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.

```
   1 2 3 4 5 6 7 8
1 0 1 1 0 0 0 0 0
2 0 1 0 1 1 0 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
```
Graph representation: \textit{adjacency lists}

\textbf{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 $O(\text{degree}(u))$ time.
- Identifying all edges takes $\Theta(m + n)$ time.

degree = number of neighbors of $u$

\[
\text{size} = \deg(1) = 2 \cdot m
\]
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.

![Diagram of a graph with labeled nodes 1 to 8 and a cycle 1-2-4-5-3-1 highlighted.]

$\text{cycle } C = 1-2-4-5-3-1$
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.

Claim: Tree on $n$ nodes has $n-1$ edges.
Connectivity

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

s-t shortest path problem. Given two node $s$ 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. \[ \text{size } \Theta(m+n) \] \[ \text{linear time} \]

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 $\sum_{v \in V} \text{degree}(u) = 2m$.  

\[ \text{each edge } (u, v) \text{ is counted exactly twice} \]
\[ \text{in sum: once in degree}(u) \text{ and once in degree}(v) \]
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.
Bipartite graphs

**Def.** An undirected graph $G = (V, E)$ is **bipartite** if the nodes can be colored blue or white such that every edge has one white and one blue end.

**Applications.**
- Stable marriage: men = blue, women = white.
- Scheduling: machines = blue, jobs = white.

![Diagram of a bipartite graph]
Testing bipartiteness

Many graph problems become:

- Easier if the underlying graph is bipartite (matching).
- Tractable if the underlying graph is bipartite (independent set).

Before attempting to design an algorithm, we need to understand structure of bipartite graphs.
An obstruction to bipartiteness

**Lemma.** If a graph $G$ is bipartite, it cannot contain an odd length cycle.

**Pf.** Not possible to 2-color the odd cycle, let alone $G$. 

- [Diagram of bipartite graph]
- [Diagram of non-bipartite graph]
Bipartite graphs

Lemma. Let $G$ be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node $s$. Exactly one of the following holds.

(i) No edge of $G$ joins two nodes of the same layer, and $G$ is bipartite.
(ii) An edge of $G$ joins two nodes of the same layer, and $G$ contains an odd-length cycle (and hence is not bipartite).
Bipartite graphs

**Lemma.** Let $G$ be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node $s$. Exactly one of the following holds.

(i) No edge of $G$ joins two nodes of the same layer, and $G$ is bipartite.

(ii) An edge of $G$ joins two nodes of the same layer, and $G$ contains an odd-length cycle (and hence is not bipartite).

**Pf. (i)**

- Suppose no edge joins two nodes in the same layer.
- By BFS property, each edge joins two nodes in adjacent levels.
- Bipartition: white = nodes on odd levels, blue = nodes on even levels.

![Diagram of layers L0 to Lk]

Case 1: no edges in same layer.

This algo 2-colors the graph unless it has an odd cycle.

Run time $O(m+n)$
Bipartite graphs

Lemma. Let $G$ be a connected graph, and let $L_0, \ldots, L_k$ be the layers produced by BFS starting at node $s$. Exactly one of the following holds.

(i) No edge of $G$ joins two nodes of the same layer, and $G$ is bipartite.

(ii) An edge of $G$ joins two nodes of the same layer, and $G$ contains an odd-length cycle (and hence is not bipartite).

Pf. (ii)

- Suppose $(x, y)$ is an edge with $x, y$ in same level $L_j$.
- Let $z = \text{lca}(x, y) =$ lowest common ancestor.
- Let $L_i$ be level containing $z$.
- Consider cycle that takes edge from $x$ to $y$,
  then path from $y$ to $z$, then path from $z$ to $x$.
- Its length is $1 + (j-i) + (j-i)$, which is odd. •
The only obstruction to bipartiteness

Corollary. A graph $G$ is bipartite iff it contain no odd length cycle.

bipartite (2-colorable)

not bipartite (not 2-colorable)
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.
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.
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>
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 \textbf{mutually reachable} if there is a both path from \( u \) to \( v \) and also a path from \( v \) to \( u \).

Def. A graph is \textbf{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. \( \Rightarrow \) Follows from definition.

Pf. \( \Leftarrow \) 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.  

\( \quad \checkmark \)  

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 $G_{reverse}$.
- Return true iff all nodes reached in both BFS executions.
- Correctness follows immediately from previous lemma.

Run Time $O(m + n)$

![Diagram showing strongly connected and not strongly connected graphs]

Size of data structure $O(m + n)$
Strong components

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

---

**Theorem.** [1]

In $O(m+n)$ time, to find all strong components.

---

**Depth-First Search (DFS)**

**DFS** ($u$)

$u$ explored

for each edge $u-v$

if $v$ not explored

then $\text{DFS} (v)$

end if

end for

mark $u$ explored
DFS tree

Observe:

\[ \text{DFS}(u) \]
- \( v \) explored
- \( w \) explored
- exit \( \text{DFS}(u) \)

descendants of \( u \) in \( T \)

DFS Tree Property:

\[ u \rightarrow v \rightarrow w \rightarrow \]
DFS Tree Property:

Let $T$ be a DFS tree of $G = (V,E)$.

If $(x,y) \in E$ but not an edge of $T$,

Then one of $x, y$ is an ancestor of the other in $T$.

Proof: Say $\text{DFS}(x)$ is called before $\text{DFS}(y)$ was discovered.

www.cs.sfu.ca/~kabanets/307
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 $i < j$. 

![Diagram of a DAG and a topological ordering]
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.

![Diagram of directed cycle and supposed topological order](image)
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.
- $G - \{v\}$ is a DAG, since deleting $v$ cannot create cycles.
- By inductive hypothesis, $G - \{v\}$ has a topological ordering.
- Place $v$ first in topological ordering; then append nodes of $G - \{v\}$ 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 $G - \{v\}$
and append this order after $v$

---

Topological sorting algorithm: running time

**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
and add \( w \) to \( S \) if \( \text{count}(w) \) hits \( 0 \)
this is \( O(1) \) per edge

\[
\frac{O(m+n)}{\text{init}} + \frac{O(m)}{\text{over all iterations}} = T
\]

BFS \& DFS Implementations

- BFS: use queues
- DFS: use stacks

Recursive algo \(\rightarrow\) stacks