3. **Graphs**

- basic definitions and applications
- graph connectivity and graph traversal
- testing bipartiteness
- connectivity in directed graphs
- DAGs and topological ordering
3. Graphs

- basic definitions and applications
- graph connectivity and graph traversal
- testing bipartiteness
- connectivity in directed graphs
- DAGs and topological ordering

Section 3.1
**Undirected graphs**

**Notation.** $G = (V, E)$

- $V =$ nodes.
- $E =$ edges between pairs of nodes.
- Captures pairwise relationship between objects.
- Graph size parameters: $n = |V|, m = |E|$.

$V = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$

$E = \{ 1-2, 1-3, 2-3, 2-4, 2-5, 3-5, 3-7, 3-8, 4-5, 5-6, 7-8 \}$

$m = 11, n = 8$
## Some graph applications

<table>
<thead>
<tr>
<th>graph</th>
<th>node</th>
<th>edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>communication</td>
<td>telephone, computer</td>
<td>fiber optic cable</td>
</tr>
<tr>
<td>circuit</td>
<td>gate, register, processor</td>
<td>wire</td>
</tr>
<tr>
<td>mechanical</td>
<td>joint</td>
<td>rod, beam, spring</td>
</tr>
<tr>
<td>financial</td>
<td>stock, currency</td>
<td>transactions</td>
</tr>
<tr>
<td>transportation</td>
<td>street intersection, airport</td>
<td>highway, airway route</td>
</tr>
<tr>
<td>internet</td>
<td>class C network</td>
<td>connection</td>
</tr>
<tr>
<td>game</td>
<td>board position</td>
<td>legal move</td>
</tr>
<tr>
<td>social relationship</td>
<td>person, actor</td>
<td>friendship, movie cast</td>
</tr>
<tr>
<td>neural network</td>
<td>neuron</td>
<td>synapse</td>
</tr>
<tr>
<td>protein network</td>
<td>protein</td>
<td>protein-protein interaction</td>
</tr>
<tr>
<td>molecule</td>
<td>atom</td>
<td>bond</td>
</tr>
</tbody>
</table>
Graph representation: adjacency matrix

**Adjacency matrix.** $n$-by-$n$ matrix with $A_{uv} = 1$ if $(u, v)$ is an edge.
- Two representations of each edge.
- Space proportional to $n^2$.
- Checking if $(u, v)$ is an edge takes $\Theta(1)$ time.
- Identifying all edges takes $\Theta(n^2)$ time.

\[
\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
2 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
3 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\
4 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
5 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
7 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
8 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
\end{array}
\]
Graph representation: adjacency lists

**Adjacency lists.** Node indexed array of lists.
- Two representations of each edge.
- Space is $\Theta(m + n)$.
- Checking if $(u, v)$ is an edge takes $O(degree(u))$ time.
- Identifying all edges takes $\Theta(m + n)$ time.

![Graph example](image)
**Paths and connectivity**

**Def.** A path in an undirected graph $G = (V, E)$ is a sequence of nodes $v_1, v_2, \ldots, v_k$ with the property that each consecutive pair $v_{i-1}, v_i$ is joined by an edge in $E$.

**Def.** A path is **simple** if all nodes are distinct.

**Def.** An undirected graph is **connected** if for every pair of nodes $u$ and $v$, there is a path between $u$ and $v$. 
Cycles

**Def.** A cycle is a path $v_1, v_2, \ldots, v_k$ in which $v_1 = v_k$, $k > 2$, and the first $k − 1$ nodes are all distinct.

cycle C = 1–2–4–5–3–1
Trees

**Def.** An undirected graph is a **tree** if it is connected and does not contain a cycle.

**Theorem.** Let $G$ be an undirected graph on $n$ nodes. Any two of the following statements imply the third.

- $G$ is connected.
- $G$ does not contain a cycle.
- $G$ has $n - 1$ edges.
Rooted trees

Given a tree $T$, choose a root node $r$ and orient each edge away from $r$.

**Importance.** Models hierarchical structure.

![](image)

---

**a tree**

**the same tree, rooted at 1**
3. **Graphs**

- basic definitions and applications
- graph connectivity and graph traversal
- testing bipartiteness
- connectivity in directed graphs
- DAGs and topological ordering

_Section 3.2_
Connectivity

**s-t connectivity problem.** Given two nodes $s$ and $t$, is there a path between $s$ and $t$?

**s-t shortest path problem.** Given two nodes $s$ and $t$, what is the length of the shortest path between $s$ and $t$?

**Applications.**

- Friendster.
- Maze traversal.
- Kevin Bacon number.
- Fewest number of hops in a communication network.
Breadth-first search

**BFS intuition.** Explore outward from \( s \) in all possible directions, adding nodes one "layer" at a time.

**BFS algorithm.**
- \( L_0 = \{ s \} \).
- \( L_1 = \) all neighbors of \( L_0 \).
- \( L_2 = \) all nodes that do not belong to \( L_0 \) or \( L_1 \), and that have an edge to a node in \( L_1 \).
- \( L_{i+1} = \) all nodes that do not belong to an earlier layer, and that have an edge to a node in \( L_i \).

**Theorem.** For each \( i \), \( L_i \) consists of all nodes at distance exactly \( i \) from \( s \). There is a path from \( s \) to \( t \) iff \( t \) appears in some layer.
Breadth-first search

**Property.** Let $T$ be a BFS tree of $G = (V, E)$, and let $(x, y)$ be an edge of $G$. Then, the level of $x$ and $y$ differ by at most 1.
Breadth-first search: analysis

Theorem. The above implementation of BFS runs in $O(m + n)$ time if the graph is given by its adjacency representation.

Pf.

• Easy to prove $O(n^2)$ running time:
  - at most $n$ lists $L[i]$
  - each node occurs on at most one list; for loop runs $\leq n$ times
  - when we consider node $u$, there are $\leq n$ incident edges $(u, v)$, and we spend $O(1)$ processing each edge

• Actually runs in $O(m + n)$ time:
  - when we consider node $u$, there are $\text{degree}(u)$ incident edges $(u, v)$
  - total time processing edges is $\sum_{u \in V} \text{degree}(u) = 2m$. □

  each edge $(u, v)$ is counted exactly twice
  in sum: once in $\text{degree}(u)$ and once in $\text{degree}(v)$
Connected component

**Connected component.** Find all nodes reachable from $s$.

Connected component containing node $1 = \{ 1, 2, 3, 4, 5, 6, 7, 8 \}$. 
Connected component

**Connected component.** Find all nodes reachable from $s$.

---

$R$ will consist of nodes to which $s$ has a path
Initially $R = \{s\}$
While there is an edge $(u, v)$ where $u \in R$ and $v \notin R$
   Add $v$ to $R$
Endwhile

---

**Theorem.** Upon termination, $R$ is the connected component containing $s$.

- BFS = explore in order of distance from $s$.
- DFS = explore in a different way.
3. **Graphs**

- basic definitions and applications
- graph connectivity and graph traversal
- testing bipartiteness
- connectivity in directed graphs
- DAGs and topological ordering
Directed graphs

Notation. \( G = (V, E) \).
- Edge \((u, v)\) leaves node \(u\) and enters node \(v\).

Ex. Web graph: hyperlink points from one web page to another.
- Orientation of edges is crucial.
- Modern web search engines exploit hyperlink structure to rank web pages by importance.
Some directed graph applications

<table>
<thead>
<tr>
<th>directed graph</th>
<th>node</th>
<th>directed edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>transportation</td>
<td>street intersection</td>
<td>one-way street</td>
</tr>
<tr>
<td>web</td>
<td>web page</td>
<td>hyperlink</td>
</tr>
<tr>
<td>food web</td>
<td>species</td>
<td>predator-prey relationship</td>
</tr>
<tr>
<td>WordNet</td>
<td>synset</td>
<td>hypernym</td>
</tr>
<tr>
<td>scheduling</td>
<td>task</td>
<td>precedence constraint</td>
</tr>
<tr>
<td>financial</td>
<td>bank</td>
<td>transaction</td>
</tr>
<tr>
<td>cell phone</td>
<td>person</td>
<td>placed call</td>
</tr>
<tr>
<td>infectious disease</td>
<td>person</td>
<td>infection</td>
</tr>
<tr>
<td>game</td>
<td>board position</td>
<td>legal move</td>
</tr>
<tr>
<td>citation</td>
<td>journal article</td>
<td>citation</td>
</tr>
<tr>
<td>object graph</td>
<td>object</td>
<td>pointer</td>
</tr>
<tr>
<td>inheritance hierarchy</td>
<td>class</td>
<td>inherits from</td>
</tr>
<tr>
<td>control flow</td>
<td>code block</td>
<td>jump</td>
</tr>
</tbody>
</table>
World wide web

Web graph.

- Node: web page.
- Edge: hyperlink from one page to another (orientation is crucial).
- Modern search engines exploit hyperlink structure to rank web pages by importance.
Road network

Vertex = intersection; edge = one-way street.
Political blogosphere graph

Vertex = political blog; edge = link.

The Political Blogosphere and the 2004 U.S. Election: Divided They Blog, Adamic and Glance, 2005
Ecological food web

Food web graph.
- Node = species.
- Edge = from prey to predator.

Graph search

Directed reachability. Given a node \( s \), find all nodes reachable from \( s \).

Directed \( s \)-\( t \) shortest path problem. Given two node \( s \) and \( t \), what is the length of the shortest path from \( s \) and \( t \) ?

Graph search. BFS extends naturally to directed graphs.

Web crawler. Start from web page \( s \). Find all web pages linked from \( s \), either directly or indirectly.
Strong connectivity

**Def.** Nodes $u$ and $v$ are **mutually reachable** if there is a both path from $u$ to $v$ and also a path from $v$ to $u$.

**Def.** A graph is **strongly connected** if every pair of nodes is mutually reachable.

**Lemma.** Let $s$ be any node. $G$ is strongly connected iff every node is reachable from $s$, and $s$ is reachable from every node.

**Pf.** ⇒ Follows from definition.

**Pf.** ⇐ Path from $u$ to $v$: concatenate $u \rightarrow s$ path with $s \rightarrow v$ path.

Path from $v$ to $u$: concatenate $v \rightarrow s$ path with $s \rightarrow u$ path. ▪

ok if paths overlap
Strong connectivity: algorithm

**Theorem.** Can determine if $G$ is strongly connected in $O(m + n)$ time.

**Pf.**
- Pick any node $s$.
- Run BFS from $s$ in $G$.
- Run BFS from $s$ in $G^{\text{reverse}}$.
- Return true iff all nodes reached in both BFS executions.
- Correctness follows immediately from previous lemma. □

reverse orientation of every edge in $G$

![Graph examples](image_url)

- **strongly connected**
- **not strongly connected**
**Strong components**

**Def.** A *strong component* is a maximal subset of mutually reachable nodes.

![Graph with strong components highlighted]

**Theorem.** [Tarjan 1972] Can find all strong components in $O(m + n)$ time.
Section 3.6

3. Graphs

- basic definitions and applications
- graph connectivity and graph traversal
- testing bipartiteness
- connectivity in directed graphs
- DAGs and topological ordering
Directed acyclic graphs

Def. A DAG is a directed graph that contains no directed cycles.

Def. A topological order of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge $(v_i, v_j)$ we have $i < j$. 

![a DAG](image1)

![a topological ordering](image2)
Precedence constraints

Precedence constraints. Edge \((v_i, v_j)\) means task \(v_i\) must occur before \(v_j\).

Applications.

- Course prerequisite graph: course \(v_i\) must be taken before \(v_j\).
- Compilation: module \(v_i\) must be compiled before \(v_j\). Pipeline of computing jobs: output of job \(v_i\) needed to determine input of job \(v_j\).
Directed acyclic graphs

**Lemma.** If $G$ has a topological order, then $G$ is a DAG.

**Pf.** [by contradiction]
- Suppose that $G$ has a topological order $v_1, v_2, \ldots, v_n$ and that $G$ also has a directed cycle $C$. Let's see what happens.
- Let $v_i$ be the lowest-indexed node in $C$, and let $v_j$ be the node just before $v_i$; thus $(v_j, v_i)$ is an edge.
- By our choice of $i$, we have $i < j$.
- On the other hand, since $(v_j, v_i)$ is an edge and $v_1, v_2, \ldots, v_n$ is a topological order, we must have $j < i$, a contradiction. $\blacksquare$

![Diagram](image)

The directed cycle $C$

The supposed topological order: $v_1, \ldots, v_n$
Directed acyclic graphs

**Lemma.** If $G$ has a topological order, then $G$ is a DAG.

**Q.** Does every DAG have a topological ordering?

**Q.** If so, how do we compute one?
Directed acyclic graphs

**Lemma.** If \( G \) is a DAG, then \( G \) has a node with no entering edges.

**Pf.** [by contradiction]

- Suppose that \( G \) is a DAG and every node has at least one entering edge. Let's see what happens.
- Pick any node \( v \), and begin following edges backward from \( v \). Since \( v \) has at least one entering edge \((u, v)\) we can walk backward to \( u \).
- Then, since \( u \) has at least one entering edge \((x, u)\), we can walk backward to \( x \).
- Repeat until we visit a node, say \( w \), twice.
- Let \( C \) denote the sequence of nodes encountered between successive visits to \( w \). \( C \) is a cycle. ■
Directed acyclic graphs

**Lemma.** If $G$ is a DAG, then $G$ has a topological ordering.

**Pf.** [by induction on $n$]

- Base case: true if $n = 1$.
- Given DAG on $n > 1$ nodes, find a node $v$ with no entering edges.
- $G \setminus \{v\}$ is a DAG, since deleting $v$ cannot create cycles.
- By inductive hypothesis, $G \setminus \{v\}$ has a topological ordering.
- Place $v$ first in topological ordering; then append nodes of $G \setminus \{v\}$
- in topological order. This is valid since $v$ has no entering edges.

To compute a topological ordering of $G$:

Find a node $v$ with no incoming edges and order it first
Delete $v$ from $G$
Recursively compute a topological ordering of $G \setminus \{v\}$
and append this order after $v$
Theorem. Algorithm finds a topological order in $O(m + n)$ time.

Pf.

- Maintain the following information:
  - $\text{count}(w) =$ remaining number of incoming edges
  - $S =$ set of remaining nodes with no incoming edges
- Initialization: $O(m + n)$ via single scan through graph.
- Update: to delete $v$
  - remove $v$ from $S$
  - decrement $\text{count}(w)$ for all edges from $v$ to $w$;
    and add $w$ to $S$ if $\text{count}(w)$ hits 0
- this is $O(1)$ per edge