# Lec 20: Graphs and Trees III

Prof. Adam J. Aviv

GW

CSCI 1311 Discrete Structures I
Spring 2020

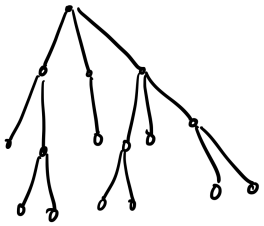


## Rooted Trees



## Rooted Tree

In a rooted tree there is one vertex that is distinguished from the other, the *root*. From the root vertex, all other vertices descended.

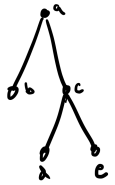


Since a Tree is acyclic, distinguishing one vertex as the root provides a way to distinguish and classify other vertices in the tree. It is also an important structure for organizing data with hierarchical relationships.

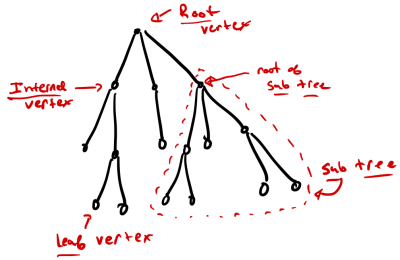


## A tree is like a family

- $a$ is the root of the tree, all vertices descend from the root
- $a$ is the parent of $b$ and $c$
- $b$ and $c$ are siblings
- $c$ is the parent of $d$ and $e$
- $d$ and $e$ are siblings
- $b, c, d, e$ are descendants of $a$
- $d$ and $e$ are descendants of $c$
- $b$, $d$, and $e$ has no descendants and thus are leafs



## Roots, Sub-Roots, Internal and Leaf vertices

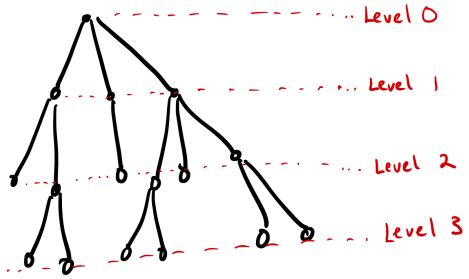



- Root of the tree has no parents
- An internal vertex is one that has a parent and a child
- A leaf vertex has a parent but no children
- An internal vertex can form the sub-root of a sub-tree



## Levels and Height of a tree

The level of the tree describes how many descendants away the given vertex is from

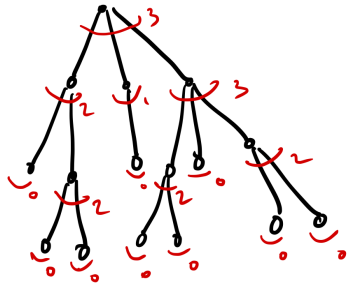



The root is at level 0, and each level counts down from there. The height of a tree is the maximum level of vertices in the tree. The tree above has height 3.



## Branching Factor

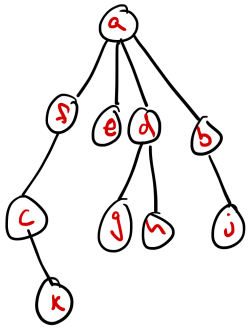
The branching factor is the number of children for each parent. If a vertex has a branching factor of 0, it is a leaf node.



The branching factor of the tree is the maximum number of chidlren for each parent. The branching factor of the tree above is 3.



## Exercise



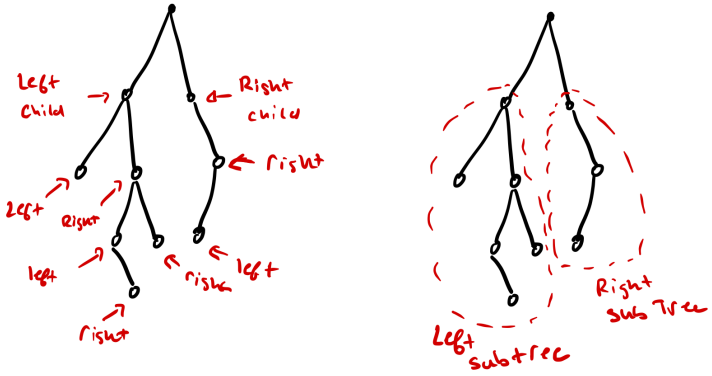

- Name all internal vertices.
- Name all leaf vertices.
- What are the siblings of $f$?
- What are the descendants of $d$?
- What is the level of $j$?
- What is the height of the tree?
- What is the height of the sub-tree where $b$ is the root?
- What is the height of the sub-tree where $e$ is the root?
- What is the branching factor of the tree?

## Binary Trees



## Binary Trees

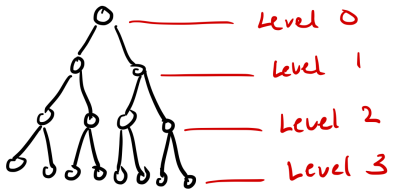
A Binary Tree is a tree where the branching factor is exactly 2. Ever internal vertex can have either a left child (or *left sub-tree*) or a right child (or *right sub-tree*)






## A Full Binary Tree

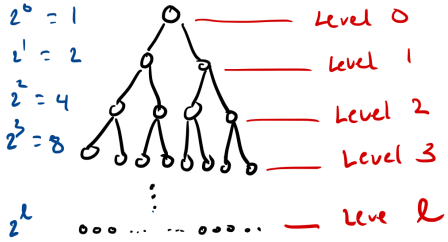
A full binary tree is a binary tree where every level of the tree contains the maximal number of vertices, or 0 vertices. Or, put another way, every parent (internal vertex) has exactly 2 children.






## Number of Vertices in Full Binary Tree

If a tree has height $h$, how many vertices must be in the tree if it is a full binary tree? That is calculate $N(h)$, number of vertices for a full tree of height $h$, where $h \geq 0$.




$$N(h) = \sum_{\ell=0}^{h} 2^{\ell} = \frac{1 - 2^{h+1}}{1 - 2} = 2^{h+1} - 1$$

## Exercise

Prove that if a binary tree is not full and has a $h$, then $N(h) \leq 2^{h+1}$.



## Induction on Binary Trees

We can apply induction to prove a property of binary trees, by either inducting on the number of vertices ($n$) or height ($h$):

- Number of vertices in the tree $n$:
  - IH provides the property is true for all trees with $n$ vertices, you must show it is also true with a tree with $n+1$ vertices.
  - Note that when you remove a vertex (and edge) from a tree with $n+1$ vertices, you have a tree with $n$ vertices and is applicable to the IH.
  - You then need to consider the cases you can go from a tree to $n$ vertices with $n+1$ vertices (or vice versa) and show the property is true.
- Height of the tree $h$:
  - IH provides the property is true for all trees with $h$ height, you must show it is also true with a tree with $h+1$ vertices.
  - Note that when you remove a level from a tree with height $h+1$, you have a tree with height $h$ and is applicable to the IH.
  - If you remove the top level, you have a two sub-trees with height $h$.



## Proof by induction on $n$ (1)

**Theorem**

If a binary tree is full, except at the last level, and has $n \geq 1$ vertices, the height of the tree is $\lfloor \log_2(n) \rfloor$.

*Base Case* $P(1)$: A tree with 1 vertex has a height of 0 and $\log_2(1) = 0$

*Inductive Step* $P(n) \implies P(n+1)$: If a tree with $n$ vertices where each level is full except for the last has a height of $\lfloor \log_2(n) \rfloor$, then a tree with $n+1$ vertices where each level is full except for the last has a height of $\lfloor \log_2(n+1) \rfloor$.

Consider a tree $T$ with $n+1$ vertices. If we remove the last vertex all the way to the right on the last level, we have a tree $T'$ with $n$ vertices where everything except the last level is full. By applying the IH to $T'$, we know that that $T'$ has a height of $\lfloor \log_2(n) \rfloor$.

What are the ways we can go from $T$ to $T'$ by adding a vertex?



## Proof by induction on $n$ (2)

There are two cases: Either $T'$ is a full binary tree, or $T'$ is not a full binary tree.

- $T'$ is a full binary tree with $n$ vertices. It has a height $h = \lfloor \log_2(n) \rfloor$ by IH. Since $T'$ is a full tree, adding a vertex to get $T$ increases the height by 1, so we must show that in this case the height of $T$ is $h+1 = \lfloor \log_2(n+1) \rfloor$

  Consider that in $T'$ there are $n = 2^{h+1} - 1$ vertices as it is a full tree. Adding a vertex, $n+1 = 2^{h+1}$ thus $\log_2(n+1) = h+1$ and $\lfloor log_2(n+1) \rfloor = h+1$, which is what is needed to be shown.
- $T'$ is not a full binary tree with $n$ vertices with a $h = \lfloor \log_2(n) \rfloor$. Since it is not full tree, if we add a vertex to form $T$, there must be a space on the last level of $T'$ for it and the height of $T$ will not increase. We must show that in this case the height of $T$ is $h = \lfloor \log_2(n+1) \rfloor$.

  Since $T'$ is not a full binary tree $2^h - 1 < n < 2^{h+1} - 1$ because it is between a height $h-1$ and $h$. When we add a vertex $2^h < n+1 < 2^{h+1}$, or $h < \log_2(n+1) < h+1$ when taking the log base 2. Thus $\lfloor log_2(n+1) \rfloor = h$ as the floor function rounds down to $h$.

QED

# Proof by induction on $h$ (1)

**Theorem**

If a binary tree $T$ with height $h$ and $\ell$ leaf vertices, then $\ell \leq 2^h$ (or equivalently $\log_2 \ell \leq h$).

*Base Case* $P(0)$: A tree with height has a single vertex that is a leaf vertex (and the root). So $\ell = 1$ and $\log_2(1) = 0 \leq 0$

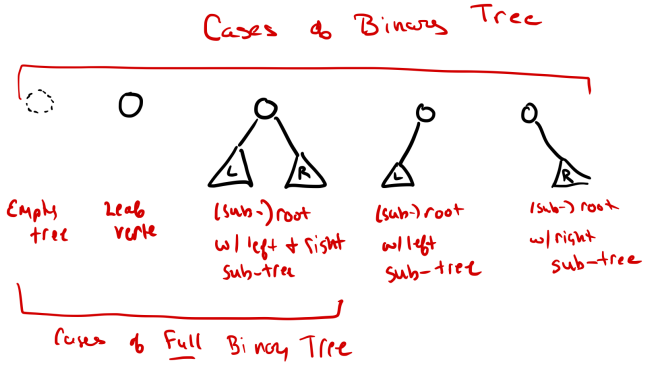
*(strong) Inductive Step* $(\forall k \leq h, P(k)) \implies P(h+1)$: If a binary tree $T$ with height $k \leq h$ has $\ell \leq 2^k$ leaf vertices, then a binary tree $T'$ with height $h+1$ has $\ell' \leq 2^{h+1}$ leaf vertices.

If we remove the root vertex from $T$ with height $h+1$, we are left with (potentially) two sub trees $T_r$ and $T_l$ each with a height $h_l \leq h$ and $h_r \leq h$. We need to consider the cases of sub trees $T_r$ and $T_l$ and how they would be combined to prove something about $T$.



# Cases of Binary Trees

Every binary tree can be described in 5 cases (or 3 if it is a full tree)




Since the height is greater than 0 (proven in the base case), we have three cases. Removing the root vertex gives us a left sub tree, a right sub tree, or both.



# Proof by induction on $h$ (2)

We consider three cases when remove the root vertex.

- Case left sub-tree: We have $T_l$ (and an empty $T_r$). The height of $T_l$ is $h$, and by the IH, the number of leaf nodes $\ell_l \leq 2^h$. If we add back in the root to get $T'$, we have not added any more leaf nodes so it is the case that $\ell$ is unchanged. Then by transitive relationship $\ell' \leq 2^h \leq 2^{h+1}$, and $\ell' \leq 2^{h+1}$ which is what was to be shown.
- Case right sub-tree: This is the same as the case above where $T_l$ and $T_r$ are swapped. This case is covered by the one above.



# Proof by induction on $h$ (3)

- Case right and left sub-tree: We have a $T_l$ and $T_r$ but the height of each $h_l$ and $h_l$, where the height $h+1$ of tree $T'$ is $h+1 = \max(h_l, h_r) + 1$ because if we merged the two sub-trees with a new root, the height would increase by one more than the max height of the sub-trees. That means $h_l \leq h$ and $h_r \leq h$ and thus $T_l$ and $T_r$ are subject to the IH.

  The number of leaf vertices in the left sub-tree $\ell_l \leq 2^{h_l} \leq 2^h$ and right sub-tree $\ell_r \leq 2^{h_r} \leq 2^h$. If we merged the two trees into $T'$, then the number of leafs does not change, so $\ell' = \ell_l + \ell_r$.

  As we are trying to show that $\ell' \leq 2^{h+1}$ we can consider the maximum value of $\ell_l \leq 2^h$ and $\ell_r = 2^h$. So $\ell' = \ell_l + \ell_r = 2^{h+1}$ when $\ell_l$ and $\ell_r$ are maximal, and so all other values off $\ell_l$ and $\ell_r$ would result in lesser values in their sums. Thus $\ell' \leq 2^{h+1}$. Which is what we need to show.

  QED.

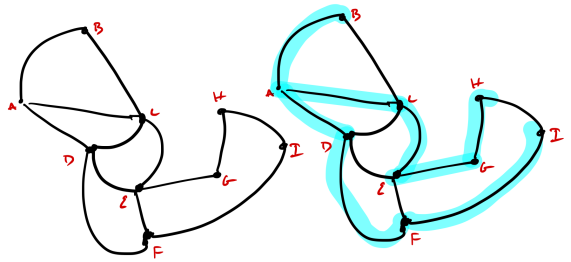
## Spanning Trees and Traversals



## Spanning Tree

**Definition**

A spanning tree of a graph $G$ is a subgraph of $G$ that contains ever vertex of $G$ and is a tree.






## Every connected graph has a spanning tree

Existence of Spanning Tree Proof (by Algorithm!).

Consider a graph $G$ that is connected.

- If it is acyclic, then it is a spanning tree.
- If it has cycles/circuits, then by the Lemma (from before) we can remove one edge from the circuit (breaking the circuit) to produce a graph $G'$ that is still connected.
  - If $G'$ is acyclic, then it is spanning tree.
  - If $G'$ has cycles, repeatedly remove an edge from each circuit until $G''$ is reached that is acyclic (which must occur). $G''$ is a spanning tree

In all cases, a spanning tree can be found.



## Traversal algorithms for finding a spanning tree

As a spanning tree contains all the vertexes, we sometimes define the routine for identifying the spanning a tree as a *traversal* of the vertices. The order in which the vertices are enumerated defines the traversal (and the spanning tree),

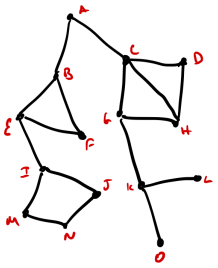
There are two important traversal algorithms that appear frequently in computer science, each starting from a root/start vertex.

- Depth-First-Search (DFS) Traversal
  - Explore entire sub-tree before exploring the next sub-tree
- Breadth-First-Search (BFS) Traversal
  - Explore each level of the tree completely before exploring the next level.

# Depth-First Search (1)

*From a start/root vertex, explore the entire sub-tree of the spanning tree first before exploring the next sub-tree*

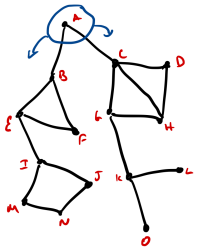
Example: DFS on the following graph starting with $A$. We will break ties by choosing edges that connect to vertices in alphabetic order.



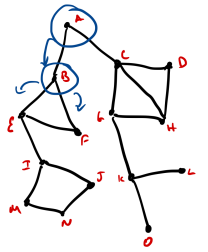



# Depth-First Search (2)

Starting at $A$, we could explore the sub-tree with roots $B$ or $C$: choose $B$ because of alphabetic ordering.



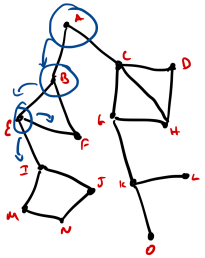

At $B$, we could explore the sub-tree with roots $E$ or $F$: choose $E$ because of alphabetic ordering.



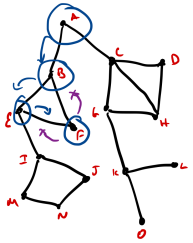



# Depth-First Search (3)

At $E$, the choice is between $F$ and $I$: choose $F$.




At $F$, we are stuck. Both $B$ and $E$ have been enumerated/visited.

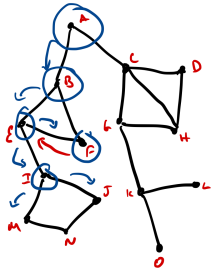



This sub-tree has been full explored, so we back up to $E$.

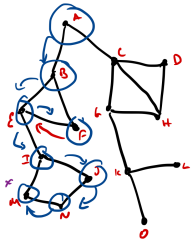


# Depth-First Search (4)

At $I$, we choose $J$ over $M$.



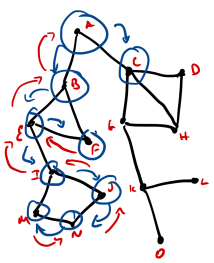

Continuing from $J$, we choose $N$. And then are stuck again.



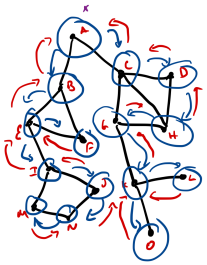

When we back up this time, it takes us all the way to $A$.

## Depth-First Search (5)

From $A$, we explore the sub-tree with sub-root $C$.



We explore the entire sub-tree, and eventually getting stuck at 0.

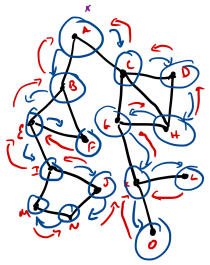


When we back up this time, it takes us all the way to $A$. But, there is no where else to go. We are done!

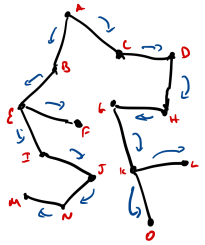


## Depth-First Search (6)

We've embedded the traversal in the graph, with arrows.



Following the arrows, we get the DFS spanning tree



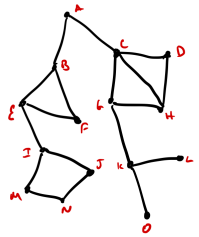
The order of the traversal is the order in which we visited the vertexes in forming the tree.

$$A, B, E, F, I, J, N, M, C, D, H, G, K, L, O$$



## Exercise

Find the DFS spanning trees starting with $E$ and $H$.



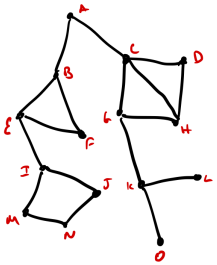
What is the order of each traversal.



## Breadth-First Search (1)

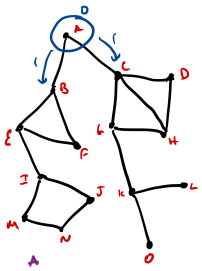
*From a start/root vertex, explore the next level of the spanning tree before exploring the following level.*

Example: BFS on the following graph starting with $A$. We will break ties by choosing edges that connect to vertices in alphabetic order.

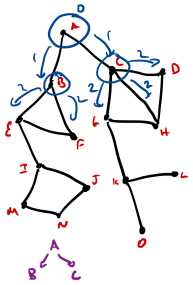


# Breadth-First Search (2)

Starting with $A$ (at level 0), the next level of the tree would include $B$ and $C$.



At $B$ and $C$, the next level (level 2), would include $E$, $F$, $G$, $D$, and $H$

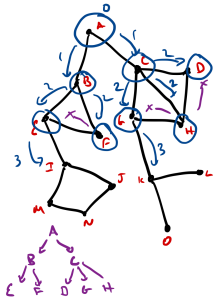


Ties are broken by Alphabetic order. We first visit $B$ and then $C$, left to right across level 1 in the spanning tree.

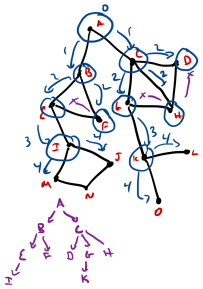


# Breadth-First Search (3)

Exploration $F$, $D$ and $H$ are stuck as all children have been visited.



We continue to explore the next level from the children of $E$ and $G$.

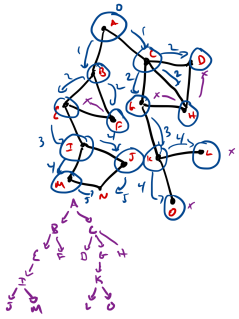


Ties are still broken by alphabetic ordering across the level of the spanning tree.

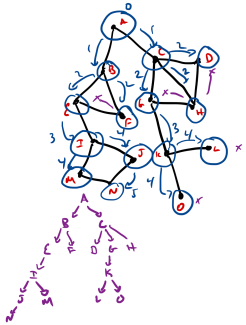


# Breadth-First Search (4)

Continuing on the next level, $L$ and $O$ are stuck



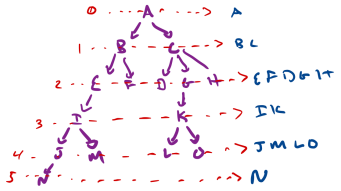
Since $N$ can be reached by both $M$ and $J$, we break the tie alphabetically, and $N$ is a child of $J$.





# Breadth-First Search (5)

We've tracked the spanning tree during the traversal.

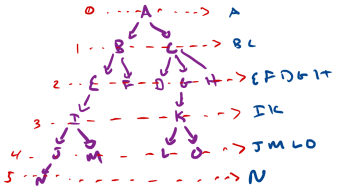


The traversal order is now a *level-order traversal* of the spanning tree, where each level is enumerated left-to-right, top-to-bottom.

$$A, B, C, E, F, D, G, H, I, I, K, J, M, L, O, N$$

## Shortest Path and BFS

The spanning tree for BFS is also a minimum spanning tree, which defines the smallest distance (in terms of number of edges in the path) between the root and other vertices.



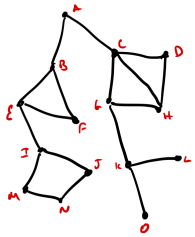
For example, the distance between $A$ and $J$ is 4 "hops" as $j$ is on level 4 of the tree.

Why does BFS define the minimum spanning tree?



## Exercise

Find the BFS spanning trees starting with $E$ and $H$.

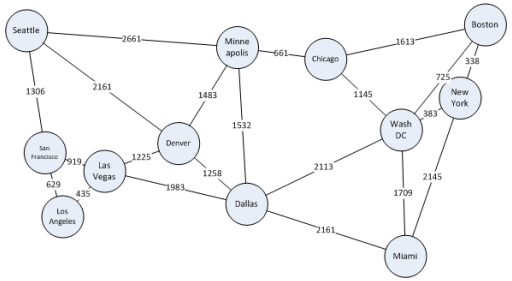


What is the order of each traversal.



## Minimum Spanning Trees with Weighted Edges

If a graph have weighted edges, a minimum spanning tree (MST) is the spanning tree with minimum total weight.




For example, a weighted graph with distances between cities in the USA, the (MST) from Washington would define the shortest path via other connecting cities.

`http://hansolav.net/sql/graphs.html`



## Solving shortest path problems

Solutions for the shortest path in a graph (or network) is extremely important to computer science. There are number of seminal algorithms.

- Dijkstra's Algorithm
- Prim's Algorithm
- Kruskal's Algorithm

The book discusses each of these in detail, but they will likely be covered in your Algorithms or Computer Network classes.