Graph Theory Review

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

January 22, 2024
Basic definitions and concepts

Movement in a graph and connectivity

Families of graphs

Algebraic graph theory

Graph data structures and algorithms
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 $E = \{(1, 2), (1, 5), (2, 3), (3, 4), \ldots$
\hspace{1cm} $(3, 5), (3, 6), (4, 5), (4, 6)\}$

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
  - Formal language we use to talk about networks

- **Components**: nodes, vertices \( V \)
- **Inter-connections**: links, edges \( E \)
- **Systems**: networks, graphs \( G(V, E) \)
### Vertices and edges in networks

<table>
<thead>
<tr>
<th>Network</th>
<th>Vertex</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet</td>
<td>Computer/router</td>
<td>Cable or wireless link</td>
</tr>
<tr>
<td>Metabolic network</td>
<td>Metabolite</td>
<td>Metabolic reaction</td>
</tr>
<tr>
<td>WWW</td>
<td>Web page</td>
<td>Hyperlink</td>
</tr>
<tr>
<td>Food web</td>
<td>Species</td>
<td>Predation</td>
</tr>
<tr>
<td>Gene-regulatory network</td>
<td>Gene</td>
<td>Regulation of expression</td>
</tr>
<tr>
<td>Friendship network</td>
<td>Person</td>
<td>Friendship or acquaintance</td>
</tr>
<tr>
<td>Power grid</td>
<td>Substation</td>
<td>Transmission line</td>
</tr>
<tr>
<td>Affiliation network</td>
<td>Person and club</td>
<td>Membership</td>
</tr>
<tr>
<td>Protein interaction</td>
<td>Protein</td>
<td>Physical interaction</td>
</tr>
<tr>
<td>Citation network</td>
<td>Article/patent</td>
<td>Citation</td>
</tr>
<tr>
<td>Neural network</td>
<td>Neuron</td>
<td>Synapse</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In general, graphs may have self-loops and multi-edges.

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 $(u, v)$, $u, v \in V$.
- Means $(u, v)$ distinct from $(v, u)$.
- Directed edges are called arcs.

Directed graphs often called digraphs.
- By convention arc $(u, v)$ points to $v$.
- 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
  - Such graphs are called \textit{weighted graphs}
- Useful in \textit{modeling} are e.g., Markov chain transition diagrams
- \textbf{Ex}: Single server queuing system (M/M/1 queue)

\[ \lambda \quad \mu \]

- Labels could correspond to \textit{measurements} of network processes
- \textbf{Ex}: Node is infected or not with influenza, IP traffic carried by a link
Typical network representations

<table>
<thead>
<tr>
<th>Network</th>
<th>Graph representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>WWW</td>
<td>Directed multi-graph (with loops), unweighted</td>
</tr>
<tr>
<td>Facebook friendships</td>
<td>Undirected, unweighted</td>
</tr>
<tr>
<td>Citation network</td>
<td>Directed, unweighted, acyclic</td>
</tr>
<tr>
<td>Collaboration network</td>
<td>Undirected, unweighted</td>
</tr>
<tr>
<td>Mobile phone calls</td>
<td>Directed, weighted</td>
</tr>
<tr>
<td>Protein interaction</td>
<td>Undirected multi-graph (with loops), unweighted</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**:
  - Vertices $u, v \in V$ are said adjacent if joined by an edge in $E$.
  - Edges $e_1, e_2 \in E$ are adjacent if they share an endpoint in $V$.

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

⇒ **Degree sequence** arranges degrees in non-decreasing order.

In figure ⇒ Vertex degrees shown in red, e.g., \(d_1 = 2\) and \(d_5 = 3\).

⇒ 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

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

⇒ 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 ⇒ Vertex in-degrees shown in red, out-degrees in blue
⇒ 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

Basic definitions and concepts

Movement in a graph and connectivity

Families of graphs

Algebraic graph theory

Graph data structures and algorithms
**Movement in a graph**

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

\[ \{v_0, e_1, v_1, \ldots, v_{l-1}, e_l, v_l\} \], where $e_i$ 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
**Def:** A *component* is a maximally connected subgraph

⇒ Maximal means adding a vertex will ruin connectivity

In figure ⇒ Components are \{1, 2, 5, 7\}, \{3, 6\} and \{4\}
⇒ Subgraph \{3, 4, 6\} not connected, \{1, 2, 5\} not maximal

Disconnected graphs have 2 or more components
⇒ 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 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

⇒ 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
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.

⇒ Extreme notions of cohesive subgroups, communities.
A d-regular graph has vertices with equal degree $d$

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

⇒ 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.

![Tree diagram]

- A directed tree is a digraph whose underlying undirected graph is a tree.
  - **Root** is the 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.
  - **DAGs** have a near-tree structure, also useful for algorithms.
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
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:
- Planar graphs are rare
- Some mapping tools minimize edge crossings
Algebraic graph theory

Basic definitions and concepts

Movement in a graph and connectivity

Families of graphs

Algebraic graph theory

Graph data structures and algorithms
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 $A \in \{0, 1\}^{N_v \times N_v}$, with entries

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

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

⇒ Binary and symmetric $A$ for unweighted and undirected graph

⇒ In words, $A$ 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**

![Graph](image)

\[
A_u = \begin{pmatrix}
0 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 \\
1 & 1 & 1 & 0 \\
\end{pmatrix}

, \quad 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
  - Also, operations on $A$ 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 $A$ is not symmetric and row-, column-wise sums differ
  \[ \sum_{j=1}^{N_v} A_{ij} = d_{i}^{out}, \quad \sum_{i=1}^{N_v} A_{ij} = d_{j}^{in} \]
- Walks: Let $A^r$ denote the $r$-th power of $A$, with entries $A_{ij}^{(r)}$
  - Then $A_{ij}^{(r)}$ yields the number of $i-j$ walks of length $r$ in $G$
- Corollary: $\text{tr}(A^2)/2 = N_e$ and $\text{tr}(A^3)/6 = \#\triangle$ in $G$
- Spectrum: $G$ is $d$-regular if and only if $1$ is an eigenvector of $A$, i.e.,
  \[ A1 = d1 \]
A graph can be also represented by its $N_v \times N_e$ incidence matrix $B$. $B$ is in general not a square matrix, unless $N_v = N_e$.

For undirected graphs, the entries of $B$ 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

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

- 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 $D$, where $D_{ii} = d_i$

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

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

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

$$L = \begin{pmatrix} 2 & -1 & 0 & -1 \\ -1 & 2 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ -1 & -1 & -1 & 3 \end{pmatrix}$$
Laplacian matrix properties

- **Smoothness:** For any vector \( x \in \mathbb{R}^{N_v} \) of “vertex values”, one has
  \[
x^T L x = \sum_{(i,j) \in E} (x_i - x_j)^2
  \]
  which can be minimized to enforce smoothness of functions on \( G \)

- **Positive semi-definiteness:** Follows since \( x^T L x \geq 0 \) for all \( x \in \mathbb{R}^{N_v} \)

- **Rank deficiency:** Since \( L1 = 0 \), \( L \) is rank deficient

- **Spectrum and connectivity:** The smallest eigenvalue \( \lambda_1 \) of \( L \) is 0
  - If the second-smallest eigenvalue \( \lambda_2 \neq 0 \), then \( G \) is connected
  - If \( L \) 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
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
  - 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 $A$ is a natural choice

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

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

Matrices (arrays) are basic data objects in software environments

⇒ Naive memory requirement is $O(N_v^2)$

⇒ May be undesirable for large, sparse graphs
Networks are sparse graphs

- Most real-world networks are sparse, meaning

\[ N_e \ll \frac{N_v(N_v - 1)}{2} \]  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

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

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

$L_a[1] = \{2, 4\}$
$L_a[2] = \{1, 4\}$
$L_a[3] = \{4\}$
$L_a[4] = \{1, 2, 3\}$

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

$L_e[1] = \{1, 2\}$
$L_e[2] = \{1, 4\}$
$L_e[3] = \{2, 4\}$
$L_e[4] = \{3, 4\}$

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

\[
L := \{s\}; \quad M := \{s\}; \quad \% \text{ Initialize exploration and marking lists}
\]

\[
\% \text{ Repeat while there are still nodes to explore}
\]

\[
\text{while } L \neq \emptyset \text{ do}
\]

\[
\quad \text{choose } u \in L; \quad \% \text{ Pick arbitrary vertex to explore}
\]

\[
\quad \text{if } \exists (u, v) \in E \text{ such that } v \notin M \text{ then}
\]

\[
\quad \quad \text{choose } (u, v) \text{ with } v \text{ of smallest index};
\]

\[
\quad \quad L := L \cup \{v\}; \quad M := M \cup \{v\}; \quad \% \text{ Mark and augment}
\]

\[
\quad \text{else}
\]

\[
\quad \quad L := L \setminus \{u\}; \quad \% \text{ Prune}
\]

\[
\text{end}
\]

\[
\text{end}
\]
Graph exploration example

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

<table>
<thead>
<tr>
<th>$L$</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>{2}</td>
<td>2</td>
</tr>
<tr>
<td>{2,1}</td>
<td>1</td>
</tr>
<tr>
<td>{2,1,5}</td>
<td>5</td>
</tr>
<tr>
<td>{2,1,5,6}</td>
<td>6</td>
</tr>
<tr>
<td>{1,5,6}</td>
<td>4</td>
</tr>
<tr>
<td>{1,5,6,4}</td>
<td>4</td>
</tr>
<tr>
<td>{5,6,4}</td>
<td>3</td>
</tr>
<tr>
<td>{5,4}</td>
<td>3</td>
</tr>
<tr>
<td>{5,4,3}</td>
<td>3</td>
</tr>
<tr>
<td>{5,3}</td>
<td>7</td>
</tr>
<tr>
<td>{5,3,7}</td>
<td>7</td>
</tr>
<tr>
<td>{5,3}</td>
<td>8</td>
</tr>
<tr>
<td>{3}</td>
<td>8</td>
</tr>
<tr>
<td>{3,8}</td>
<td>8</td>
</tr>
<tr>
<td>{3}</td>
<td>8</td>
</tr>
<tr>
<td>{}</td>
<td>8</td>
</tr>
</tbody>
</table>

- 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 $s$

$L := \{s\}; \quad M := \{s\}; \quad \% \text{ Initialize exploration and marking lists}$

% Repeat while there are still nodes to explore

while $L \neq \emptyset$ do

\[ u := \text{first}(L); \quad \text{% Breadth first} \]

if \( \exists (u, v) \in E \text{ such that } v \notin M \) then

\[ \text{choose } (u, v) \text{ with } v \text{ of smallest index}; \]

\[ L := L \cup \{v\}; \quad M := M \cup \{v\}; \quad \% \text{ Mark and augment} \]

else

\[ L := L \setminus \{u\}; \quad \% \text{ Prune} \]

end

end
BFS example

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

<table>
<thead>
<tr>
<th>$L$</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>${2}$</td>
<td>2</td>
</tr>
<tr>
<td>${2,1}$</td>
<td>1</td>
</tr>
<tr>
<td>${2,1,5}$</td>
<td>5</td>
</tr>
<tr>
<td>${1,5}$</td>
<td>5</td>
</tr>
<tr>
<td>${1,5,4}$</td>
<td>4</td>
</tr>
<tr>
<td>${1,5,4,6}$</td>
<td>6</td>
</tr>
<tr>
<td>${5,4,6}$</td>
<td>6</td>
</tr>
<tr>
<td>${4,6}$</td>
<td>6</td>
</tr>
<tr>
<td>${4,6,3}$</td>
<td>3</td>
</tr>
<tr>
<td>${6,3}$</td>
<td>3</td>
</tr>
<tr>
<td>${3}$</td>
<td>3</td>
</tr>
<tr>
<td>${3,7}$</td>
<td>7</td>
</tr>
<tr>
<td>${3,7,8}$</td>
<td>8</td>
</tr>
<tr>
<td>${7,8}$</td>
<td>8</td>
</tr>
<tr>
<td>${8}$</td>
<td>8</td>
</tr>
<tr>
<td>{}</td>
<td>8</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$L$</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>${2}$</td>
<td>2</td>
</tr>
<tr>
<td>${2,1}$</td>
<td>1</td>
</tr>
<tr>
<td>${2,1,4}$</td>
<td>4</td>
</tr>
<tr>
<td>${2,1,4,3}$</td>
<td>3</td>
</tr>
<tr>
<td>${2,1,4,3,7}$</td>
<td>7</td>
</tr>
<tr>
<td>${2,1,4,3}$</td>
<td>8</td>
</tr>
<tr>
<td>${2,1,4,3,8}$</td>
<td>8</td>
</tr>
<tr>
<td>${2,1,4,3}$</td>
<td>6</td>
</tr>
<tr>
<td>${2,1,4}$</td>
<td>5</td>
</tr>
<tr>
<td>${2,1}$</td>
<td>2</td>
</tr>
<tr>
<td>${2}$</td>
<td>{}</td>
</tr>
</tbody>
</table>

The algorithm builds longer paths (depth first)
Recall a path \( \{v_0, e_1, v_1, \ldots, v_{l-1}, e_l, v_l\} \) has length \( l \)

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

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

\[
\Rightarrow \text{In the absence of a } u - v \text{ path, the distance is } \infty
\]

\[
\Rightarrow \text{The diameter of a graph is the value of the largest distance}
\]

**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 $L \neq \emptyset$ do
  $u := \text{first}(L); \%$ 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
    $d(v) := d(u) + 1 \%$ Increment distance
  else
    $L := L \setminus \{u\}; \%$ Prune
  end
end
Example: Distances in a social network

- BFS tree output for your friendship network

- all nodes, not already discovered, that have an edge to some node in the previous layer
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