Graphs and Trees: Basic Concepts and Properties
(Discrete Math Review)

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong
This lecture formally defines graphs and trees, and proves some of their basic properties.
An undirected simple graph is a pair of \((V, E)\) where:

- \(V\) is a set of elements;
- \(E\) is a set of unordered pairs \(\{u, v\}\) such that \(u\) and \(v\) are distinct elements in \(V\).

Each element in \(V\) is called a node or a vertex. Each pair in \(E\) is called an edge.

An edge \(\{u, v\}\) is said to be incident to vertices \(u\) and \(v\); the two vertices are said to be adjacent to each other.
This is a graph \((V, E)\) where

- \(V = \{a, b, c, d, e\}\)
- \(E = \{\{a, b\}, \{b, c\}, \{a, d\}, \{b, d\}, \{c, e\}\}\).

- The number of edges equals \(|E| = 5|\).
Vertex-Induced Graphs

Let $G = (V, E)$ be an undirected graph. Fix a subset $V' \subseteq V$. The subgraph of $G$ induced by $V'$ is $(V', E')$ where

$$E' = \{ \{u, v\} \in E \mid u \in V' \text{ and } v \in V' \}.$$
Let $G = (V, E)$ be an undirected simple graph. A **path** in $G$ is a sequence of nodes $(v_1, v_2, \ldots, v_k)$ such that

- $v_i$ and $v_{i+1}$ are adjacent, for each $i \in [1, k-1]$.

A **cycle** in $G$ is a path $(v_1, v_2, \ldots, v_k)$ such that $k \geq 4$ and $v_1 = v_k$. 

---

**Paths and Cycles**

---
(a, b, d, a) is a cycle, whereas (a, b, c, e) is a path but not a cycle.
An undirected graph $G = (V, E)$ is **connected** if, for any two distinct vertices $u$ and $v$, $G$ has a path from $u$ to $v$. 
Example

- The graph on the left is connected.
- The graph on the right is not connected.

Connected graphs are those in which there is a path between every pair of vertices, whereas disconnected graphs contain at least one pair of vertices that are not connected to each other. 

Graphs and Trees: Basic Concepts and Properties (Discrete Math Review)
A **tree** is a connected undirected graph with no cycles.

```
not a tree
```

```
a
/
|
/
|
e  d  c
/
|
|
b
```

```
a
/
|
/
|
e  d  c
/
|
|
b
```

```a tree```
A Property

Lemma: A tree with \( n \) nodes has \( n - 1 \) edges.

The proof will be left to you as an exercise.
Rooting a Tree

Given any tree $T$ and an arbitrary node $r$, we can allocate a level to each node as follows:

- $r$ is the root of $T$ — this is level 0 of the tree.
- All the nodes that are 1 edge away from $r$ constitute level 1 of $T$.
- All the nodes that are 2 edges away from $r$ constitute level 2 of $T$.
- And so on.

The number of levels is called the height of $T$. We say that $T$ has been rooted once a root has been designated.
Example

Rooting the tree at $b$

Rooting the tree at $e$

Height 3

Height 4
Consider a tree $T$ that has been rooted.

Let $u$ and $v$ be two nodes in $T$. We say that $u$ is the parent of $v$ if
- the level of $v$ is one more than that of $u$, and
- $u$ and $v$ are adjacent.

Accordingly, we say that $v$ is a child of $u$. 
Example

Node $b$ is the parent of two child nodes: $a, d$.
Node $e$ is the parent of $c$, which is in turn the parent of $b$. 

Yufei Tao

Graphs and Trees: Basic Concepts and Properties (Discrete Math Review)
Concepts on Rooted Trees — Ancestors and Descendants

Consider a rooted tree $T$.

Let $u$ and $v$ be two nodes in $T$. We say that $u$ is an **ancestor** of $v$ if one of the following holds:

- the level of $u$ is at most that of $v$;
- $u$ has a path to $v$.

**Note:** A node is an ancestor of itself.

Accordingly, if $u$ is an ancestor of $v$, then $v$ is a **descendant** of $u$.

In particular, if $u \neq v$, we say that $u$ is a **proper ancestor** of $v$, and likewise, $v$ is a **proper descendant** of $u$. 
Node $b$ is an ancestor of $b$, $a$ and $d$.
Node $c$ is an ancestor of $c$, $b$, $a$, and $d$.
Node $c$ is a proper ancestor of $b$, $a$, $d$. 
Let $u$ be a node in a rooted tree $T$. Let $T_u$ be the subgraph of $T$ induced by the set of descendants of $u$. The subtree of $u$ is the rooted tree obtained by rooting $T_u$ at $u$. 

**Diagram:**

- **Tree:**
  - Level 0: $e$
  - Level 1: $c$, $b$
  - Level 2: $a$, $d$

- **Subtree of $c$:**
  - Level 1: $c$, $b$
  - Level 2: $a$, $d$

- **Subtree of $b$:**
  - Level 1: $b$
  - Level 2: $a$, $d$
In a rooted tree, a node is a **leaf** if it has no children; otherwise, it is an **internal node**.

Internal nodes: $e$, $c$, and $b$. Leaf nodes: $a$ and $d$. 
**Lemma:** Let $T$ be a rooted tree where every internal node has at least 2 child nodes. If $m$ is the number of leaf nodes, then the number of internal nodes is at most $m - 1$.

**Proof:** Consider the tree as the schedule of a tournament described as follows. The competing teams are initially placed at the leaf nodes. Each internal node $v$ represents a match among the teams at the child nodes, such that only the winning team advances to $v$. The team winning the match at the root is the champion.

Each match eliminates at least one team. There are at most $m - 1$ teams to eliminate before the champion is determined. Hence, there can be at most $m - 1$ matches (i.e., nodes).
A **k-ary tree** is a rooted tree where every internal node has at most $k$ child nodes.

A 2-ary tree is called a **binary tree**.
A binary tree is **left-right labeled** if

- Every node $v$ — except the root — has been designated either as a **left** or **right** node of its parent.
- Every internal node has at most one left child, and at most one right child.

Throughout this course, we will discuss only binary trees that have been left-right labeled. Because of this, by a “binary tree”, we always refer to a left-right labeled one.
A (left-right labeled) binary tree implies an ordering among the nodes at the same level.

Let \( u \) and \( v \) be nodes at the same level with parents \( p_u \) and \( p_v \), respectively. We say that \( u \) is on the left of \( v \) if either of the following holds:

- \( p_u = p_v \) and \( u \) is the left child (implying that \( v \) is the right child);
- \( p_u \neq p_v \) and \( p_u \) is on the left of \( p_v \).

Accordingly, we say that \( v \) is on the right of \( u \).
At Level 1, $b$ is on the left of $c$.
At Level 2, the nodes from left to right are $d$, $e$, and $f$.
At Level 3, the nodes from left to right are $g$, $h$, $i$, $j$, and $k$. 
Consider a binary tree with height $h$. Its level $\ell$ ($0 \leq \ell \leq h - 1$) is **full** if it contains $2^\ell$ nodes.

Levels 0 and 1 are full, but levels 2 and 3 are not.
A binary tree of height \( h \) is **complete** if:

- Levels 0, 1, ..., \( h - 2 \) are all full (i.e., the only possible exception is the bottom level).
- At level \( h - 1 \), the leaf nodes are as far left as possible.
Example

Complete binary trees:

Not complete binary trees:
A Property

**Lemma:** A complete binary tree with \( n \geq 2 \) nodes has height \( O(\log n) \).

**Proof:** Let \( h \) be the height of the binary tree. As Levels 0, 1, ..., \( h-2 \) are full, we know that

\[
2^0 + 2^1 + ... + 2^{h-2} \leq n
\]

\[
\Rightarrow 2^{h-1} - 1 \leq n
\]

\[
\Rightarrow h \leq 1 + \log_2(n + 1) = O(\log n).
\]