Today: Trees

Basic Definitions

- **Tree**: A connected, undirected graph with no simple circuits (no path from a vertex to itself w/o repeated edges).
- **Forest**: A graph with multiple, disconnected trees.

Tree, Forest, or neither? (§§ 11.1)
• If a graph is a tree, then there is a unique simple path between any two distinct vertices in the graph.
• If not, then there are vertices \( x \) and \( y \) with two distinct simple paths from \( x \) to \( y \) — this would form a simple circuit, which trees aren't supposed to have.

\[ \text{Not in trees!} \]

---

• **Rooted tree**: a tree where one node is designated to be the "root" and nodes can be characterized by their positions relative to the root. (§§ 11.1.1)
• If \( x \) is adjacent to \( y \), the one closer to the root is the \textit{parent} and the other is the \textit{child}
• Siblings are non-adjacent nodes with the same parent
• There is a unique path from any node to the root.
• \textit{Ancestors of} \( x \): The nodes along the path from \( x \) to the root. \textit{Descendants of} \( x \): the nodes that have \( x \) s an ancestor.
Examples of rooted trees

- **Internal nodes**: Nodes with children
- **Leaf nodes**: Nodes without children
- **Depth of node x**: # edges on path from x to root.
- **Height of node x**: max # edges from x to all descendant leaves

CS Dept., IIT

© J. Sasaki 2021
- **n-ary tree**: A tree where no node has > \( n \) children.
- **Binary tree**: 2-ary tree
- **Full n-ary tree**: Every internal node has exactly \( n \) children.

---

**n-ary Trees**

- 1-ary
  - Full

- Binary
  - Not full

- 3-ary
  - Not full
  - Not to 1 child
  - Not to 0 or 3

---

**n-ary Trees**

- Full 0-ary

- Completely full 2-ary

- Full 3-ary
  - Not completely full

- Also 3-ary but not full 3-ary
**How many edges in a tree?**

*Conjecture: A tree with $n$ nodes has $n-1$ edges*

- **Proof by induction**
- **Base case:** $n = 1$ Tree has zero edges. $\checkmark$
- **Ind. case:** A tree with $n+1$ nodes has $n$ edges?
  - Remove one of the leaves and the edge to it
  - Resulting subtree has $n$ nodes, hence $n-1$ edges
  - *(Ind. hyp: A tree with $m \leq n$ nodes has $m-1$ edges)*
  - Add back the leaf and edge: original tree, $n$ edges $\checkmark$

Plain ind. ok *(strong not necessary)*

---

**Spanning Tree of a Graph (§§ 11.4)**

- *Spanning tree of a graph:* A tree that includes all the vertices of a (connected) graph.
Weighted, Minimum Spanning Trees (§§ 11.5)

- If a graph has edge weights, then so do spanning trees
- Minimal spanning tree (MST): The (or a) spanning tree with lowest weight.
- Great for minimizing cost of laying out a network, bus route, etc.

Prim’s MST Algorithm

- tree ← edge with smallest weight + incident vertices
- for k ← 2, ..., |V| - 1
  - Of the edges that have one vertex in the tree and one that isn’t, take the one (a one) with lowest weight.
- tree ← tree plus new edge

Notes:
- The MST is unique if the weights are unique
- Helps if you first sort the edges by \( \leq \) weight \( \Theta(m \log m) \)

\[
\begin{align*}
|V| &= n \\
|E| &= m \\
\end{align*}
\]
**Prim's MST Algorithm: Example**

- **Minimum cost edge of graph**
- **3 has lowest cost**
- **2 edges would add circuit**
- **Choice of edges of 4 and 5**

**Kruskal’s MST Algorithm**

- Similar to Prim’s but instead of maintaining one tree and extending it, Kruskal maintains a forest.
- Kruskal adds an edge with minimum weight that doesn’t have both incident vertices in the same tree.
- New edge might:
  - Start a new tree (with one edge)
  - Extend an existing tree
  - Join two trees together

**4th case**

- Examples of edges causing circuits
- Don’t add red edge

© J. Sasaki 2021
**Example of Kruskal's MST Algorithm**

MST unique because edge weights are unique.

- Add new tree
- Join trees

Prim's algorithm added edges 1, 4, 3, 5, 2, 6.

Same MST.

---

**Universal Address System (§§ 11.3.2)**

- One way to give unique names to the nodes of a tree.
- Addresses have the form `nbr.nbr.nbr` etc.
- Root has address 0. If a node has address \( \alpha \), then its children have addresses \( \alpha.1, \alpha.2, \alpha.3, \ldots \) except if \( \alpha = 0 \).
- (If a node has address \( \alpha \), all its descendants have addresses than begin with \( \alpha \).)
Searching for Spanning Trees in Unweighted Graphs

- To find a spanning tree for an unweighted graph, traverse the graph (without revisiting vertices).
- Two popular techniques: Depth-first search (DFS) and Breadth-First Search (BFS) [§§ 11.4.2, 11.4.3]
- For both, you make some arbitrary node the root.
- In DFS, if you're at node α, you add the edge to α.1 and then recursively visit α.1; on return, repeat for α.2, etc.
- In BFS, you maintain a queue of nodes to visit
  - When you visit α, you add its children α.1, α.2, etc to the queue (and edges from α to α.1, α.2, etc. to the tree). Then you get the next node from the queue, etc.