Trees
Lecture 16
Trees
Trees

- A tree is a special kind of graph.
A tree is a special kind of graph.
Trees

- A tree is a special kind of graph.
- In graph theory: a connected acyclic (simple) graph.
Trees

- A tree is a special kind of graph.
- In graph theory: a connected acyclic (simple) graph.
- If not necessarily connected, it’s a forest.
Trees

- A tree is a special kind of graph.

- In graph theory: a **connected** **acyclic** (simple) graph.
  
  - If not necessarily connected, it’s a forest.

- Trees in computer science: an extra feature.
Trees

- A tree is a special kind of graph.
- In graph theory: a connected acyclic (simple) graph.
  - If not necessarily connected, it’s a forest.
- Trees in computer science: an extra feature.
  - One node is designated as the root of the tree.
Trees

- A tree is a special kind of graph.
- In graph theory: a connected acyclic (simple) graph.
  - If not necessarily connected, it's a forest.
- Trees in computer science: an extra feature.
  - One node is designated as the root of the tree.
  - (And we draw it is an upside down tree.)
Trees

- A tree is a special kind of graph.
- In graph theory: a connected acyclic (simple) graph.
- If not necessarily connected, it’s a forest.
- Trees in computer science: an extra feature.
- One node is designated as the root of the tree.
- (And we draw it is an upside down tree.)
Trees
Trees

Alternate definition:
Trees

- Alternate definition:

- A graph with a root node such that every node $u$ has a **unique** path to the root
Alternate definition:

A graph with a root node such that every node $u$ has a unique path to the root

There is a path: it is a connected graph.
Trees

Alternate definition:

A graph with a root node such that every node $u$ has a **unique** path to the root.

- There is a path: it is a connected graph.
- Unique: there can be two paths between root and $u$ if and only if there is a cycle.
Anatomy of a Tree
Anatomy of a Tree
Anatomy of a Tree

Any edge \( \{u,v\} \) appears in the path to root from exactly one of \( u \), \( v \).
Any edge \{u,v\} appears in the path to root from exactly one of u, v.

If root to v path uses \{u,v\} then we say v is a child of u.
Anatomy of a Tree

- Any edge \( \{u,v\} \) appears in the path to root from exactly one of \( u, v \)
- If root to \( v \) path uses \( \{u,v\} \) then we say \( v \) is a child of \( u \)
Anatomy of a Tree

- Any edge \( \{u,v\} \) appears in the path to root from exactly one of \( u, v \).
- If root to \( v \) path uses \( \{u,v\} \) then we say \( v \) is a child of \( u \).
- And \( u \) is the parent of \( v \).
Anatomy of a Tree

- Any edge \( \{u,v\} \) appears in the path to root from exactly one of \( u, v \)
- If root to \( v \) path uses \( \{u,v\} \) then we say \( v \) is a child of \( u \)
- And \( u \) is the parent of \( v \)
- Every node except the root has a unique parent
Anatomy of a Tree

- Any edge \{u,v\} appears in the path to root from exactly one of \(u, v\)
  - If root to \(v\) path uses \{u,v\} then we say \(v\) is a child of \(u\)
    - And \(u\) is the parent of \(v\)
  - Every node except the root has a unique parent
- Leaf: a node without children
Anatomy of a Tree

- Any edge \(\{u,v\}\) appears in the path to root from exactly one of \(u\), \(v\).
  - If root to \(v\) path uses \(\{u,v\}\) then we say \(v\) is a child of \(u\).
  - And \(u\) is the parent of \(v\).
- Every node except the root has a unique parent.
- Leaf: a node without children.
**Anatomy of a Tree**

- Any edge \( \{u,v\} \) appears in the path to root from exactly one of \( u, v \)
  - If root to \( v \) path uses \( \{u,v\} \) then we say \( v \) is a **child** of \( u \)
  - And \( u \) is the **parent** of \( v \)
- Every node except the root has a **unique parent**
- **Leaf**: a node without children
  - Every tree has at least one leaf
Anatomy of a Tree

- Any edge \{u,v\} appears in the path to root from exactly one of u, v
- If root to v path uses \{u,v\} then we say v is a child of u
- And u is the parent of v
- Every node except the root has a unique parent
- Leaf: a node without children
- Every tree has at least one leaf
- If tree has only the root, it is a leaf as well
Any edge \{u,v\} appears in the path to root from exactly one of u, v. If root to v path uses \{u,v\} then we say v is a child of u. And u is the parent of v.

Every node except the root has a unique parent.

Leaf: a node without children.

Every tree has at least one leaf.

If tree has only the root, it is a leaf as well.

Internal nodes: all nodes other than the leaves.
Anatomy of a Tree

- **root**
- **u**
  - the parent of **v**
  - a child of **u**
- **v**
  - a leaf
Anatomy of a Tree

- Ancestor of u: all the nodes in the path from root to u (including root and u)
Anatomy of a Tree

- **Ancestor of** $u$: all the nodes in the path from root to $u$ (including root and $u$)
- **Descendants of** $u$: all nodes which have $u$ as an ancestor
Anatomy of a Tree

- **Ancestor of u**: all the nodes in the path from root to u (including root and u)
- **Descendants of u**: all nodes which have u as an ancestor
- **Sub-tree rooted at u**: u and all its descendants, and edges among them
Anatomy of a Tree

- **Ancestor of u:** all the nodes in the path from root to u (including root and u)
- **Descendants of u:** all nodes which have u as an ancestor
- **Sub-tree rooted at u:** u and all its descendants, and edges among them
- **Recursive structure (useful for induction):** Deleting the root breaks up the tree into subtrees whose roots are the children of the original root
Anatomy of a Tree

- **Ancestor of** \( u \): all the nodes in the path from root to \( u \) (including root and \( u \))

- **Descendants of** \( u \): all nodes which have \( u \) as an ancestor

- **Sub-tree rooted at** \( u \): \( u \) and all its descendants, and edges among them

- **Recursive structure** (useful for induction): Deleting the root breaks up the tree into subtrees whose roots are the children of the original root
Anatomy of a Tree

- **root**
- **u** (the parent of v)
- **v** (a child of u)
- **a leaf**
Anatomy of a Tree

- Depth of a node: distance (length of the unique path to it) from root

Diagram:
- root
- the parent of v
- a child of u
- a leaf
Anatomy of a Tree

- **Depth of a node**: distance (length of the unique path to it) from root
- **Depth(root)** = 0
Anatomy of a Tree

- **Depth of a node**: distance (length of the unique path to it) from root
  - Depth(root) = 0
- **Levels**: All nodes of the same depth form
Anatomy of a Tree

- **Depth of a node**: distance (length of the unique path to it) from root
  - Depth(root) = 0
- **Levels**: All nodes of the same depth form
  - Level 0: only the root
Anatomy of a Tree

- **Depth of a node**: distance (length of the unique path to it) from root
  - $\text{Depth(root)} = 0$

- **Levels**: All nodes of the same depth form
  - Level 0: only the root
  - Level 1: roots children ...
Anatomy of a Tree

- Depth of a node: distance (length of the unique path to it) from root
  - Depth(root) = 0
- Levels: All nodes of the same depth form
  - Level 0: only the root
  - Level 1: roots children ...
- Edges only between adjacent levels (hence bipartite)
Anatomy of a Tree

- **Depth of a node**: distance (length of the unique path to it) from root
  - Depth(root) = 0
- **Levels**: All nodes of the same depth form
  - Level 0: only the root
  - Level 1: roots children ...
  - Edges only between adjacent levels (hence bipartite)
- Leaves may occur at various levels
Anatomy of a Tree

- **root**
- **u** is a child of **v**
- **v** is a leaf
- **u** is the parent of **v**
Anatomy of a Tree

- **Arity of a tree**: Maximum number of children (= max-degree - 1, or root’s degree)
Anatomy of a Tree

- **Arity of a tree**: Maximum number of children (= max-degree - 1, or root's degree)
- Binary, Ternary, m-ary, ...

Anatomy of a Tree

- **Arity of a tree**: Maximum number of children (= max-degree - 1, or root's degree)
  - Binary, Ternary, m-ary, ...
- **Full m-ary tree**: Every internal node has exactly m children
Anatomy of a Tree

- **Arity of a tree**: Maximum number of children (= max-degree - 1, or root’s degree)
  - Binary, Ternary, m-ary, ...
- **Full m-ary tree**: Every internal node has exactly m children
- **Complete & Full tree**: A full m-ary tree with all leaves at the same level
Anatomy of a Tree

- **Arity of a tree**: Maximum number of children (= max-degree - 1, or root's degree)
  - Binary, Ternary, m-ary, ...
- **Full m-ary tree**: Every internal node has exactly m children
- **Complete & Full tree**: A full m-ary tree with all leaves at the same level
- In a full m-ary tree with p internal nodes:
Anatomy of a Tree

- **Arity of a tree**: Maximum number of children (= max-degree - 1, or root's degree)
  - Binary, Ternary, m-ary, ...
- **Full m-ary tree**: Every internal node has exactly \( m \) children
- **Complete & Full tree**: A full m-ary tree with all leaves at the same level
- In a full m-ary tree with \( p \) internal nodes:
  - \(|V| = m \cdot p + 1\) (why?)
Anatomy of a Tree

- **Arity of a tree**: Maximum number of children (= max-degree - 1, or root’s degree)
  - Binary, Ternary, m-ary, ...
- **Full m-ary tree**: Every internal node has exactly m children
- **Complete & Full tree**: A full m-ary tree with all leaves at the same level
- In a full m-ary tree with p internal nodes:
  - $|V| = m.p + 1$ (why?)
  - $|E| = m.p$
Anatomy of a Tree

Arity of a tree: Maximum number of children (= max-degree - 1, or root’s degree)

- Binary, Ternary, m-ary, ...

Full m-ary tree: Every internal node has exactly m children

Complete & Full tree: A full m-ary tree with all leaves at the same level

In any full m-ary tree with $p$ internal nodes:

- $|V| = m.p + 1$ (why?)
- $|E| = m.p$

In any tree: $|E| = |V| - 1$
Question
Question

Pick the correct statement

A. In a full binary tree the degree of each internal node is 2
B. In a full binary tree a leaf must always have degree 1
C. A full binary tree with p internal nodes has exactly p+1 leaves
D. A full binary tree with p internal nodes has exactly 2p+1 leaves
E. None of the above
Trees in the Wild
Trees in the Wild
Trees in the Wild
Trees in the Wild
Trees in the Wild
Trees in the Wild
Trees in the Wild
Trees in the Wild
Number of Edges
Number of Edges

In any tree, $|E| = |V| - 1$
Number of Edges

- In any tree, $|E| = |V| - 1$
- Proof by induction: induct on the number of nodes, $|V|$
Number of Edges

In any tree, $|E| = |V|-1$

Proof by induction: induct on the number of nodes, $|V|$

Base case: $|V| = 1$. Only one such tree, and it has $|E|=0$. 
Number of Edges

In any tree, $|E| = |V|-1$

Proof by induction: induct on the number of nodes, $|V|$

Base case: $|V| = 1$. Only one such tree, and it has $|E|=0$.

Induction step: for all $k > 1$

Hypothesis: for every tree $(V,E)$ with $|V|=k-1$, $|E|=|V|-1$

To prove: for every tree $(V,E)$ with $|V|=k$, $|E|=|V|-1$
In any tree, \(|E| = |V| - 1\)

Proof by induction: induct on the number of nodes, \(|V|\)

Base case: \(|V| = 1\). Only one such tree, and it has \(|E| = 0\).

Induction step: for all \(k > 1\)

Hypothesis: for every tree \((V,E)\) with \(|V| = k - 1\), \(|E| = |V| - 1\)

To prove: for every tree \((V,E)\) with \(|V| = k\), \(|E| = |V| - 1\)

Suppose \(G = (V,E)\) is a tree with \(|V| = k > 1\). It must have at least one leaf. This leaf is not the root. So its degree = 1 (one parent, no child). Now consider \(G' = (V',E')\) obtained by deleting this leaf and the one edge incident on it. \(|V'| = k - 1\). \(G'\) is still a tree (no other path passed through this leaf). By induction hypothesis, \(|E'| = k - 2\). But \(|E| = |E'| + 1\). So \(|E| = k - 1\).
Number of Edges

In any tree, $|E| = |V| - 1$

Proof by induction: induct on the number of nodes, $|V|$

Base case: $|V| = 1$. Only one such tree, and it has $|E| = 0$.

Induction step: for all $k > 1$

Hypothesis: for every tree $(V, E)$ with $|V| = k - 1$, $|E| = |V| - 1$

To prove: for every tree $(V, E)$ with $|V| = k$, $|E| = |V| - 1$

Suppose $G = (V, E)$ is a tree with $|V| = k > 1$. It must have at least one leaf. This leaf is not the root. So its degree = 1 (one parent, no child). Now consider $G' = (V', E')$ obtained by deleting this leaf and the one edge incident on it. $|V'| = k - 1$. $G'$ is still a tree (no other path passed through this leaf). By induction hypothesis, $|E'| = k - 2$. But $|E| = |E'| + 1$. So $|E| = k - 1$. 
Binary Trees
Binary Trees

- Full binary tree: $|V| = 2 \times \text{#internal nodes} + 1$
Binary Trees

- Full binary tree: \(|V| = 2 \times \text{#internal nodes} + 1\)
- A little over half of the tree lives in the leaves
Binary Trees

- Full binary tree: $|V| = 2 \times \#\text{internal nodes} + 1$
- A little over half of the tree lives in the leaves
- If the tree is complete and full, then the height is only logarithmic in the total number of nodes (or leaves)
Binary Trees

- Full binary tree: \(|V| = 2 \times \#\text{internal nodes} + 1\)
- A little over half of the tree lives in the leaves
- If the tree is complete and full, then the height is only logarithmic in the total number of nodes (or leaves)

\[ |V| = 2^{h+1} - 1 \]
\[ h = \lceil \log_2 |V| \rceil \]
Binary Trees

- Full binary tree: $|V| = 2 \times \#\text{internal nodes} + 1$
- A little over half of the tree lives in the leaves
- If the tree is complete and full, then the height is only logarithmic in the total number of nodes (or leaves)
- Every node has a “short path” to the root

$|V| = 2^{h+1} - 1$

$h = \lceil \log_2 |V| \rceil$
Binary Trees

- Full binary tree: $|V| = 2 \times \#\text{internal nodes} + 1$
- A little over half of the tree lives in the leaves
- If the tree is complete and full, then the height is only logarithmic in the total number of nodes (or leaves)
- Every node has a "short path" to the root
- Using only degree 3

$|V| = 2^{h+1} - 1$
$h = \lfloor \log_2 |V| \rfloor$
Binary Trees

- Full binary tree: $|V| = 2 \times \#\text{internal nodes} + 1$
- A little over half of the tree lives in the leaves
- If the tree is complete and full, then the height is only logarithmic in the total number of nodes (or leaves)
- Every node has a "short path" to the root
- Using only degree 3
- Ideal for storing data (later)

$|V| = 2^{h+1} - 1$
$h = \lceil \log_2 |V| \rceil$
Binary Trees

- Full binary tree: $|V| = 2 \times \#\text{internal nodes} + 1$
- A little over half of the tree lives in the leaves
- If the tree is complete and full, then the height is only logarithmic in the total number of nodes (or leaves)
  - Every node has a “short path” to the root
  - Using only degree 3
  - Ideal for storing data (later)
  - (But there is only one path, so not suitable in applications that need some form of resilience)
Binary Trees

- Full binary tree: $|V| = 2 \times \#\text{internal nodes} + 1$
- A little over half of the tree lives in the leaves
- If the tree is complete and full, then the height is only logarithmic in the total number of nodes (or leaves)
- Every node has a “short path” to the root
- Using only degree 3
- Ideal for storing data (later)

- (But there is only one path, so not suitable in applications that need some form of resilience)

- Important problem: To maintain the “balance” of the tree even if the tree evolves (nodes added/deleted) over time
Binary Trees

- Full binary tree: $|V| = 2 \times \text{#internal nodes} + 1$
- A little over half of the tree lives in the leaves
- If the tree is complete and full, then the height is only logarithmic in the total number of nodes (or leaves)
- Every node has a “short path” to the root
- Using only degree 3
- Ideal for storing data (later)
- (But there is only one path, so not suitable in applications that need some form of resilience)

Important problem: To maintain the “balance” of the tree even if the tree evolves (nodes added/deleted) over time
- i.e., keep height roughly $\log_2 |V|$
Recursion Trees
(déjà vu)

- $T(0) = 1$
- $T(n) = 2T(n-1) + 1$
- $T(n) = 2^{n+1} - 1$ (guess)

Base case: $n=0$.

Inductive step: $k \geq 1$.

- $T(k) = 2T(k-1) + 1 = 2(2^k-1) + 1 = 2^{k+1} - 1$ ✔
Recursion Trees
(déjà vu)

- $T(0) = 1$
- $T(n) = 2T(n-1) + 1$
- $T(n) = 2^{n+1} - 1$ \( \text{(guess)} \)

Base case: $n=0$.

Inductive step: $k \geq 1$.

- $T(k) = 2T(k-1) + 1 = 2(2^k - 1) + 1 = 2^{k+1} - 1$ \( \checkmark \)

$T(h) = \#\text{nodes in a complete & full binary tree of height } h$
Recursion Trees (déjà vu)

- $T(0) = 1$
- $T(n) = 2T(n-1) + 1$
- $T(n) = 2^{n+1} - 1$ (guess)

Base case: $n=0$.

Inductive step: $k \geq 1$.

- $T(k) = 2T(k-1) + 1 = 2(2^k - 1) + 1 = 2^{k+1} - 1$ ✔

$T(h) = \#\text{nodes in a complete & full binary tree of height } h$

$|V| = 2^{h+1} - 1$
Recursion Trees (déjà vu)

- \( T(0) = 1 \)
- \( T(n) = 2T(n-1) + 1 \)
- \( T(n) = 2^{n+1} - 1 \) (guess)
- Base case: \( n=0 \).
- Inductive step: \( k \geq 1 \).
  - \( T(k) = 2T(k-1) + 1 = 2(2^{k-1}) + 1 = 2^{k+1} - 1 \)
  - Induction is on the height of the tree

\( T(h) = \#\text{nodes in a complete & full binary tree of height } h \)

\( |V| = 2^{h+1} - 1 \)
Recursion Trees
(déjà vu)

T(0) = 1
T(n) = 2T(n-1) + 1
T(n) = 2^{n+1} - 1 (guess)

Base case: n=0.
Inductive step: k ≥ 1.

T(k) = 2T(k-1) + 1 = 2(2^k-1) + 1 = 2^{k+1} - 1 ✔

Induction is on the height of the tree
Then in the inductive step, we “delete” the root, not a leaf
Context-Free Grammar
Context-Free Grammar

Example: a (simplistic) syntax for arithmetic expressions
Context-Free Grammar

Example: a (simplistic) syntax for arithmetic expressions

- $\text{Expr} \rightarrow \text{Expr} + \text{Expr}$
- $\text{Expr} \rightarrow \text{Expr} \times \text{Expr}$
- $\text{Expr} \rightarrow \text{Var}$
- $\text{Var} \rightarrow a$
- $\text{Var} \rightarrow b$
- $\text{Var} \rightarrow c$
Example: a (simplistic) syntax for arithmetic expressions

\[\text{Expr} \rightarrow \text{Expr} + \text{Expr}\]
\[\text{Expr} \rightarrow \text{Expr} \times \text{Expr}\]
\[\text{Expr} \rightarrow \text{Var}\]
\[\text{Var} \rightarrow a\]
\[\text{Var} \rightarrow b\]
\[\text{Var} \rightarrow c\]

Start: Expr

Terminals: +, \times, a, b, c
Context-Free Grammar

Example: a (simplistic) syntax for arithmetic expressions

- **Expr → Expr + Expr**
- **Expr → Expr × Expr**
- **Expr → Var**
  - **Var → a**
  - **Var → b**
  - **Var → c**
- **Start: Expr**
- **Terminals: +, ×, a, b, c**
- **e.g. a + b × c**
Example: a (simplistic) syntax for arithmetic expressions

\[
\begin{align*}
\text{Expr} & \rightarrow \text{Expr} + \text{Expr} \\
\text{Expr} & \rightarrow \text{Expr} \times \text{Expr} \\
\text{Expr} & \rightarrow \text{Var} \\
\text{Var} & \rightarrow a \\
\text{Var} & \rightarrow b \\
\text{Var} & \rightarrow c
\end{align*}
\]

Start: \text{Expr}

Terminals: +, \times, a, b, c

e.g. \(a + b \times c\)
Example: a (simplistic) syntax for arithmetic expressions

- \text{Expr} \rightarrow \text{Expr} + \text{Expr}
- \text{Expr} \rightarrow \text{Expr} \times \text{Expr}
- \text{Expr} \rightarrow \text{Var}
- \text{Var} \rightarrow a
- \text{Var} \rightarrow b
- \text{Var} \rightarrow c

Start: \text{Expr}

Terminals: +, \times, a, b, c

e.g. \( a + b \times c \)

(This grammar is “ambiguous” since there is another parse tree for the same string)
Question
Question

Which of the following strings are generated by (i.e., have a valid parse tree under) the grammar $S \rightarrow aSa \mid bSb \mid \epsilon$ (with start symbol $S$, and terminals $a,b$)?

A. $abSab$
B. $aabb$
C. $abba$
D. $abab$
E. None of the above