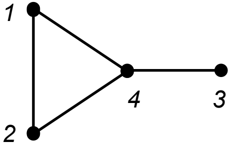
$$\mathbf{B}_u = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}, \quad \mathbf{B}_d = \begin{pmatrix} -1 & 0 & -1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 \\ 0 & -1 & 1 & -1 & 0 \end{pmatrix}$$

- If the graph is weighted, modify nonzero entries accordingly

Graph Laplacian

- Vertex degrees often stored in the diagonal matrix $\mathbf{D}$, where $D_{ii} = d_i$

$$\mathbf{D} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$



- The $N_v \times N_v$ symmetric matrix $\mathbf{L} := \mathbf{D} - \mathbf{A}$ is called graph Laplacian

$$L_{ij} = \begin{cases} d_i, & \text{if } i = j \\ -1, & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases}, \quad \mathbf{L} = \begin{pmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ -1 & -1 & -1 & 3 \end{pmatrix}$$

Laplacian matrix properties

- Smoothness: For any vector $\mathbf{x} \in \mathbb{R}^{N_v}$ of "vertex values", one has

$$\mathbf{x}^\top \mathbf{L}\mathbf{x} = \sum_{(i,j)\in E} (x_i - x_j)^2$$

  which can be minimized to enforce smoothness of functions on G
- Positive semi-definiteness: Follows since $\mathbf{x}^\top \mathbf{L}\mathbf{x} \geq 0$ for all $\mathbf{x} \in \mathbb{R}^{N_v}$
- Rank deficiency: Since $\mathbf{L}\mathbf{1} = \mathbf{0}$, $\mathbf{L}$ is rank deficient
- Spectrum and connectivity: The smallest eigenvalue λ_1 of $\mathbf{L}$ is 0
  - If the second-smallest eigenvalue $\lambda_2 \neq 0$, then G is connected
  - If $\mathbf{L}$ has n zero eigenvalues, G has n connected components

Graph data structures and algorithms

Basic definitions and concepts

Movement in a graph and connectivity

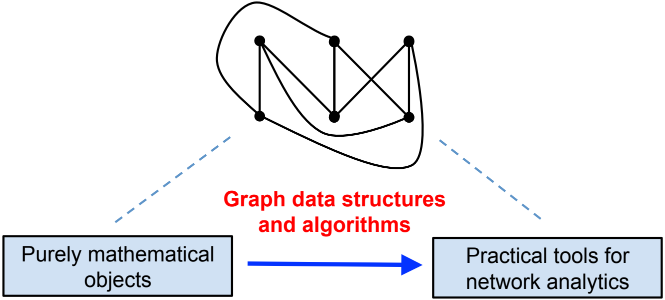
Families of graphs

Algebraic graph theory

Graph data structures and algorithms

Graph data structures and algorithms

- Q: How can we store and analyze a graph G using a computer?



- Data structures: efficient storage and manipulation of a graph
- Algorithms: scalable computational methods for graph analytics

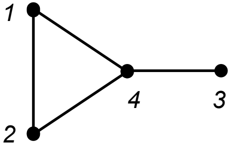
  $\Rightarrow$ Contributions in this area primarily due to computer science

Adjacency matrix as a data structure

- Q: How can we represent and store a graph G in a computer?
- A: The $N_v \times N_v$ adjacency matrix $\mathbf{A}$ is a natural choice

$$A_{ij} = \begin{cases} 1, & \text{if } (i,j) \in E \\ 0, & \text{otherwise} \end{cases} .$$

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$



- Matrices (arrays) are basic data objects in software environments
  - $\Rightarrow$ Naive memory requirement is $O(N_v^2)$
  - $\Rightarrow$ May be undesirable for large, sparse graphs

Networks are sparse graphs

- Most real-world networks are sparse, meaning

$$N_e \ll \frac{N_v(N_v-1)}{2} \text{ or equivalently } \bar{d} := \frac{1}{N_v}\sum_{v=1}^{N_v} d_v \ll N_v - 1$$

- Figures from the study by Leskovec et al '09 are eloquent

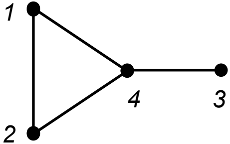
Network dataset	**Order** N_v	**Avg. degree** $\bar{d}$
WWW (Stanford-Berkeley)	319,717	9.65
Social network (LinkedIn)	6,946,668	8.87
Communication (MSN IM)	242,720,596	11.1
Collaboration (DBLP)	317,080	6.62
Roads (California)	1,957,027	2.82
Proteins (S. Cerevisiae)	1,870	2.39

- Graph density $\rho := \frac{N_e}{N_v^2} = \frac{\bar{d}}{2N_v}$ is another useful metric

Adjacency and edge lists

- An adjacency-list representation of graph G is an array of size N_v
  $\Rightarrow$ The i-th array element is a list of the vertices adjacent to i

$$\begin{aligned} L_a[1] &= \{2,4\} \\ L_a[2] &= \{1,4\} \\ L_a[3] &= \{4\} \\ L_a[4] &= \{1,2,3\} \end{aligned}$$



- Similarly, an edge list stores the vertex pairs incident to each edge

$$\begin{aligned} L_e[1] &= \{1,2\} \\ L_e[2] &= \{1,4\} \\ L_e[3] &= \{2,4\} \\ L_e[4] &= \{3,4\} \end{aligned}$$

- In either case, the memory requirement is $O(N_e)$

Graph algorithms and complexity

- Numerous interesting questions may be asked about a given graph
- For few simple ones, lookup in data structures suffices
  - Q1: Are vertices u and v linked by an edge?
  - Q2: What is the degree of vertex u?
- Some others require more work. Still can tackle them efficiently
  - Q1: What is the shortest path between vertices u and v?
  - Q2: How many connected components does the graph have?
  - Q3: Is a given digraph acyclic?
- Unfortunately, in some cases there is likely no efficient algorithm
  - Q1: What is the maximal clique in a given graph?
- Algorithmic complexity key in the analysis of modern network data

Testing for connectivity

- Goal: verify connectivity of a graph based on its adjacency list
- Idea: start from vertex s, explore the graph, mark vertices you visit

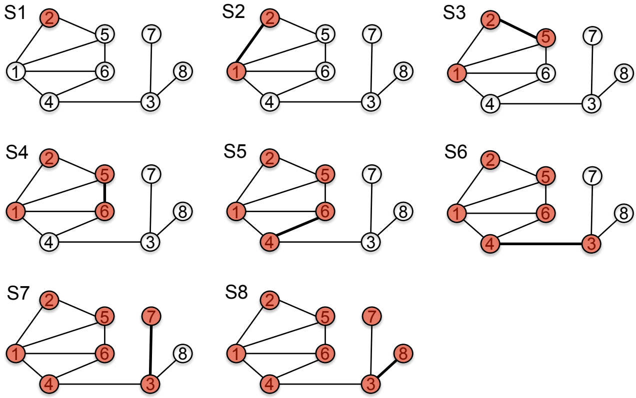
Output : List M of marked vertices in the component
Input : Graph G (e.g., adjacency list)
Input : Starting vertex s

$L := \{s\}$; $M := \{s\}$; % Initialize exploration and marking lists
% Repeat while there are still nodes to explore
while $L \neq \emptyset$ **do**
    choose $u \in L$; % Pick arbitrary vertex to explore
    if $\exists\,(u, v) \in E$ *such that* $v \notin M$ **then**
        choose (u, v) with v of smallest index;
        $L := L \cup \{v\}$; $M := M \cup \{v\}$; % Mark and augment
    else
        $L := L \setminus \{u\}$; % Prune
    end
end

Graph exploration example

- Below we indicate the chosen and marked nodes. Initialize $s = 2$

L	Mark
{2}	2
{2,1}	1
{2,1,5}	5
{2,1,5,6}	6
{1,5,6}	
{1,5,6,4}	4
{5,6,4}	
{5,4}	
{5,4,3}	3
{5,3}	
{5,3,7}	7
{5,3}	
{3}	
{3,8}	8
{3}	
{}	



- Exploration takes $2N_v$ steps. Each node is added and removed once

Breadth-first search

- Choices made arbitrarily in the exploration algorithm. Variants?
- Breadth-first search (BFS): choose for u the first element of L

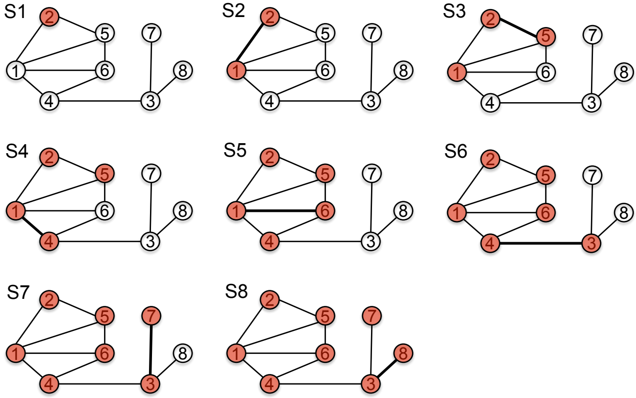
Output : List M of marked vertices in the component
Input : Graph G (e.g., adjacency list)
Input : Starting vertex s

$L := \{s\}$; $M := \{s\}$; % Initialize exploration and marking lists
% Repeat while there are still nodes to explore
while $L \neq \emptyset$ **do**
    $u :=$ first(L); % Breadth first
    if $\exists\,(u, v) \in E$ *such that* $v \notin M$ **then**
        choose (u, v) with v of smallest index;
        $L := L \cup \{v\}$; $M := M \cup \{v\}$; % Mark and augment
    else
        $L := L \setminus \{u\}$; % Prune
    end
end

BFS example

- Below we indicate the chosen and marked nodes. Initialize $s = 2$

L	Mark
$\{2\}$	2
$\{2,1\}$	1
$\{2,1,5\}$	5
$\{1,5\}$	
$\{1,5,4\}$	4
$\{1,5,4,6\}$	6
$\{5,4,6\}$	
$\{4,6\}$	
$\{4,6,3\}$	3
$\{6,3\}$	
$\{3\}$	
$\{3,7\}$	7
$\{3,7,8\}$	8
$\{7,8\}$	
$\{8\}$	
$\{\}$	



- The algorithm builds a wider tree (breadth first)

Depth-first search

- Depth-first search (DFS): choose for u the last element of L

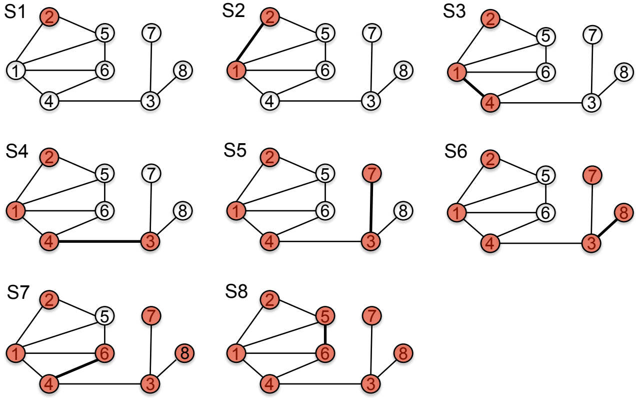
Output : List M of marked vertices in the component
Input : Graph G (e.g., adjacency list)
Input : Starting vertex s

$L := \{s\}$; $M := \{s\}$; % Initialize exploration and marking lists
% Repeat while there are still nodes to explore
while $L \neq \emptyset$ **do**
    $u := \text{last}(L)$; % Depth first
    if $\exists\,(u, v) \in E$ *such that* $v \notin M$ **then**
        choose (u, v) with v of smallest index;
        $L := L \cup \{v\}$; $M := M \cup \{v\}$; % Mark and augment
    else
        $L := L \setminus \{u\}$; % Prune
    end
end

DFS example

- Below we indicate the chosen and marked nodes. Initialize $s = 2$

L	Mark
{2}	2
{2,1}	1
{2,1,4}	4
{2,1,4,3}	3
{2,1,4,3,7}	7
{2,1,4,3}	
{2,1,4,3,8}	8
{2,1,4,3}	
{2,1,4}	
{2,1,4,6}	6
{2,1,4,6,5}	5
{2,1,4,6}	
{2,1,4}	
{2,1}	
{2}	
{}	



- The algorithm builds longer paths (depth first)

Distances in a graph

- Recall a path $\{v_0, e_1, v_1, \ldots, v_{l-1}, e_l, v_l\}$ has length l

  $\Rightarrow$ Edges weights $\{w_e\}$, length of the walk is $w_{e_1} + \ldots + w_{e_l}$

- **Def:** The distance between vertices u and v is the length of the shortest $u - v$ path. Oftentimes referred to as geodesic distance

  $\Rightarrow$ In the absence of a $u - v$ path, the distance is ∞

  $\Rightarrow$ The diameter of a graph is the value of the largest distance

- Q: What are efficient algorithms to compute distances in a graph?
- A: BFS (for unit weights) and Dijkstra's algorithm

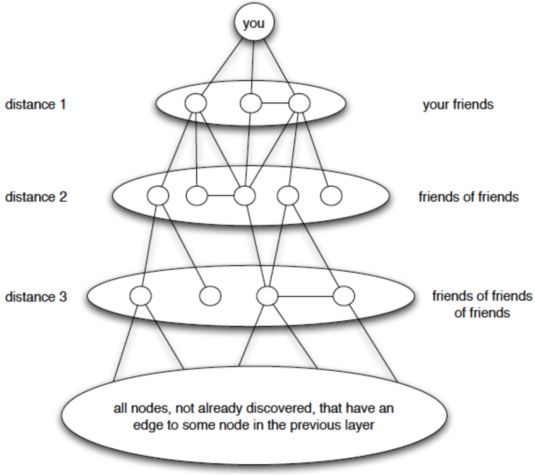
Computing distances with BFS

- Use BFS and keep track of path lengths during the exploration
- Increment distance by 1 every time a vertex is marked

```
Output : Vector d of distances from reference vertex
Input  : Graph G (e.g., adjacency list)
Input  : Reference vertex s
L := {s}; M := {s}; d(s) = 0; % Initialization
% Repeat while there are still nodes to explore
while L ≠ ∅ do
    u := first(L); % Breadth first
    if ∃ (u, v) ∈ E such that v ∉ M then
        choose (u, v) with v of smallest index;
        L := L ∪ {v}; M := M ∪ {v};% Mark and augment
        d(v) := d(u) + 1 % Increment distance
    else
        L := L \ {u}; % Prune
    end
end
```

Example: Distances in a social network

- BFS tree output for your friendship network



Glossary

- (Di) Graph
- Arc
- (Induced) Subgraph
- Incidence
- Degree sequence
- Walk, trail and path
- Connected graph
- Giant connected component
- Strongly connected digraph
- Clique
- Tree
- Bipartite graph
- Directed acyclic graph (DAG)
- Adjacency matrix
- Graph Laplacian
- Adjacency and edge lists
- Sparse graph
- Graph density
- Breadth-first search
- Depth-first search (DFS)
- Geodesic distance (BFS)
- Diameter