Trees

-and-

Structural Induction

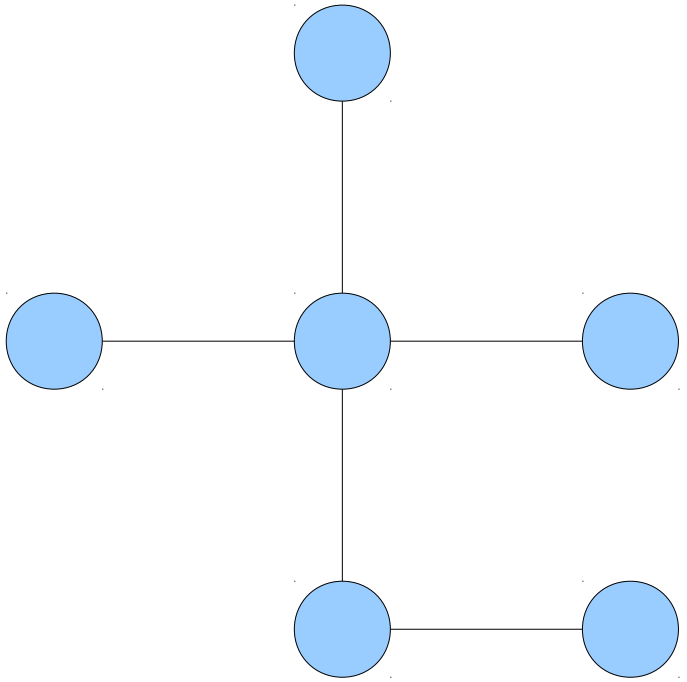
Announcements

- Problem Set 3 due right now.
- Problem Set 4 out, due **Thursday, October 27** at **7:00PM** (right before the midterm).
  - Please ask questions if you have them!
  - OH schedule will be updated; we'll send out details later on.
- Some problem sets are missing names! You are not getting credit for your work!
  - **1x PSet 1**
  - **2x PSet 2**
- Friday four square today, 4:15 PM in front of Gates!

Trees!

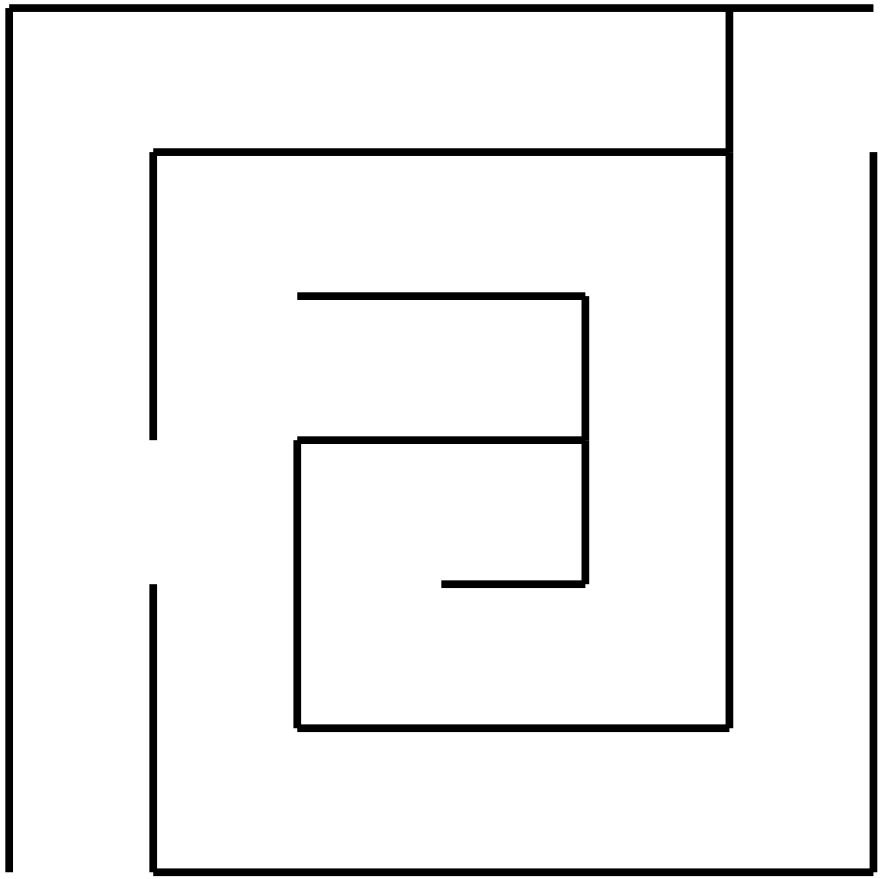
A **tree** is an undirected, connected graph with no cycles.

Trees!

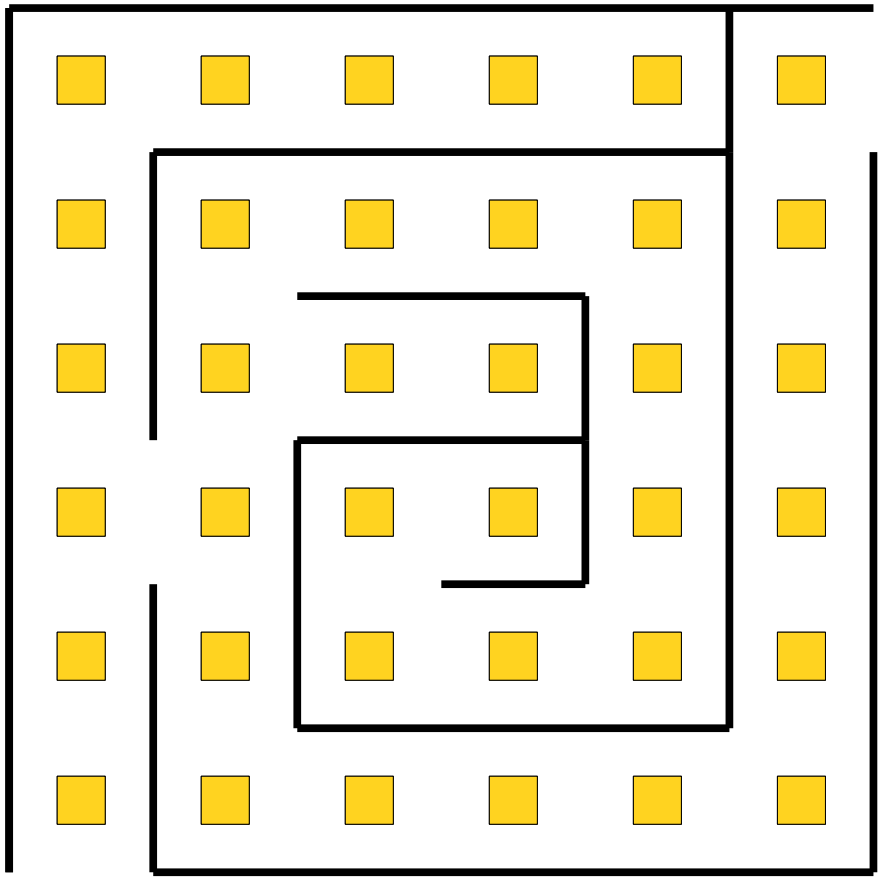


A **tree** is an undirected, connected graph with no cycles.

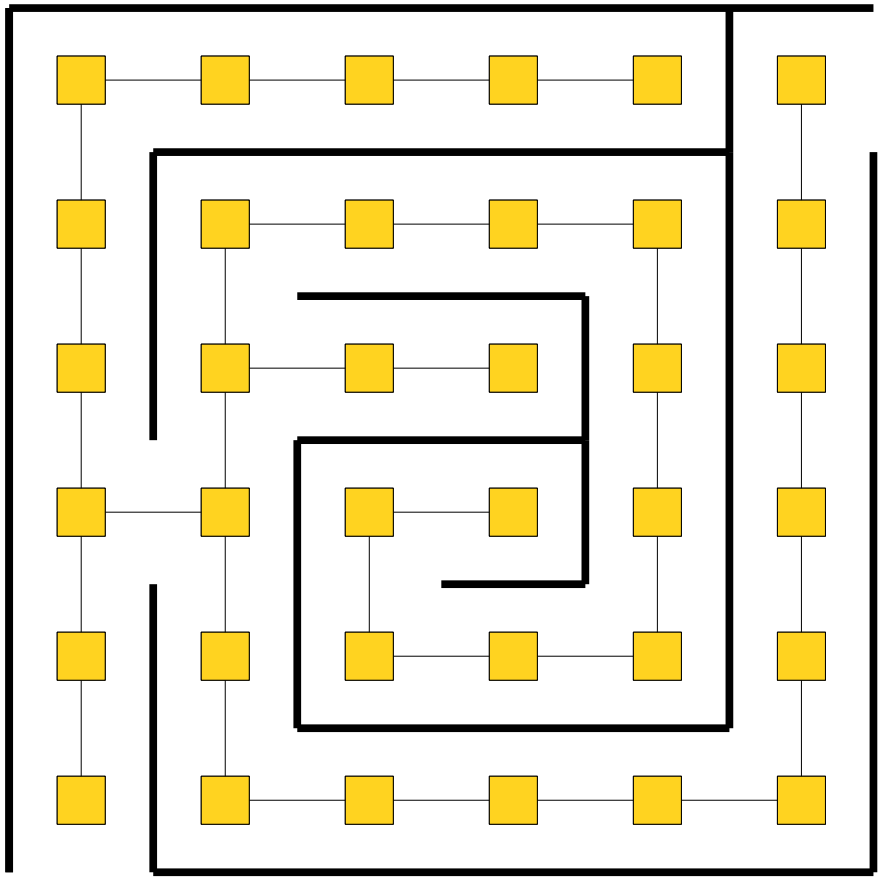
Trees!



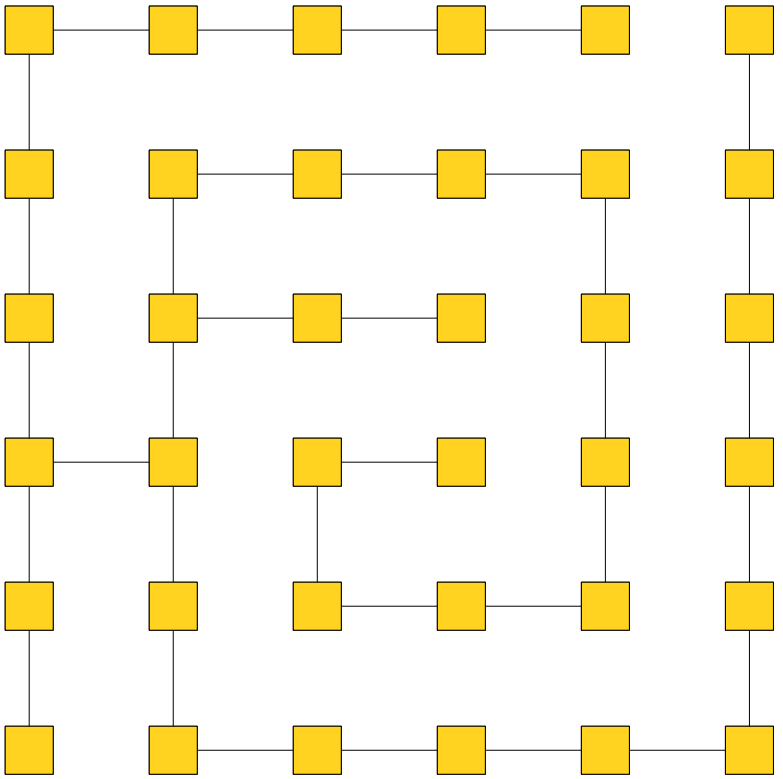
Trees!



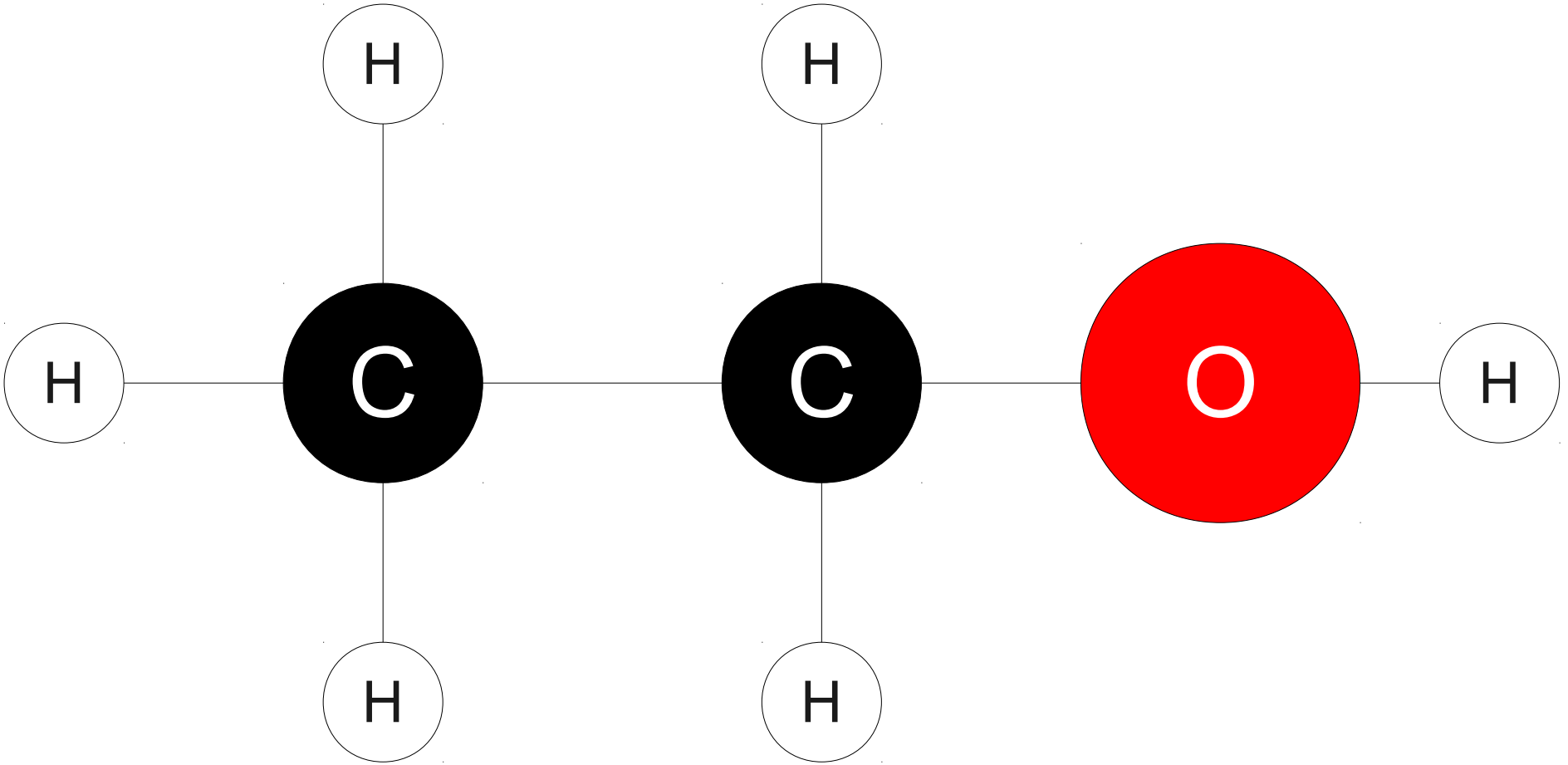
Trees!



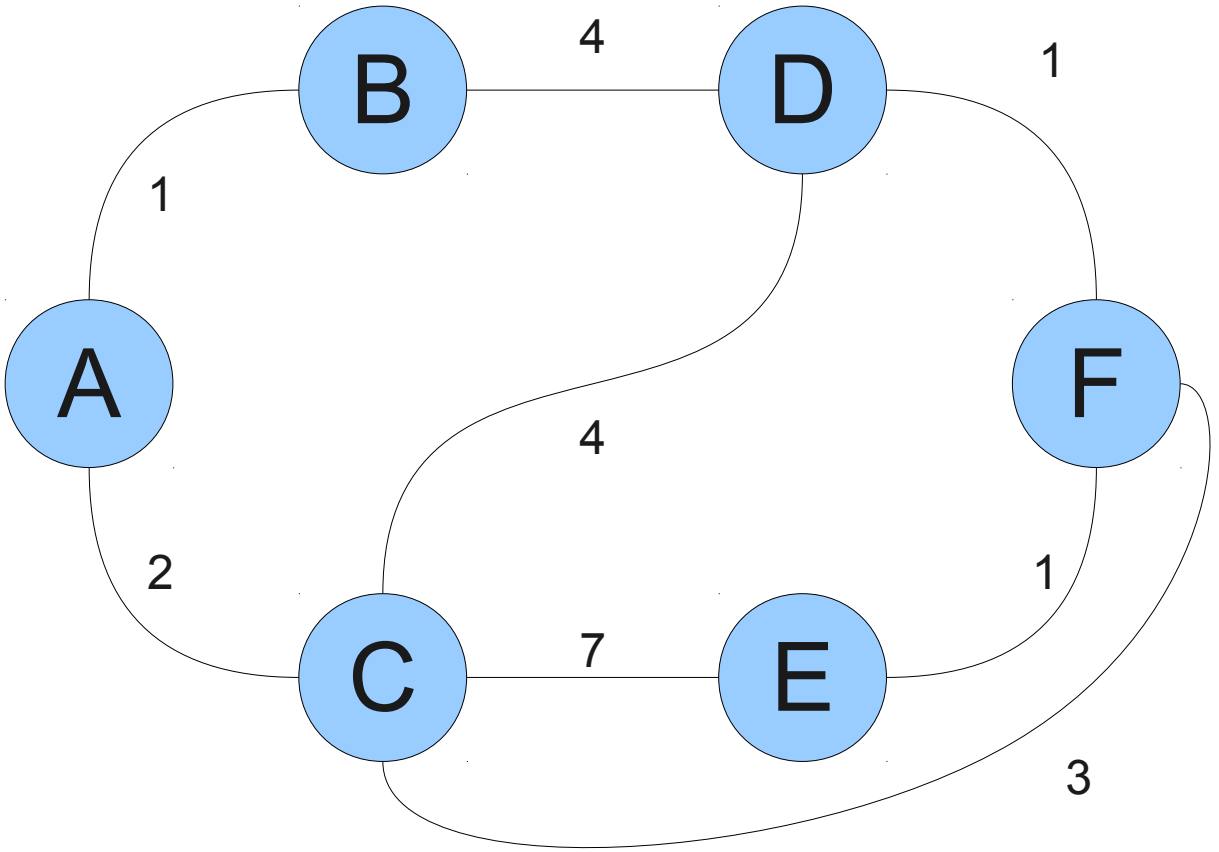
Trees!



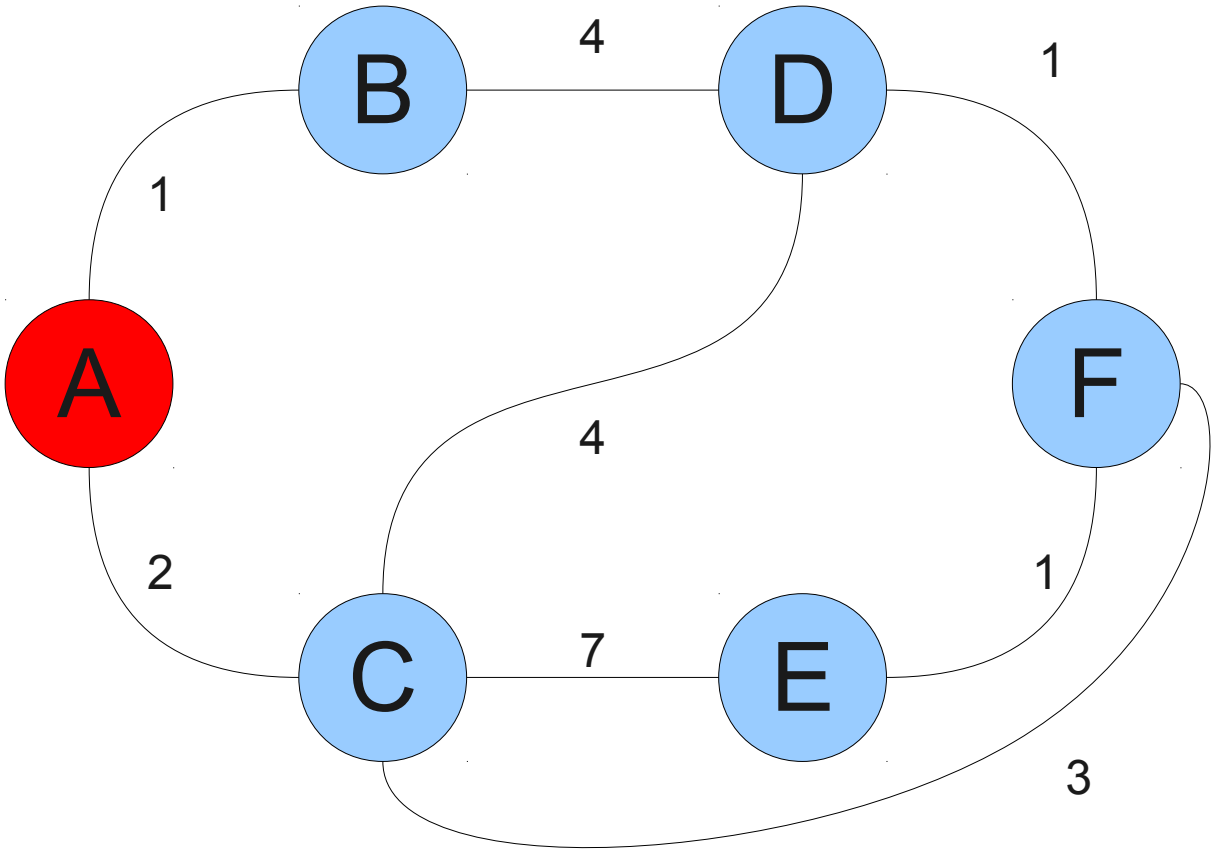
Trees!



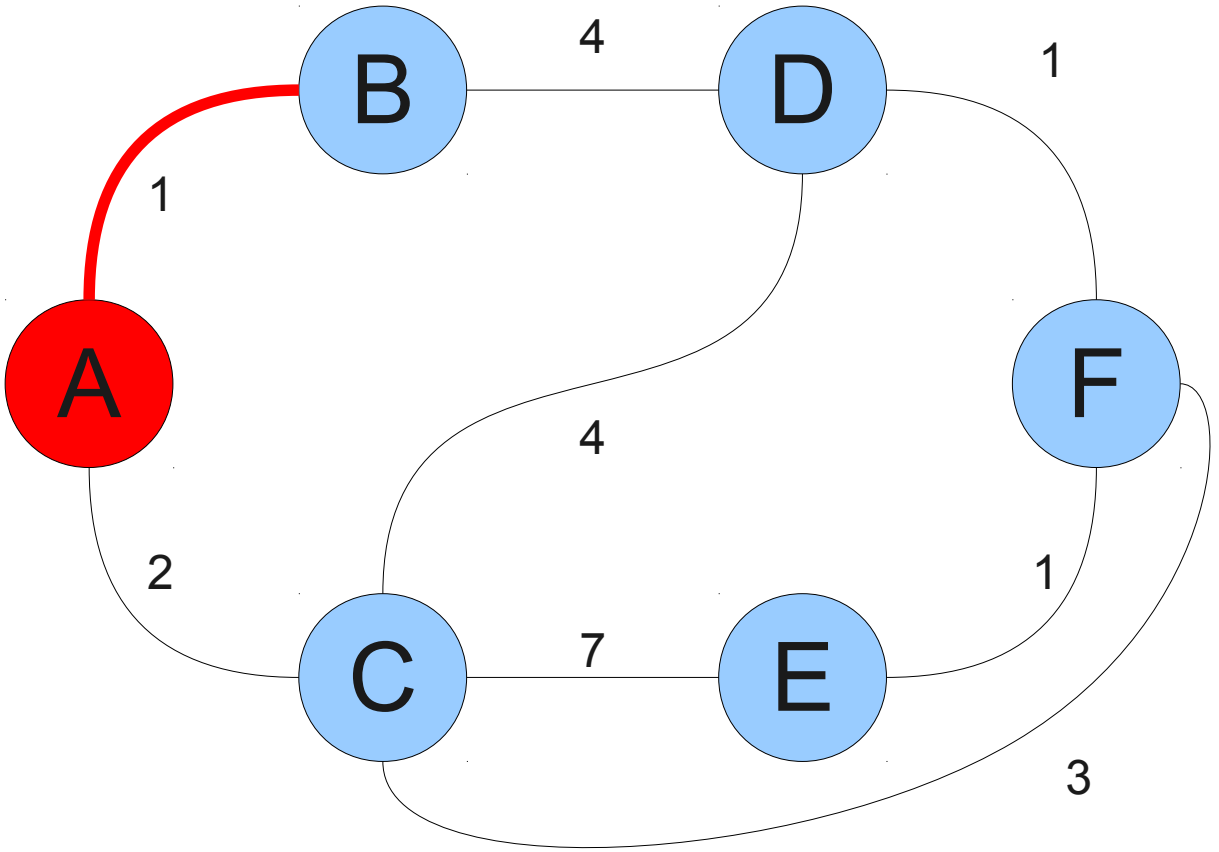
Trees!



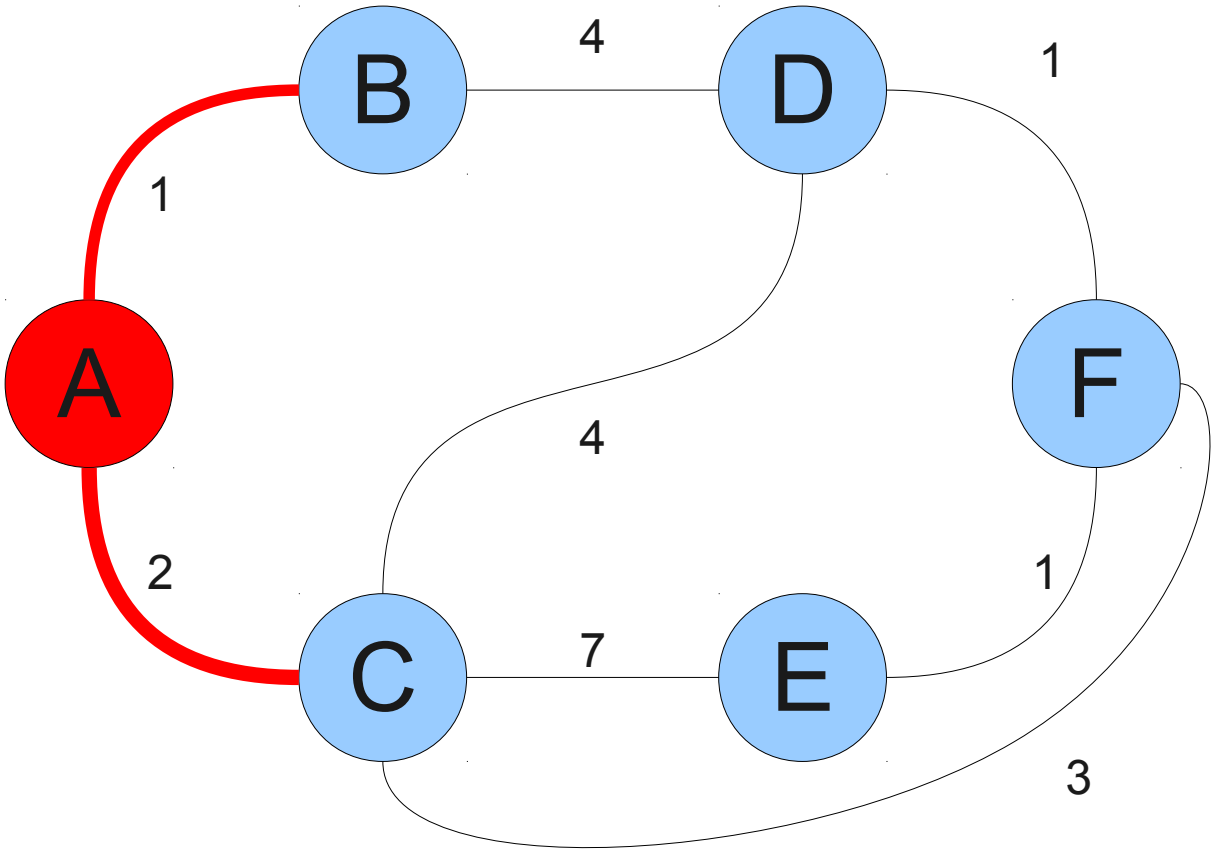
Trees!



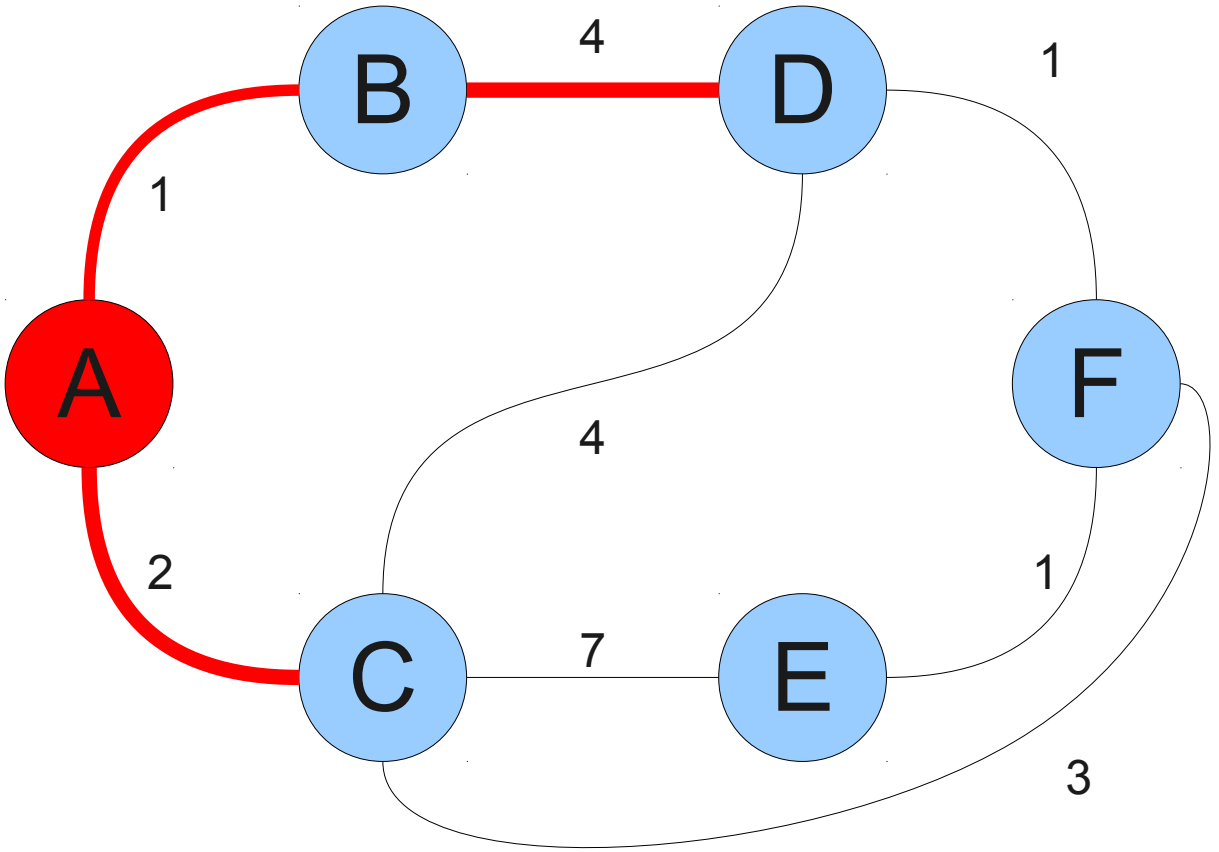
Trees!



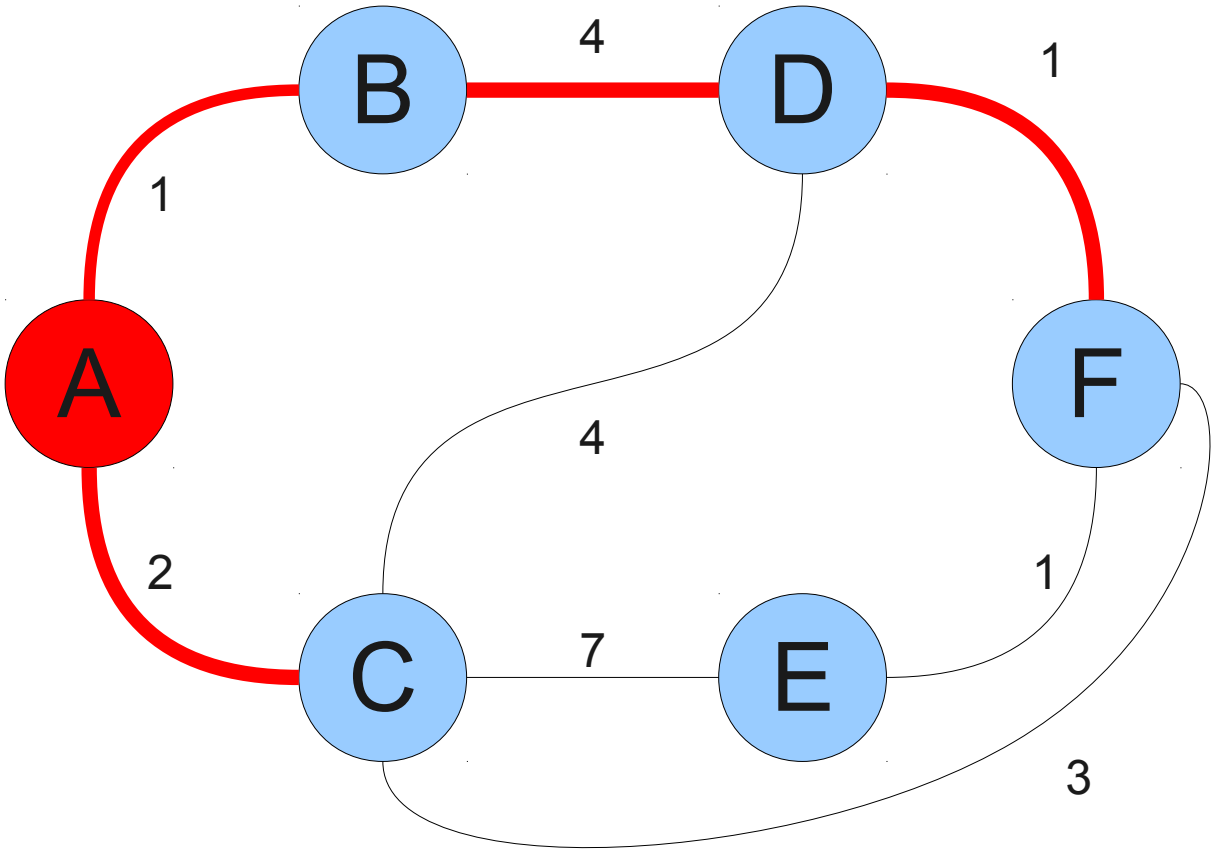
Trees!



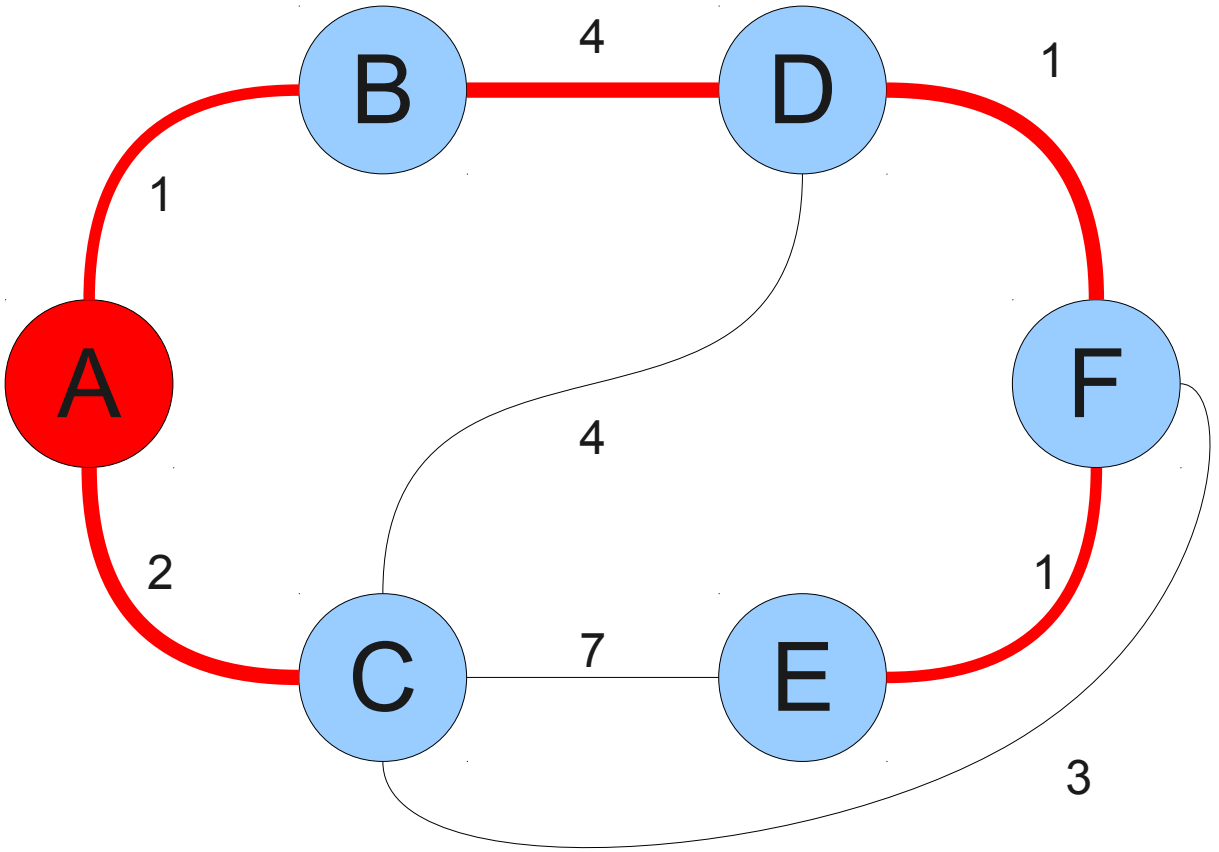
Trees!



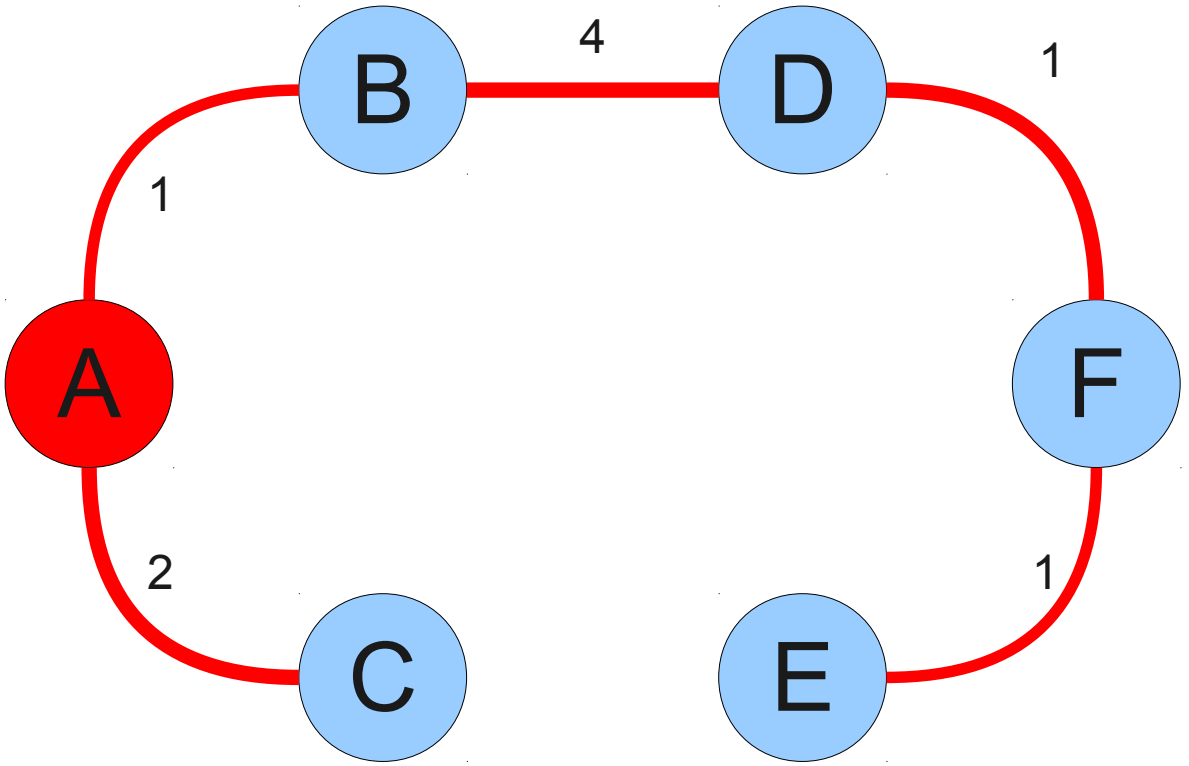
Trees!



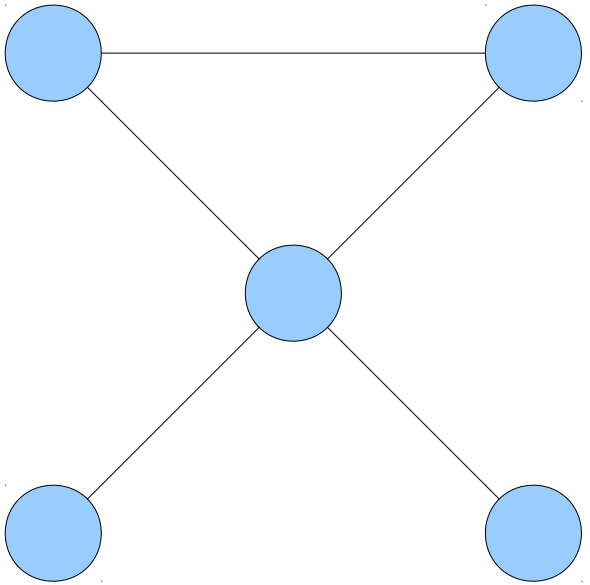
Trees!



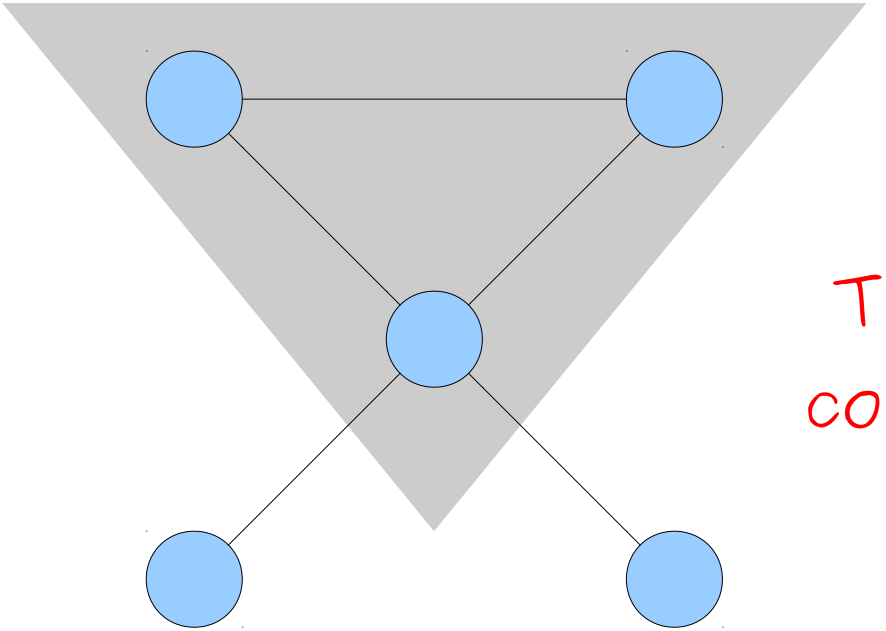
Trees!



Not a Tree!

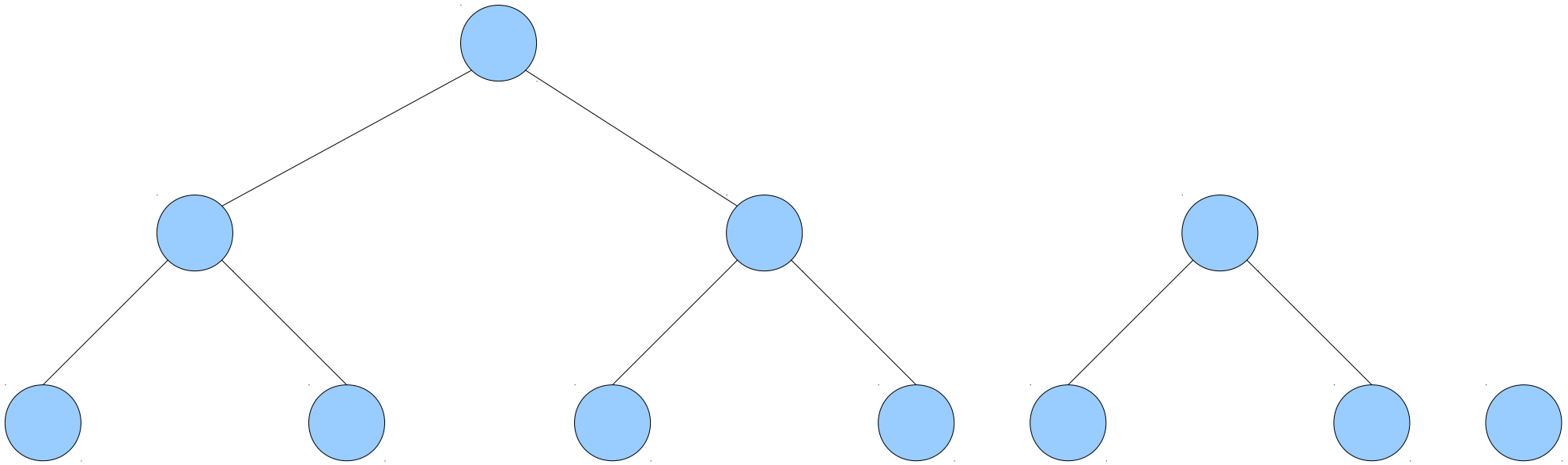


Not a Tree!



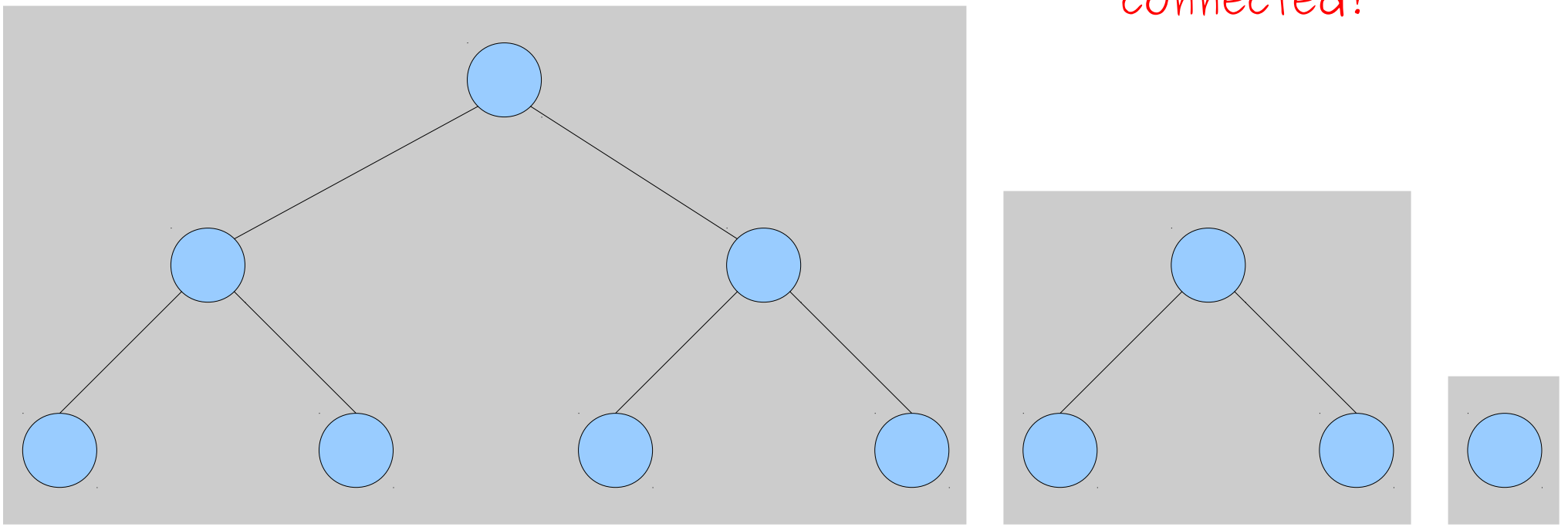
Trees cannot contain cycles!

Not a Tree!



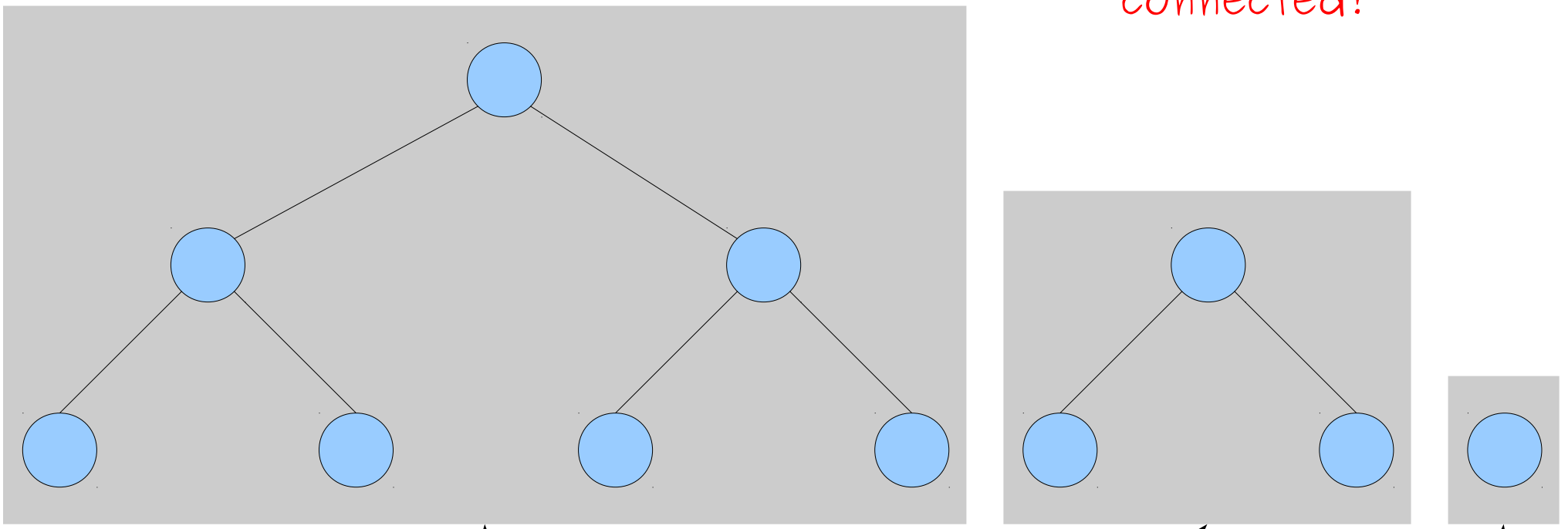
Not a Tree!

A tree must be connected!



Not a Tree!

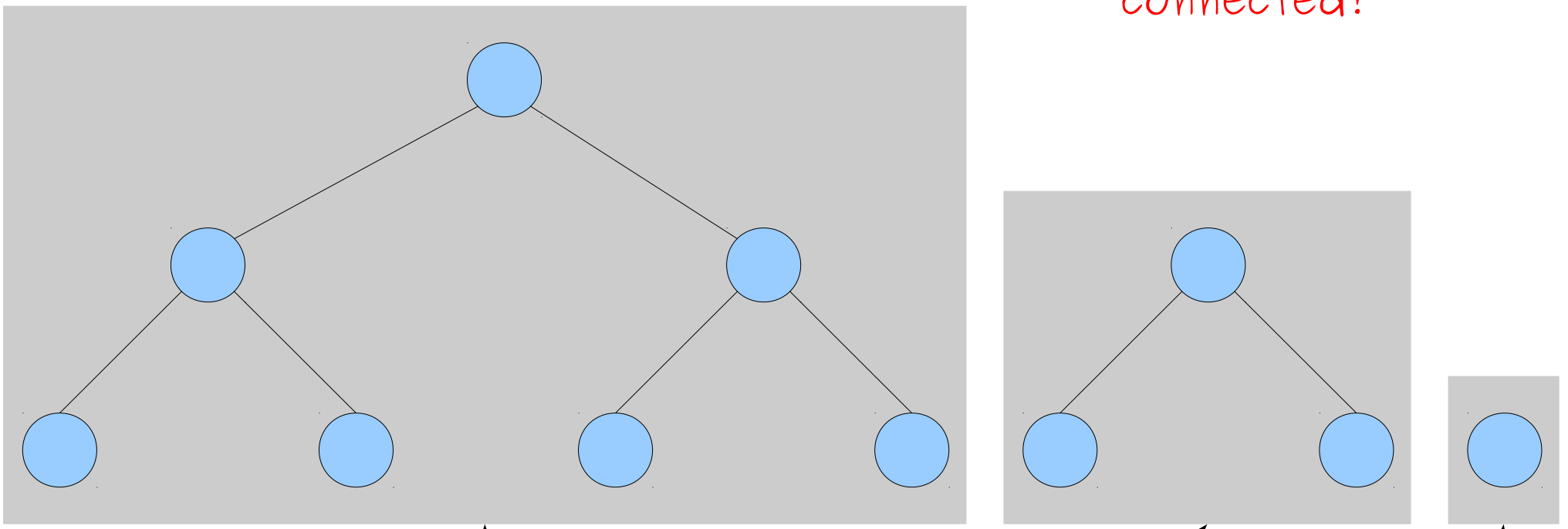
A tree must be connected!



(Though individually these are all trees)

Not a Tree!

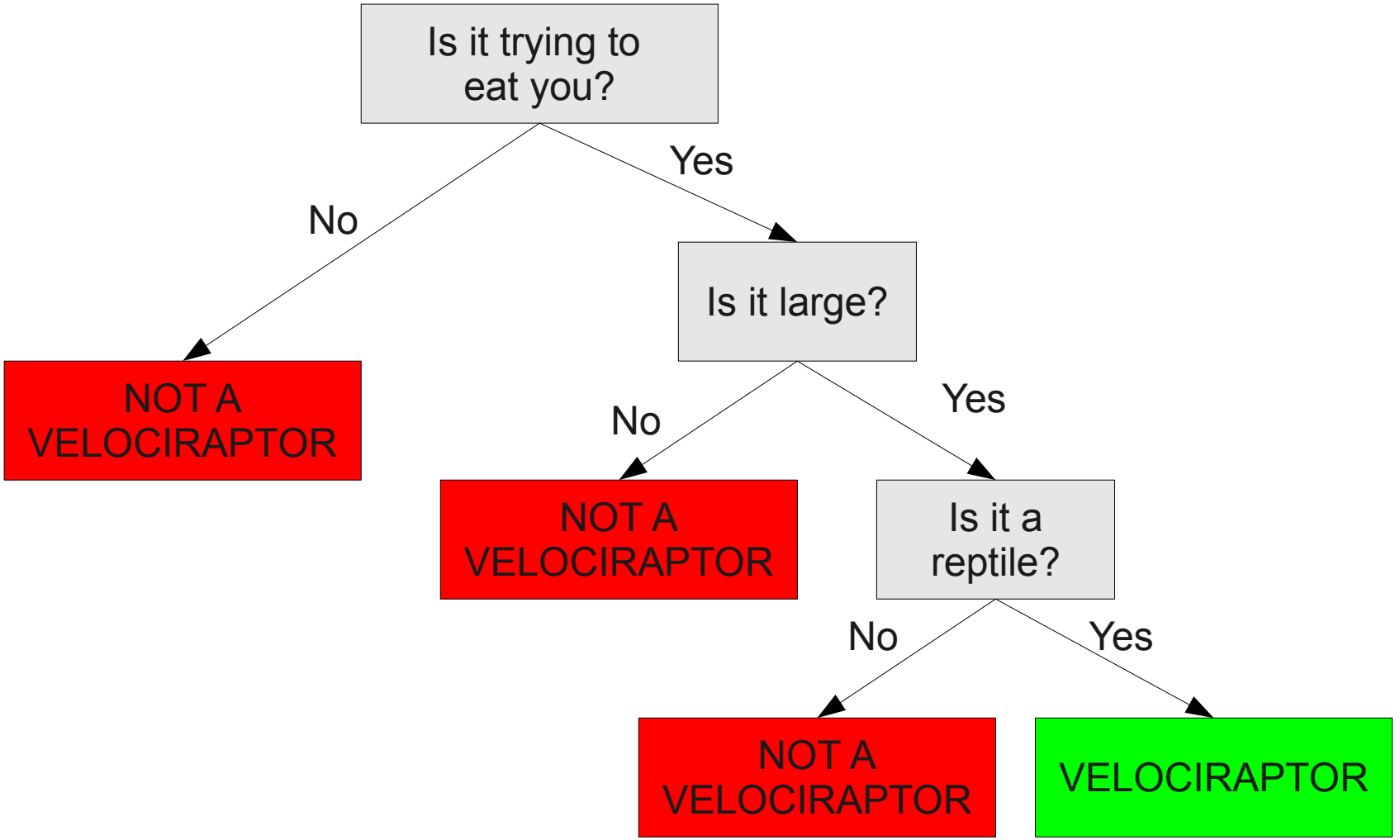
A tree must be connected!



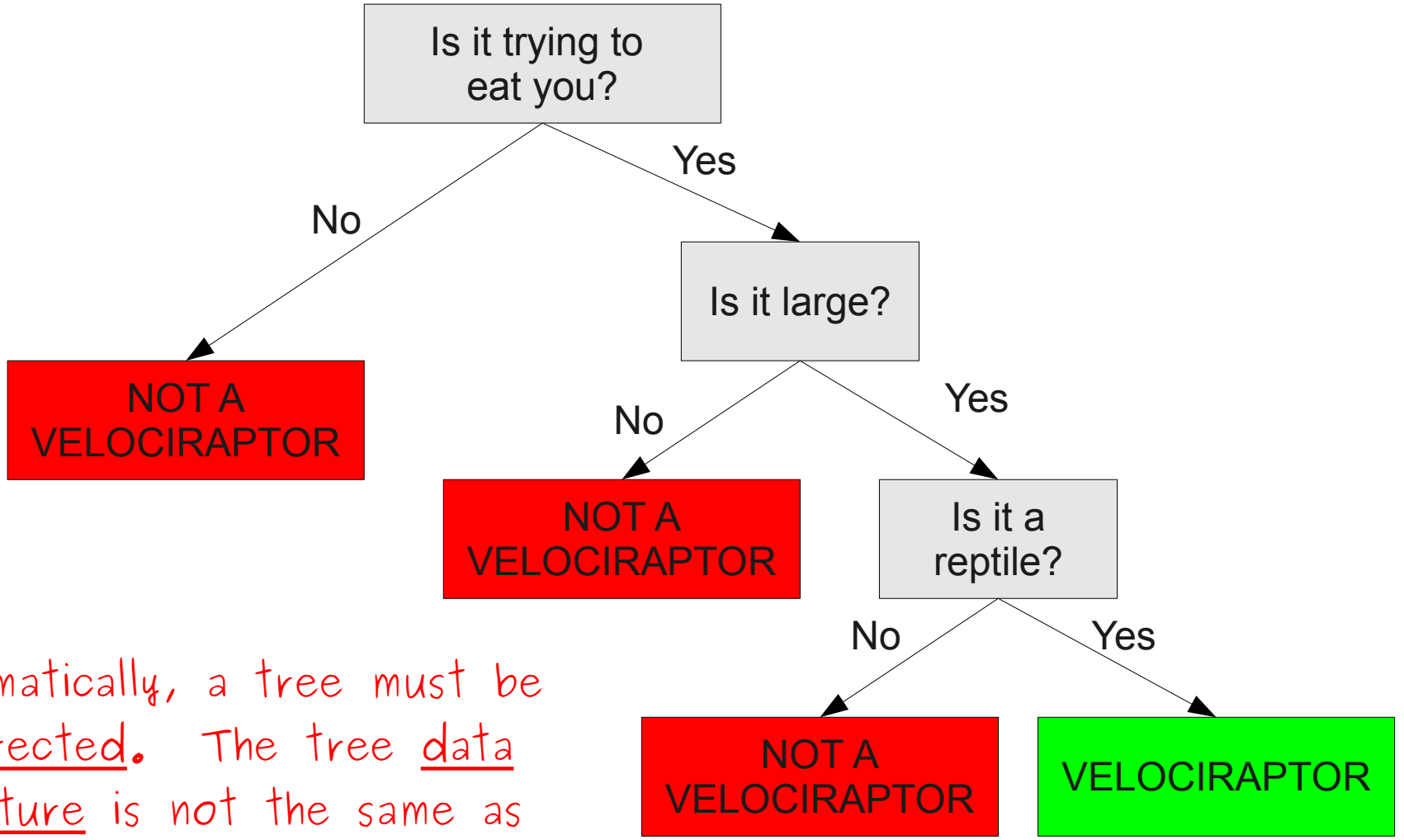
(Though individually these are all trees)

A graph that is a collection of trees is called a forest

Not a Tree!


Is it trying to eat you?
No
Yes
NOT A VELOCIRAPTOR
Is it large?
No
Yes
NOT A VELOCIRAPTOR
Is it a reptile?
No
Yes
NOT A VELOCIRAPTOR
VELOCIRAPTOR

Not a Tree!



Mathematically, a tree must be undirected. The tree data structure is not the same as a graph-theoretic tree.

Trees, Formally

- A **tree** is a graph G = (V, E) that is undirected, connected, and acyclic.
- **All of the following are equivalent**:
  - G is a tree.
  - Any two nodes in G are connected by a unique simple path.
  - G is **maximally acyclic** (the graph has no cycles, but adding any edge creates a cycle)
  - G is **minimally connected** (the graph is connected, but removing any edge disconnects the graph)

An Important Proof Technique

- To show that properties A, B, C, and D are all equivalent to one another, we can prove
  - A → B
  - B → C
  - C → D
  - D → A
- We will prove these three are equivalent and leave the fact that they are equivalent to trees as an exercise:
  - Any two nodes in G are connected by a unique simple path.
  - G is **maximally acyclic** (the graph has no cycles, but adding any edge creates a cycle)
  - G is **minimally connected** (the graph is connected, but removing any edge disconnects the graph)

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

Properties of Trees

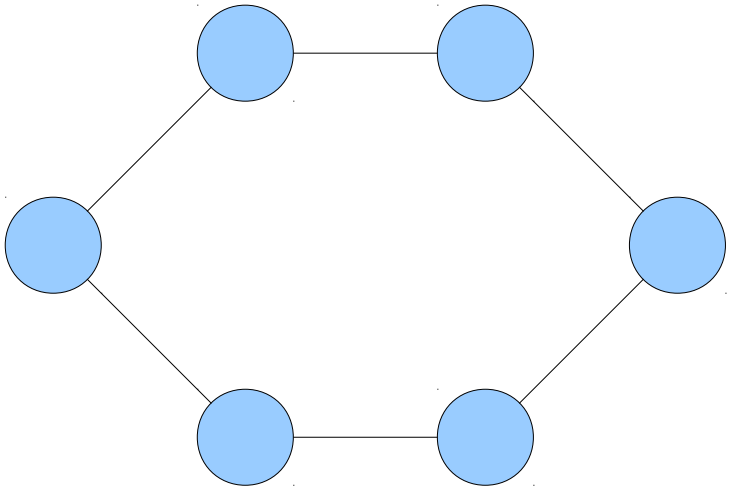
Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

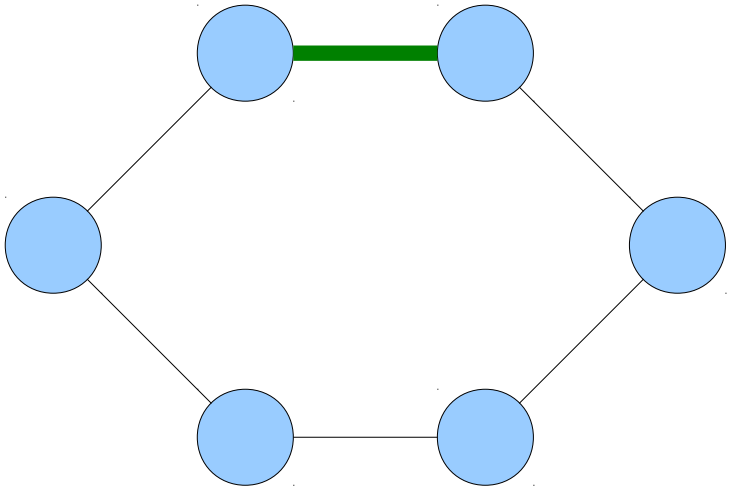
Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.



Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

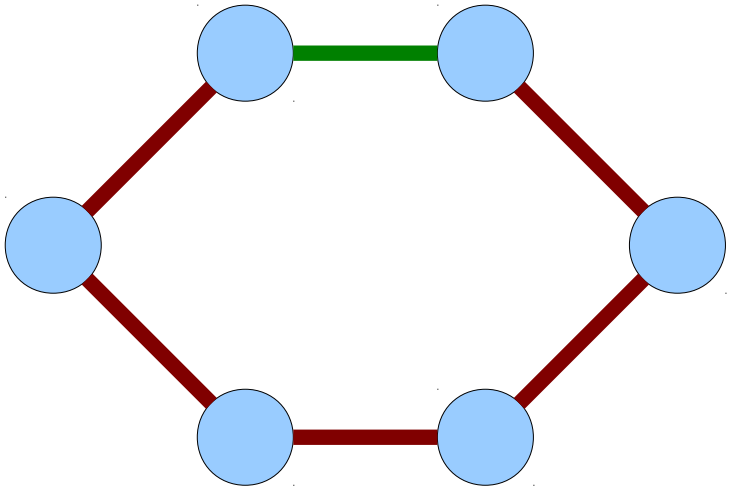
Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.



Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.



Properties of Trees

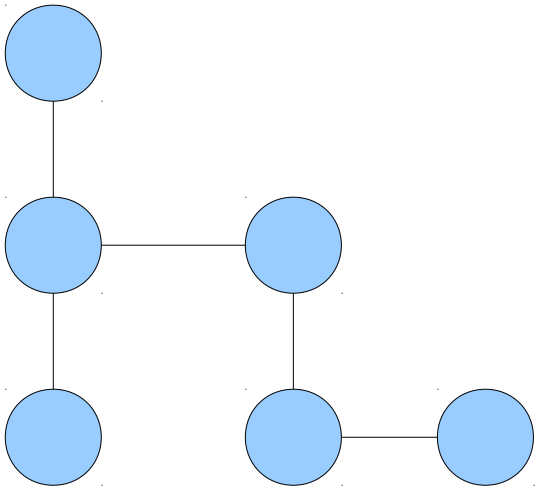
Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

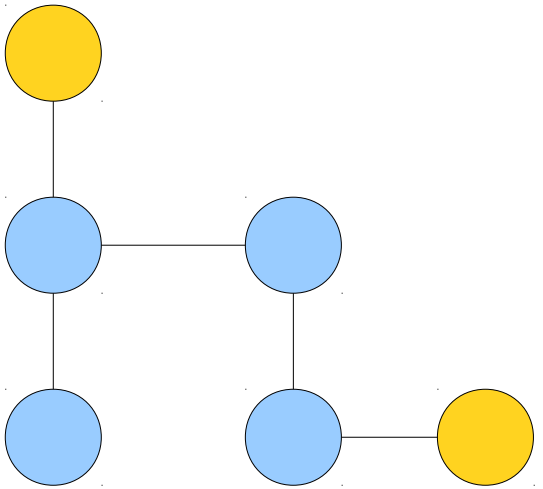
Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.



Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

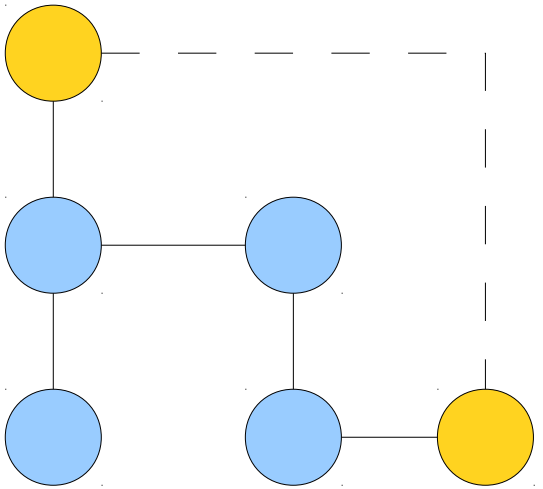
Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.



Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

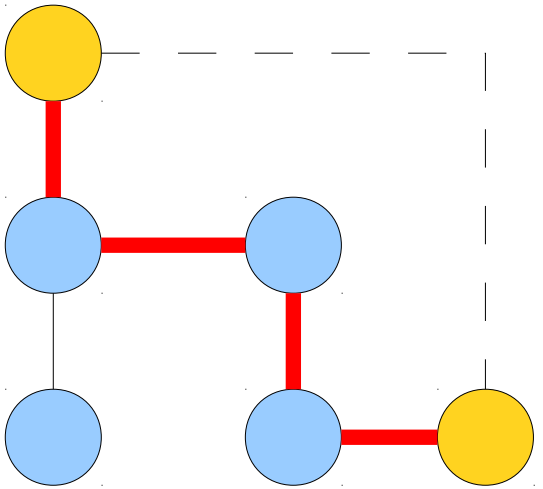
Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.



Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

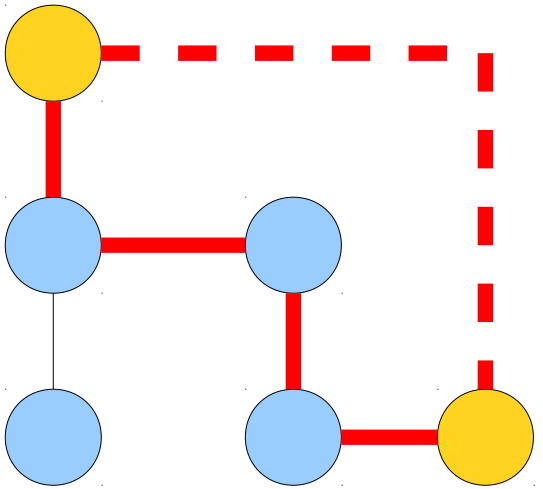
Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.



Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

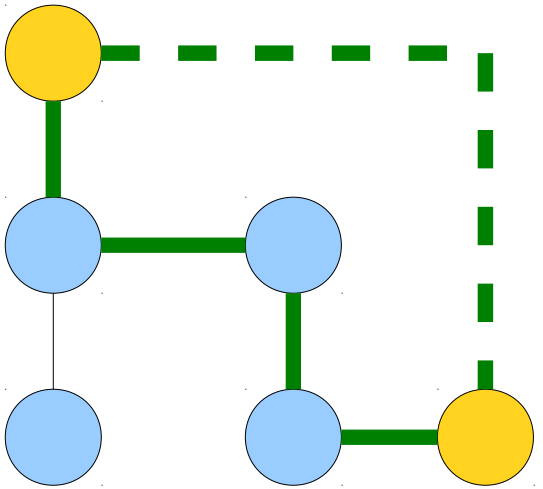
Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.



Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.



Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

To show that G contains no cycles, we proceed by contradiction and assume that it does contain a cycle; call it C.

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

To show that G contains no cycles, we proceed by contradiction and assume that it does contain a cycle; call it C. Let {u, v} be adjacent nodes in the cycle.

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

To show that G contains no cycles, we proceed by contradiction and assume that it does contain a cycle; call it C. Let {u, v} be adjacent nodes in the cycle. Then there are two simple paths from u to v – one formed by following {u, v}, and one formed by going around the cycle from u to v.

.

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

To show that G contains no cycles, we proceed by contradiction and assume that it does contain a cycle; call it C. Let {u, v} be adjacent nodes in the cycle. Then there are two simple paths from u to v – one formed by following {u, v}, and one formed by going around the cycle from u to v. This contradicts that every two nodes in G have a unique path between them, so our assumption was wrong and G is acyclic.

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

To show that G contains no cycles, we proceed by contradiction and assume that it does contain a cycle; call it C. Let {u, v} be adjacent nodes in the cycle. Then there are two simple paths from u to v – one formed by following {u, v}, and one formed by going around the cycle from u to v. This contradicts that every two nodes in G have a unique path between them, so our assumption was wrong and G is acyclic.

To show that adding an edge to G creates a cycle, consider any edge $\{u, v\} \notin E$.

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

To show that G contains no cycles, we proceed by contradiction and assume that it does contain a cycle; call it C. Let {u, v} be adjacent nodes in the cycle. Then there are two simple paths from u to v – one formed by following {u, v}, and one formed by going around the cycle from u to v. This contradicts that every two nodes in G have a unique path between them, so our assumption was wrong and G is acyclic.

To show that adding an edge to G creates a cycle, consider any edge $\{u, v\} \notin E$. Then if we add {u, v} to G, we form a cycle as follows:

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

To show that G contains no cycles, we proceed by contradiction and assume that it does contain a cycle; call it C. Let {u, v} be adjacent nodes in the cycle. Then there are two simple paths from u to v – one formed by following {u, v}, and one formed by going around the cycle from u to v. This contradicts that every two nodes in G have a unique path between them, so our assumption was wrong and G is acyclic.

To show that adding an edge to G creates a cycle, consider any edge $\{u, v\} \notin E$. Then if we add {u, v} to G, we form a cycle as follows: Let P be the unique simple path from u to v. Then we can extend P with {v, u} to form a cycle in G that begins and ends at u.

Properties of Trees

Theorem: Let G be an undirected graph. If any two nodes in G are connected by a unique simple path, then G is maximally acyclic.

Proof: Let G = (V, E) be a graph where any two nodes have a unique simple path between them. We need to show that G is acyclic, but that adding in any new edge creates a cycle.

To show that G contains no cycles, we proceed by contradiction and assume that it does contain a cycle; call it C. Let {u, v} be adjacent nodes in the cycle. Then there are two simple paths from u to v – one formed by following {u, v}, and one formed by going around the cycle from u to v. This contradicts that every two nodes in G have a unique path between them, so our assumption was wrong and G is acyclic.

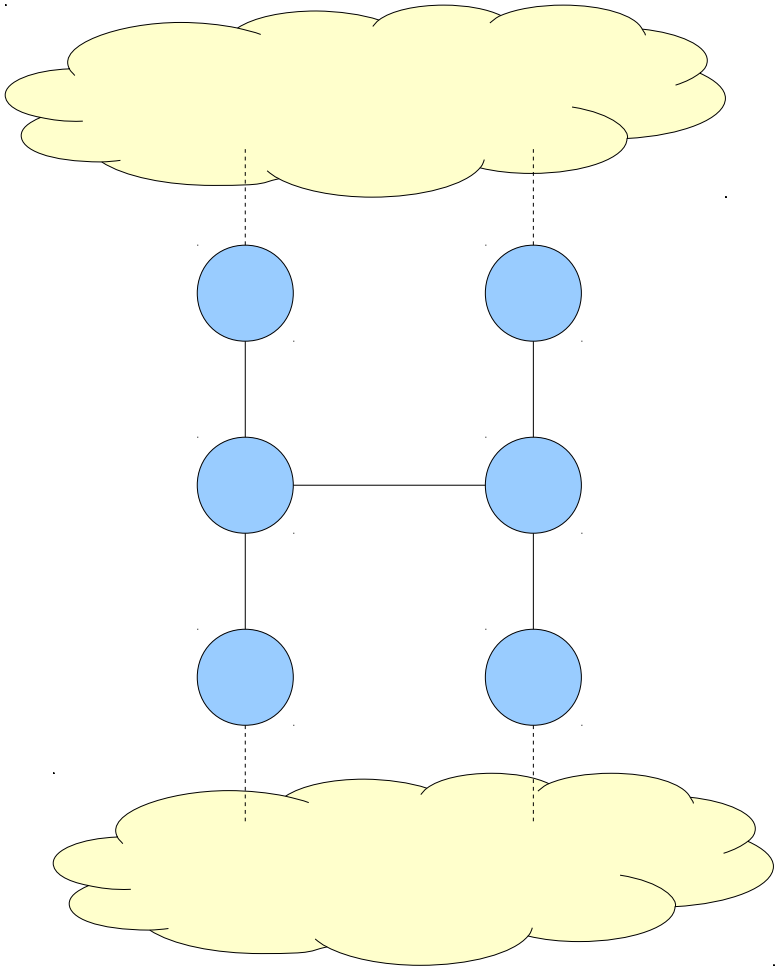
To show that adding an edge to G creates a cycle, consider any edge $\{u, v\} \notin E$. Then if we add {u, v} to G, we form a cycle as follows: Let P be the unique simple path from u to v. Then we can extend P with {v, u} to form a cycle in G that begins and ends at u. ■

Properties of Trees

Theorem: Let G be a maximally acyclic graph. Then G is minimally connected.

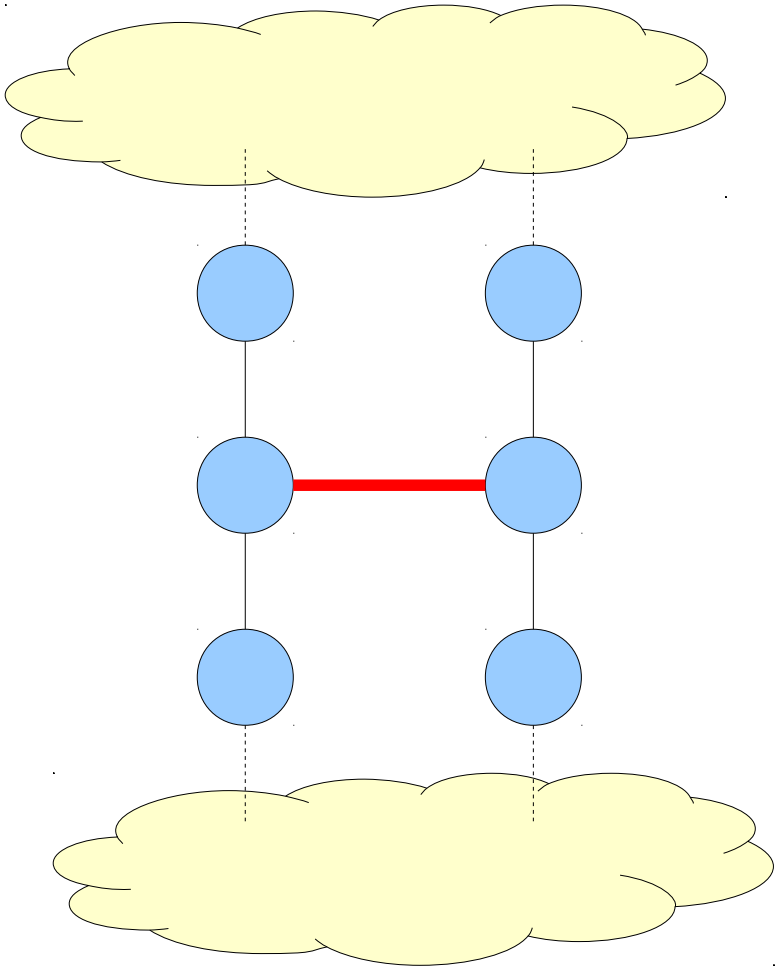
Properties of Trees

Theorem: Let G be a maximally acyclic graph. Then G is minimally connected.



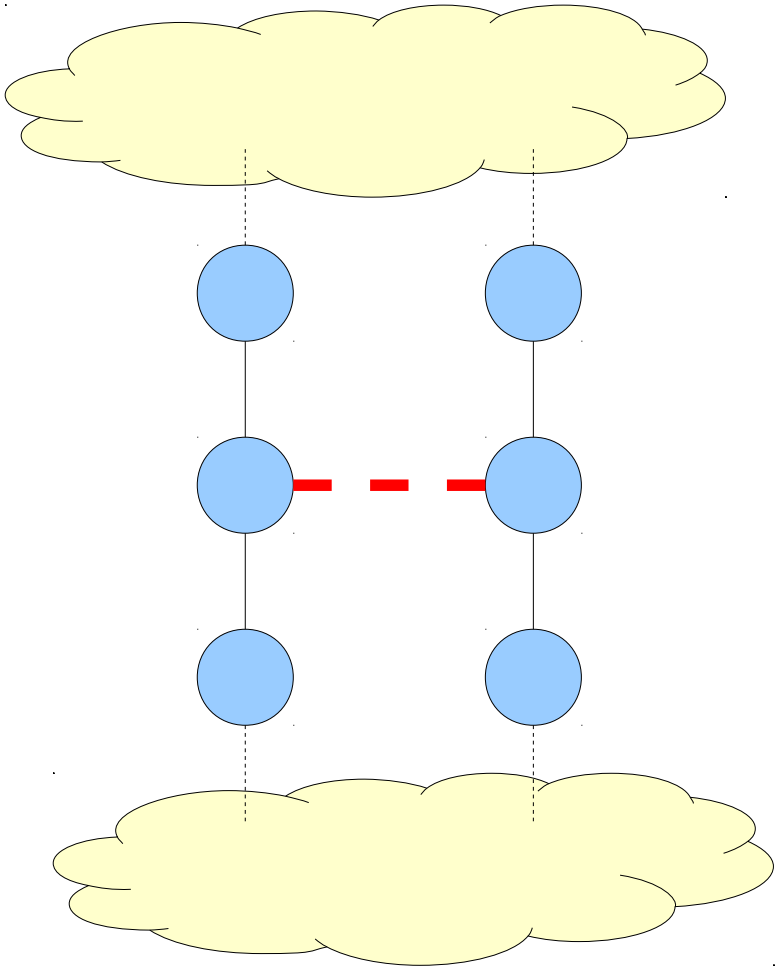
Properties of Trees

Theorem: Let G be a maximally acyclic graph. Then G is minimally connected.



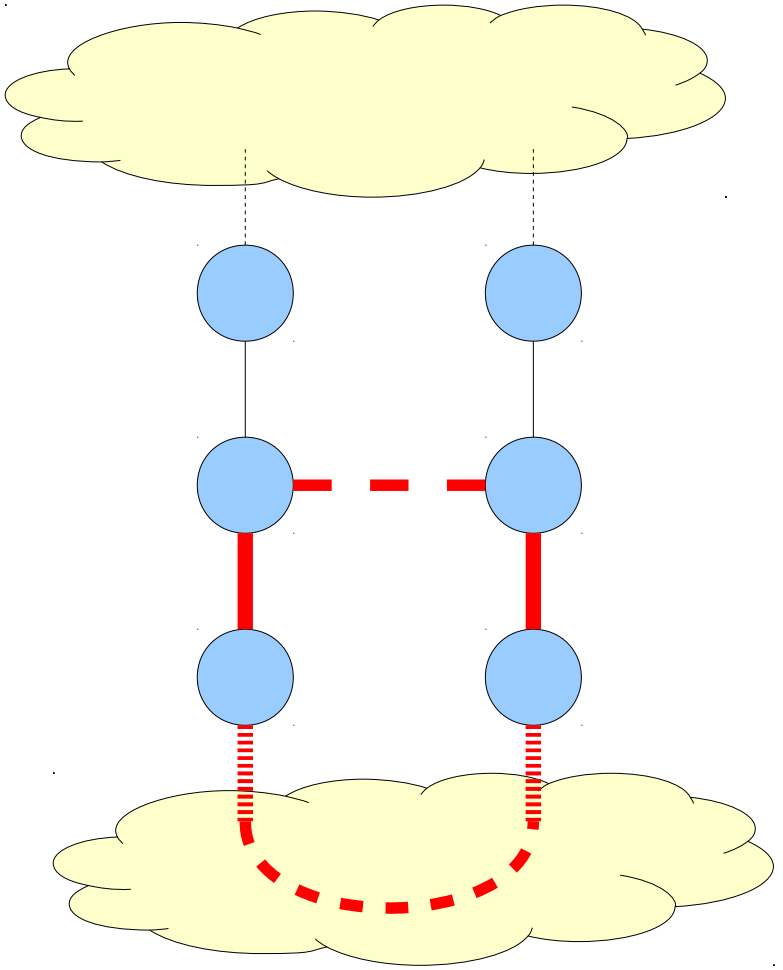
Properties of Trees

Theorem: Let G be a maximally acyclic graph. Then G is minimally connected.



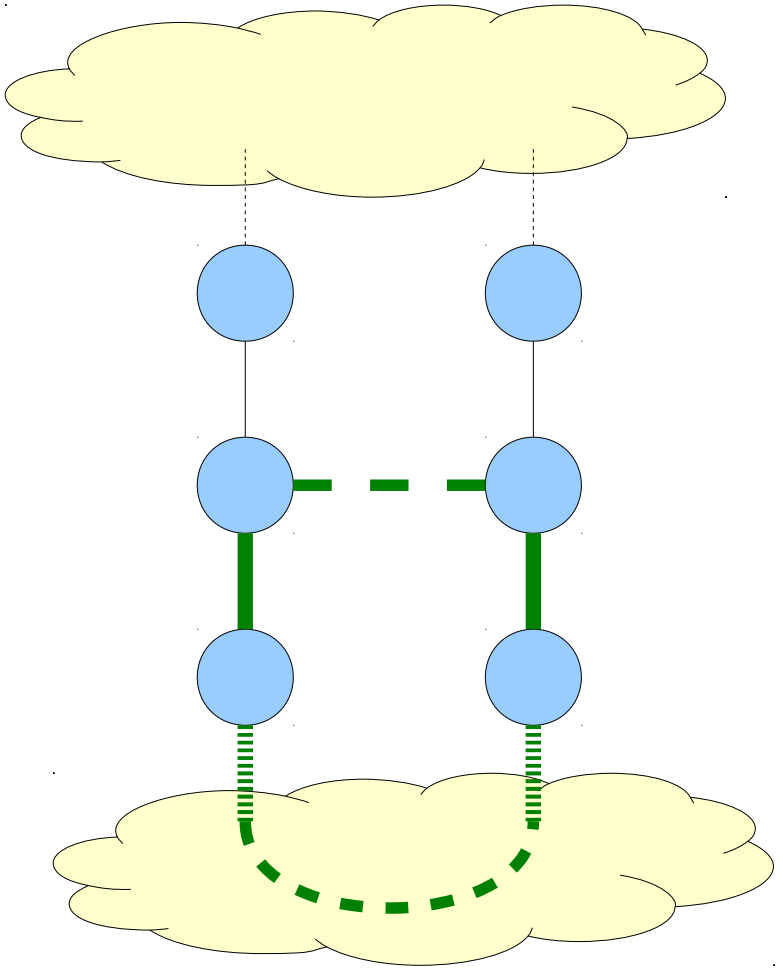
Properties of Trees

Theorem: Let G be a maximally acyclic graph. Then G is minimally connected.



Properties of Trees

Theorem: Let G be a maximally acyclic graph. Then G is minimally connected.



Properties of Trees

Theorem: Let G be a maximally acyclic graph. Then G is minimally connected.

Theorem: Let G be a maximally acyclic graph. Then G is minimally connected.

Proof: By contradiction; assume that G = (V, E) is maximally acyclic, but that it is not minimally connected. Thus the graph is either not connected or some edge can be removed without disconnecting the graph.

If G is not connected, then there must be some nodes u and v such that there is no path from u to v. Then we can add the edge {u, v} without introducing a cycle. To see this, note that this were to introduce a cycle, there would have to be some new cycle C that starts at u, then follows a path to v, then follows {u, v}. But this is impossible, since it would mean that u and v were already connected. Thus this edge does not introduce a cycle, but this contradicts the fact that G is maximally acyclic. So the graph must be connected.

Then there must be some edge $\{u, v\} \in E$ such that removing {u, v} from the graph does not disconnect it. Since the graph is not disconnected, there must be some path P from u to v. But then we could form a cycle in the original graph by following path P and then taking the edge {u, v}. We have reached a contradiction, so our assumption was wrong and G is minimally connected. ■

Properties of Trees

Theorem: If G is a minimally connected graph, then there is a unique simple path between any two nodes of G.

Properties of Trees

Theorem: If G is a minimally connected graph, then there is a unique simple path between any two nodes of G.

Properties of Trees

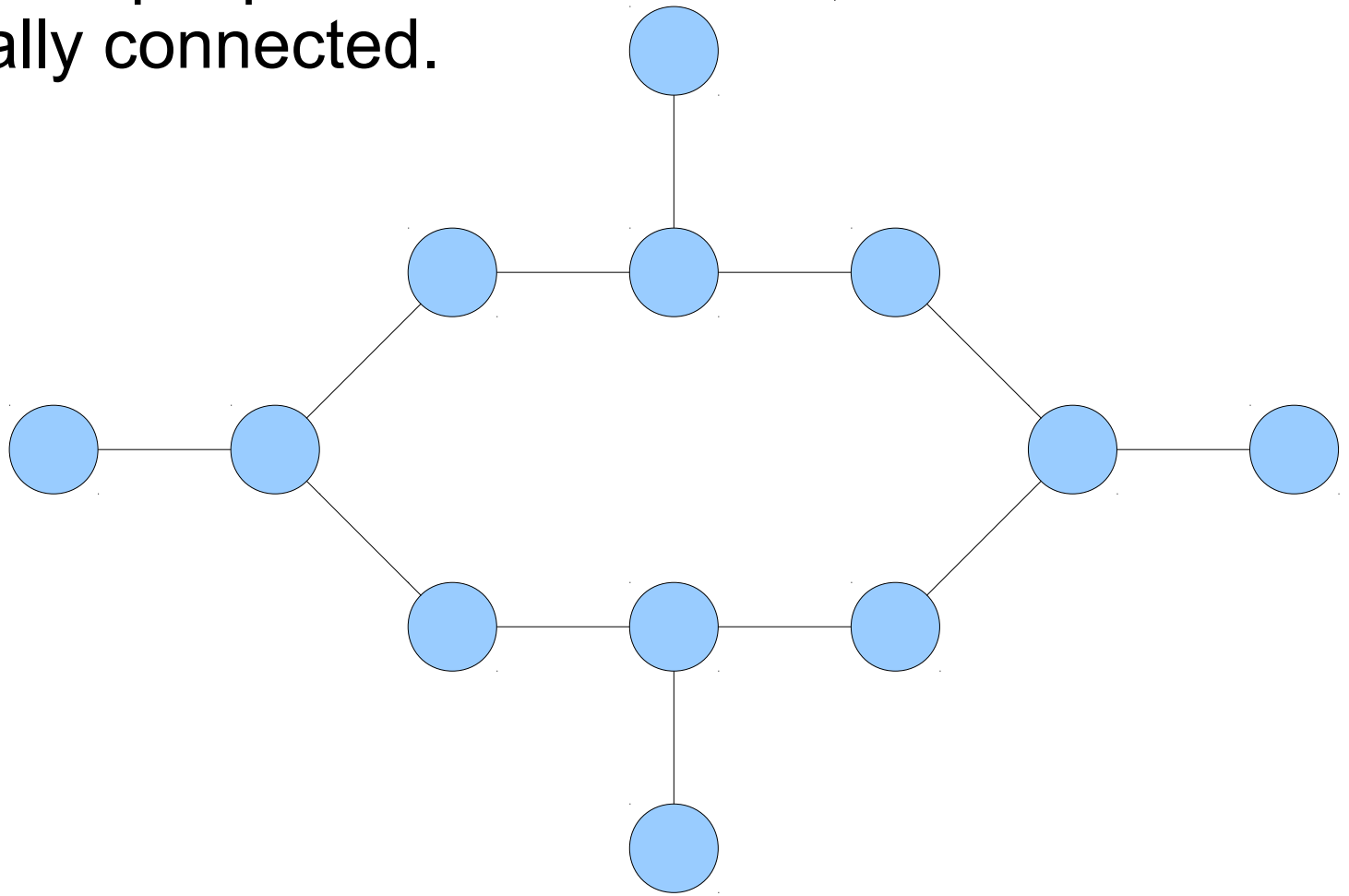
Theorem: If some pair of nodes in G does not have a unique simple path between them, then G is not minimally connected.

Properties of Trees

Theorem: If some pair of nodes in G does not have a unique simple path between them, then G is not minimally connected.

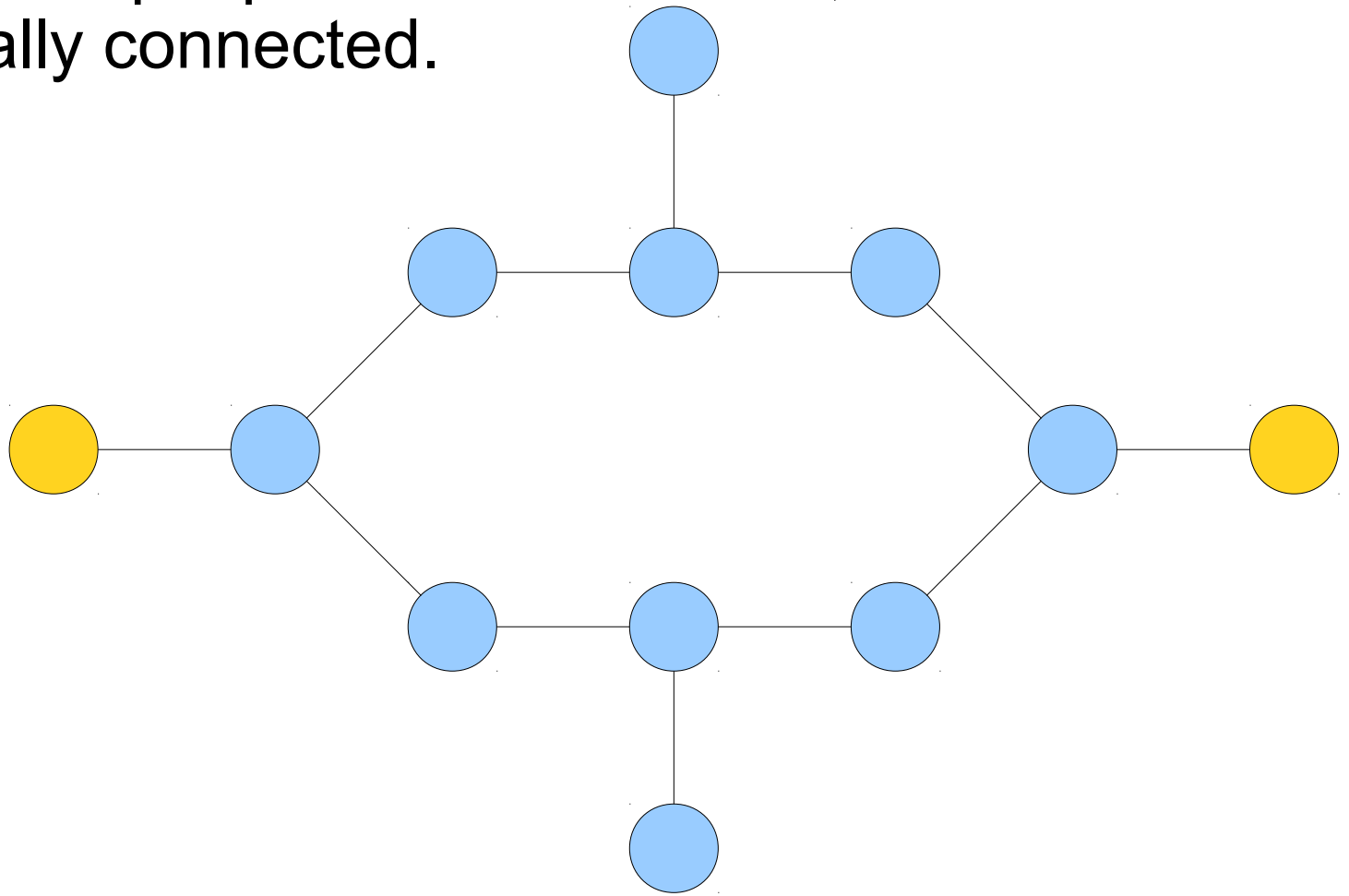
Properties of Trees

Theorem: If some pair of nodes in G does not have a unique simple path between them, then G is not minimally connected.



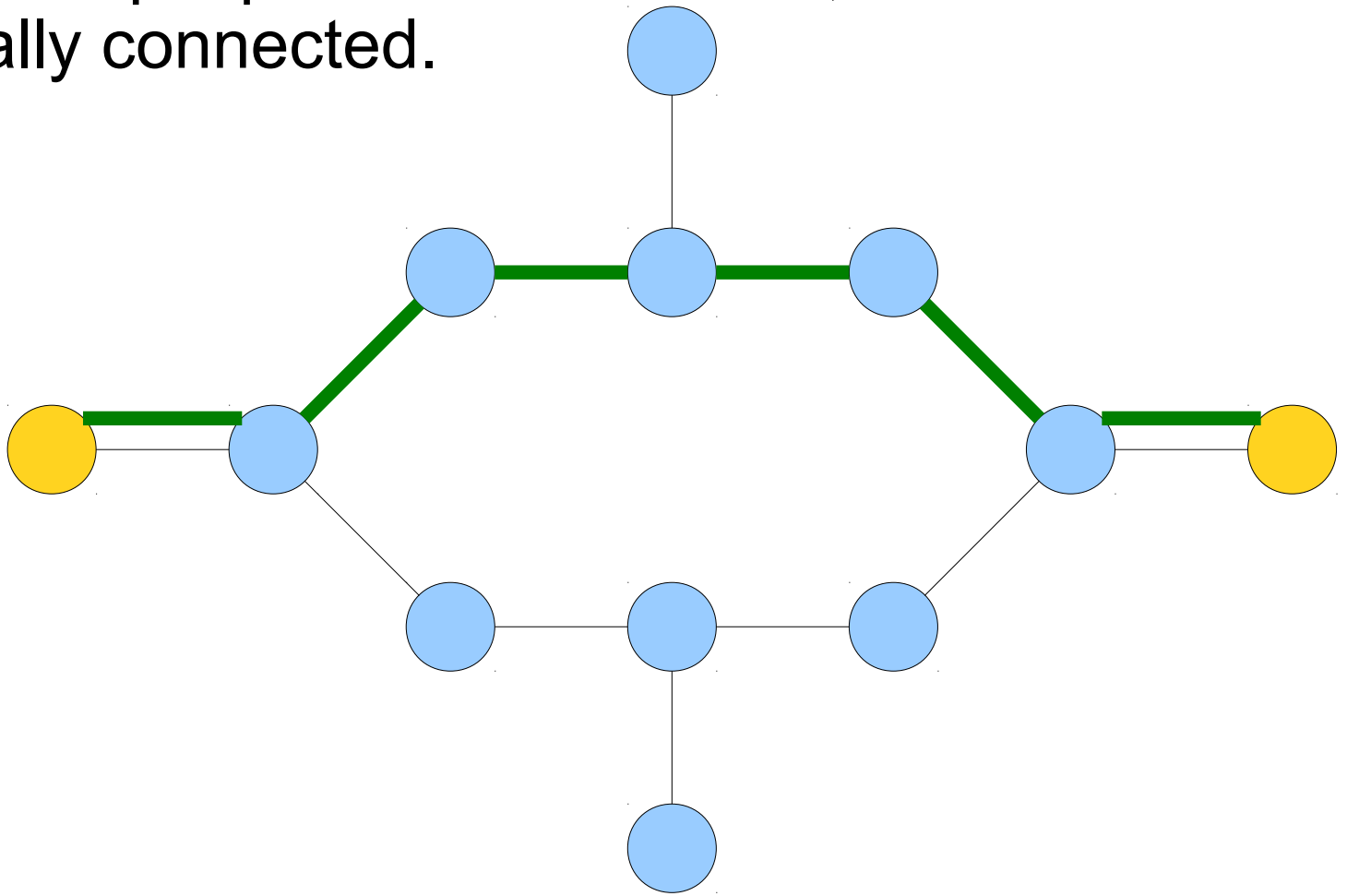
Properties of Trees

Theorem: If some pair of nodes in G does not have a unique simple path between them, then G is not minimally connected.



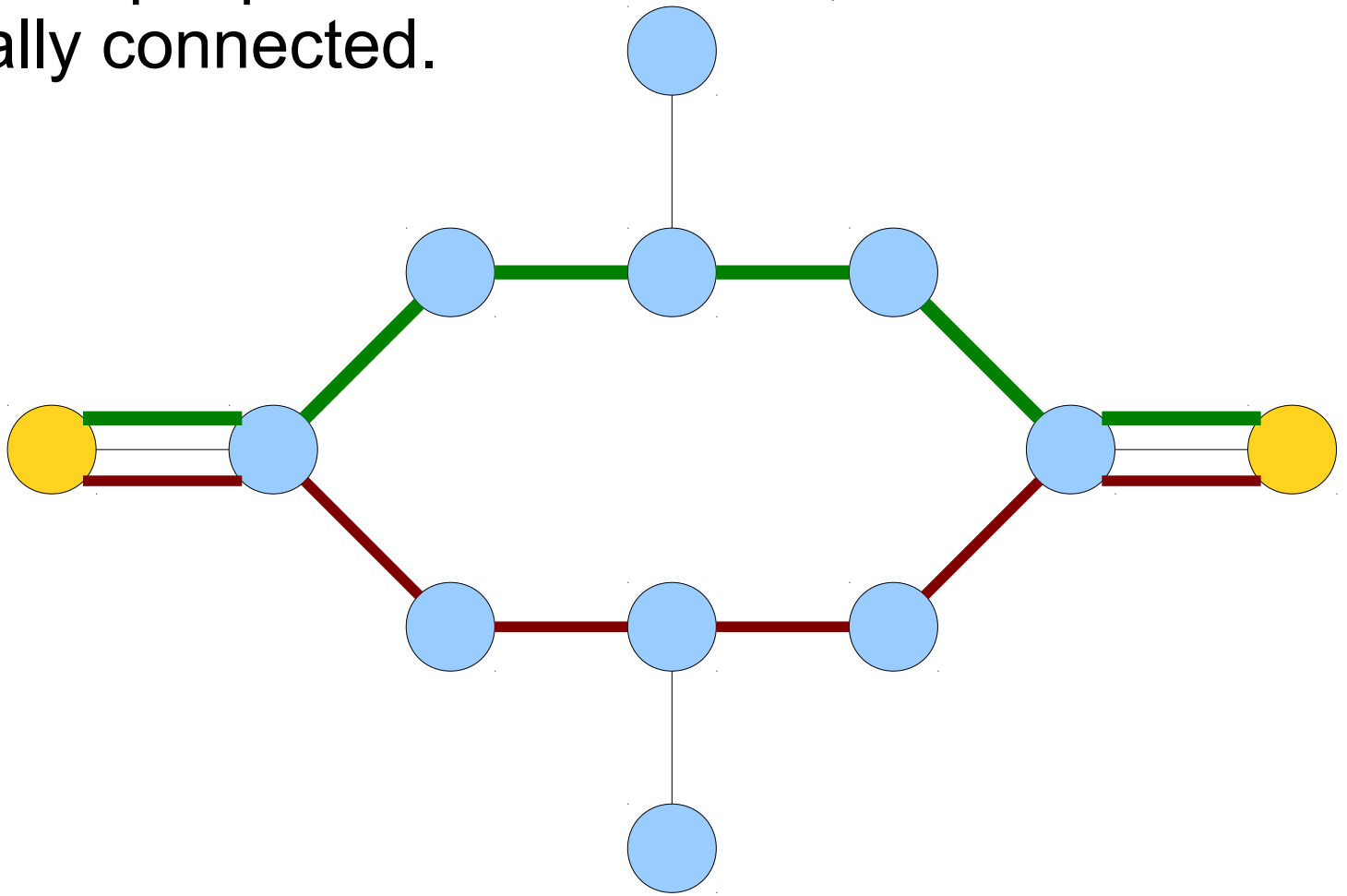
Properties of Trees

Theorem: If some pair of nodes in G does not have a unique simple path between them, then G is not minimally connected.



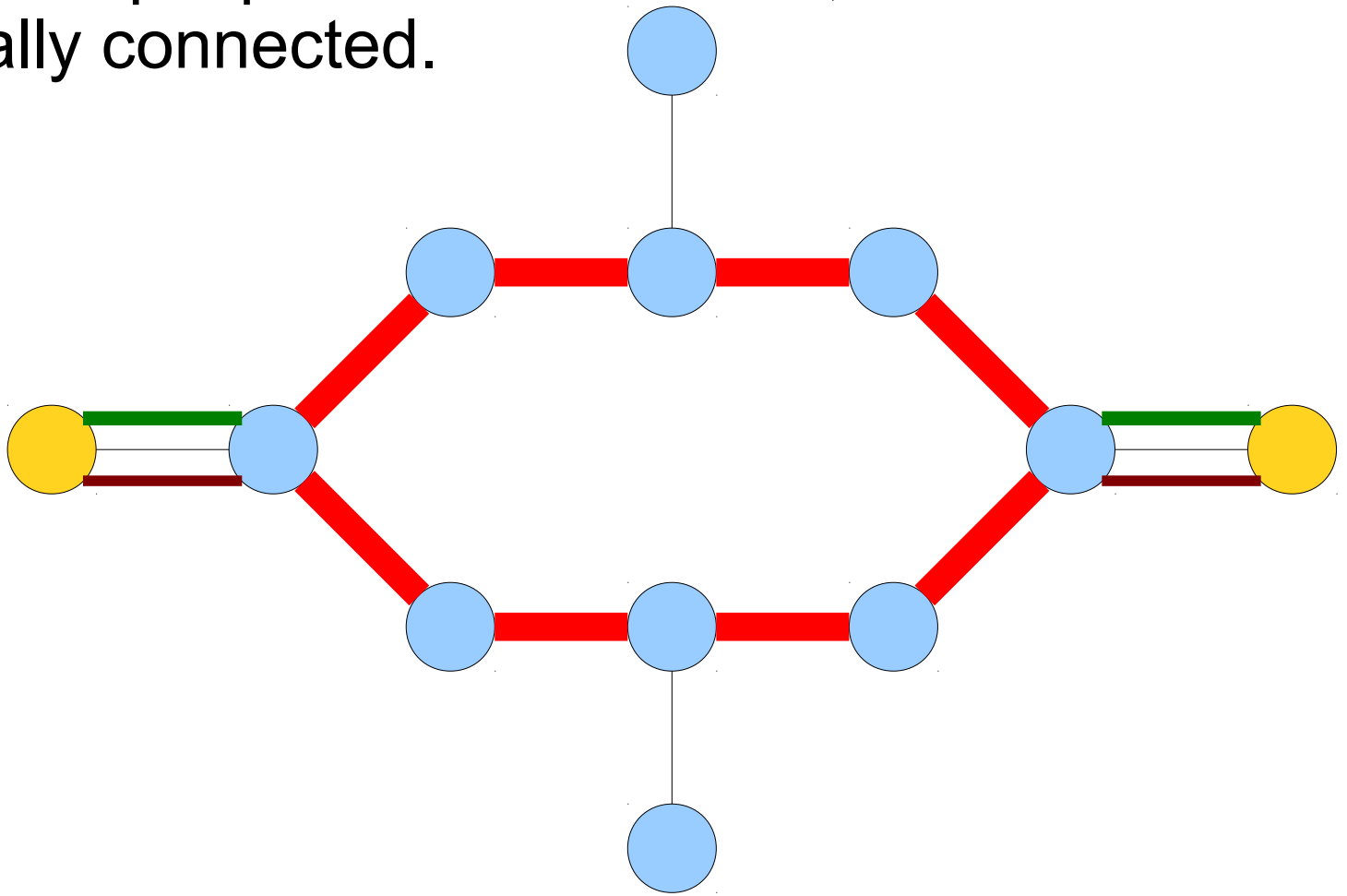
Properties of Trees

Theorem: If some pair of nodes in G does not have a unique simple path between them, then G is not minimally connected.



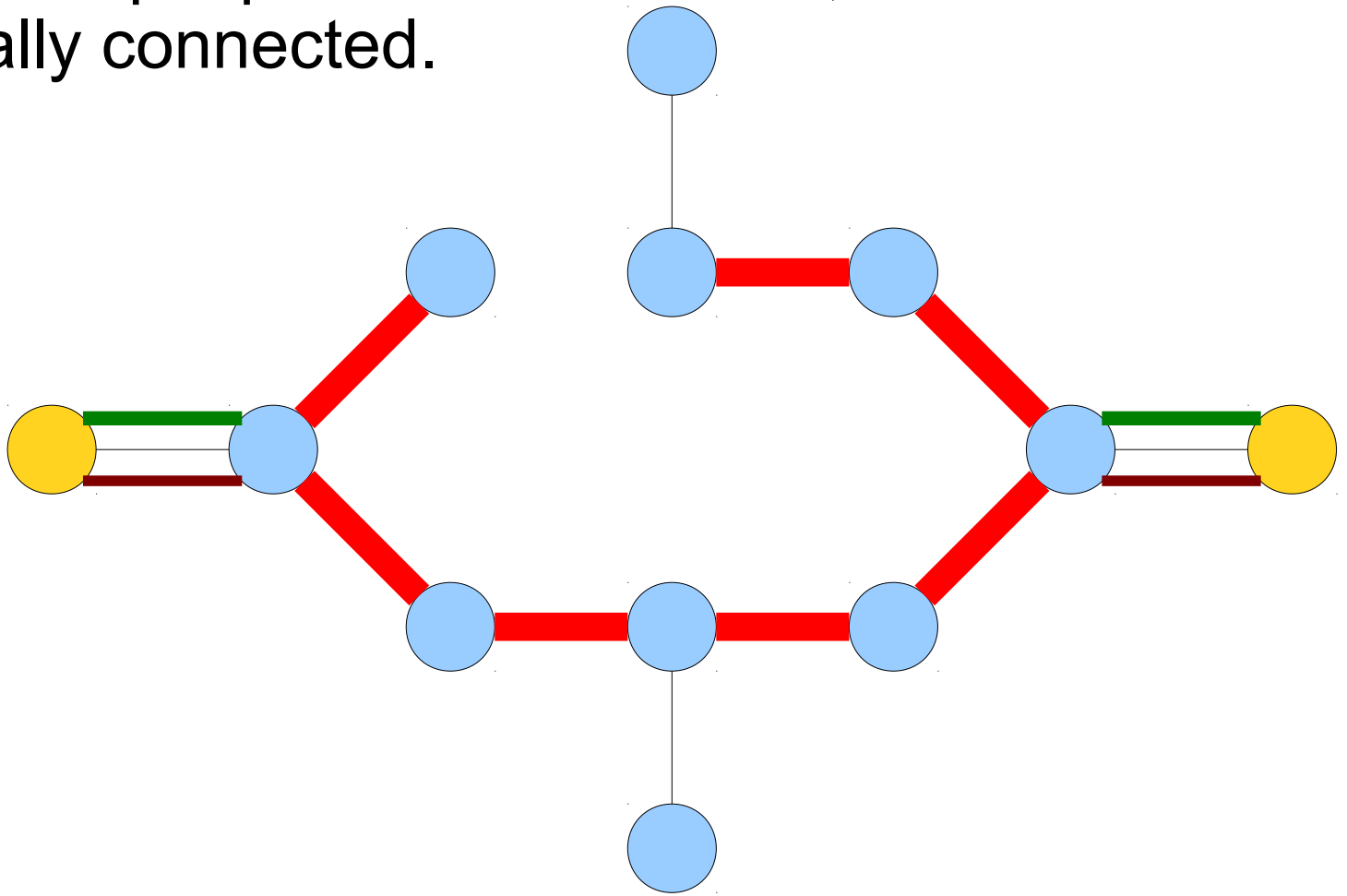
Properties of Trees

Theorem: If some pair of nodes in G does not have a unique simple path between them, then G is not minimally connected.



Properties of Trees

Theorem: If some pair of nodes in G does not have a unique simple path between them, then G is not minimally connected.



Properties of Trees

Theorem: Let G be a minimally connected graph. Then there is a unique path between any pair of nodes.

Proof: By contrapositive; we show that if there is not necessarily a unique path between any pair of nodes, that the graph is not minimally connected. If there is some pair of nodes u, v that have no path between them, then G is not connected. Otherwise, we have that all nodes have some path connecting them, but that there is some pair of nodes u and v such that there are at least two different paths from u to v; let them be P_1 and P_2. Because $P_1 \neq P_2$, the graph contains a cycle: follow path P_1 from u to v and path P_2 back from v to u. Then the cycle is either a simple cycle or it contains a simple cycle; call this cycle C.

We now claim that if we delete any edge {x, y} from C, the graph is still connected. To see this, consider any pair of nodes w, z in G. Since the graph is connected, there must be some path from w to z. If that path does not contain the edge {x, y}, then w and z are still connected even if we delete {x, y}. Otherwise, if the path does contain {x, y}, then modify the path as follows. Follow the path from w to x. Since {x, y} is an edge in a cycle, there must then be some path from x to y along the edges in the cycle other than {x, y}. Then, continue along the original path from y to z. In either case, there is a path from w to z, so the graph is still connected. Thus the graph is not minimally connected. ■

A Recap of Proof Techniques

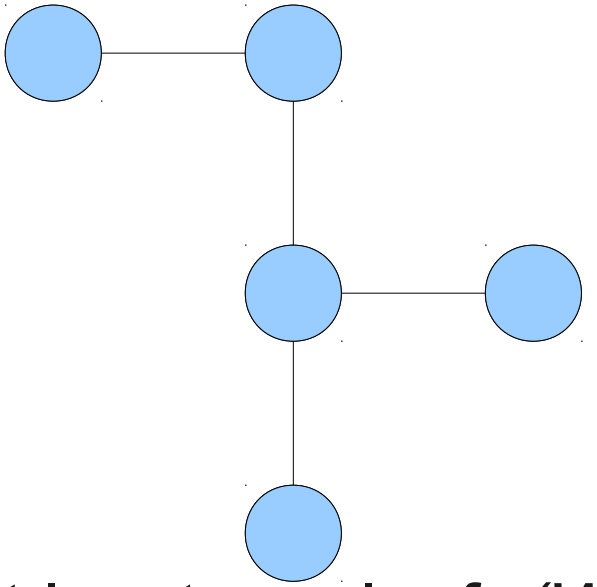
- Each of the three proofs we just did covered some fundamental proof idea:
  - **Direct proof**
  - **Proof by contradiction**
  - **Proof by contrapositive**
- What other techniques can we apply to trees?

A Recap of Proof Techniques

- Each of the three proofs we just did covered some fundamental proof idea:
  - **Direct proof**
  - **Proof by contradiction**
  - **Proof by contrapositive**
- What other techniques can we apply to trees?
  - **Proof by induction!**

Some Graph Terminology

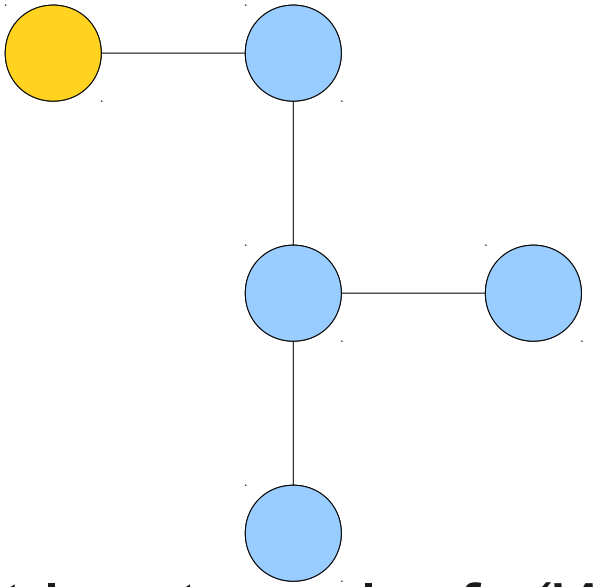
- A **leaf** in a tree is a node with degree at most one (at most one edge connected to it).



- Every tree has at least one leaf. *(Why?)*
- Removing a leaf from a tree of size ≥ 2 leaves the resulting graph a tree. *(Why?)*

Some Graph Terminology

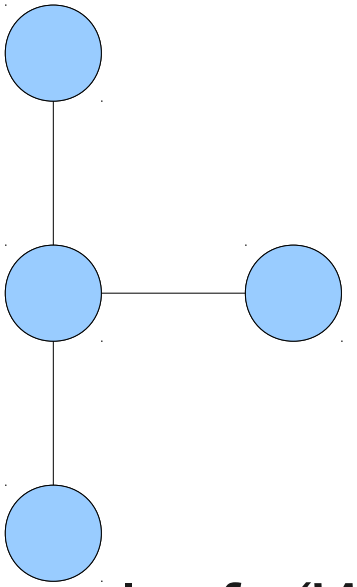
- A **leaf** in a tree is a node with degree at most one (at most one edge connected to it).



- Every tree has at least one leaf. *(Why?)*
- Removing a leaf from a tree of size ≥ 2 leaves the resulting graph a tree. *(Why?)*

Some Graph Terminology

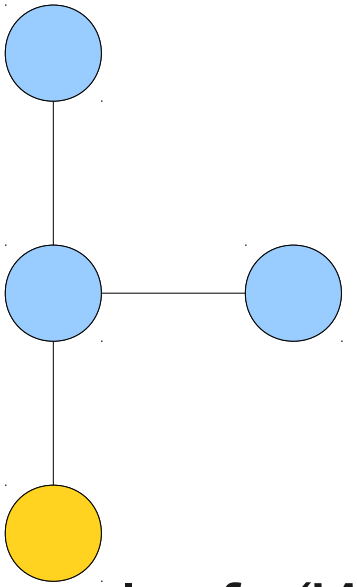
- A **leaf** in a tree is a node with degree at most one (at most one edge connected to it).



- Every tree has at least one leaf. *(Why?)*
- Removing a leaf from a tree of size ≥ 2 leaves the resulting graph a tree. *(Why?)*

Some Graph Terminology

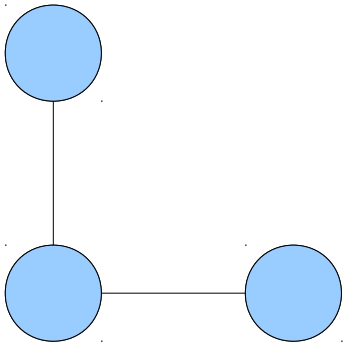
- A **leaf** in a tree is a node with degree at most one (at most one edge connected to it).



- Every tree has at least one leaf. *(Why?)*
- Removing a leaf from a tree of size ≥ 2 leaves the resulting graph a tree. *(Why?)*

Some Graph Terminology

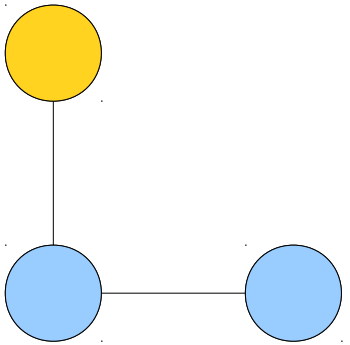
- A **leaf** in a tree is a node with degree at most one (at most one edge connected to it).



- Every tree has at least one leaf. *(Why?)*
- Removing a leaf from a tree of size ≥ 2 leaves the resulting graph a tree. *(Why?)*

Some Graph Terminology

- A **leaf** in a tree is a node with degree at most one (at most one edge connected to it).



- Every tree has at least one leaf. *(Why?)*
- Removing a leaf from a tree of size ≥ 2 leaves the resulting graph a tree. *(Why?)*

Some Graph Terminology

- A **leaf** in a tree is a node with degree at most one (at most one edge connected to it).

- Every tree has at least one leaf. *(Why?)*
- Removing a leaf from a tree of size ≥ 2 leaves the resulting graph a tree. *(Why?)*

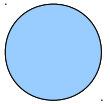
Some Graph Terminology

- A **leaf** in a tree is a node with degree at most one (at most one edge connected to it).

- Every tree has at least one leaf. *(Why?)*
- Removing a leaf from a tree of size ≥ 2 leaves the resulting graph a tree. *(Why?)*

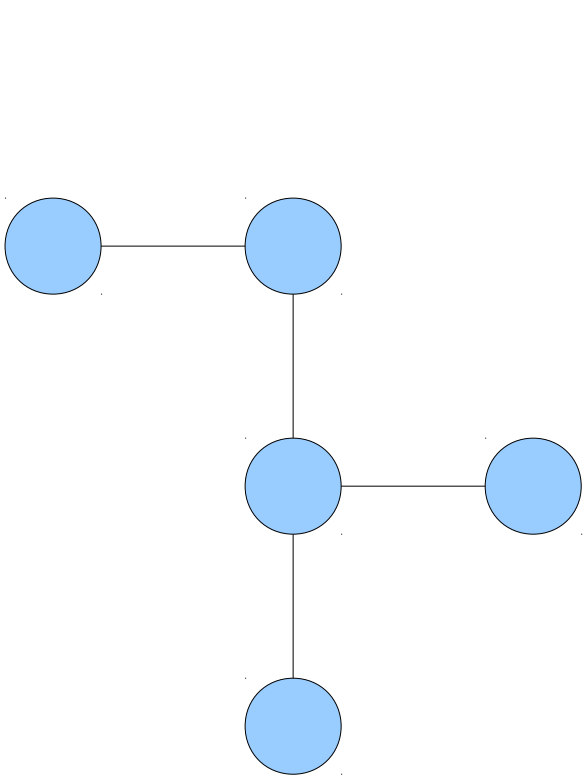
Some Graph Terminology

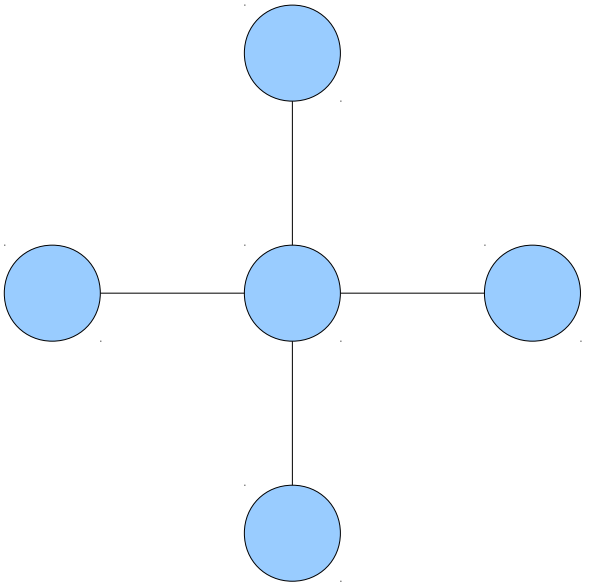
- A **leaf** in a tree is a node with degree at most one (at most one edge connected to it).

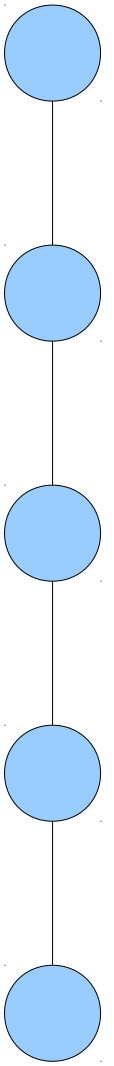


- Every tree has at least one leaf. *(Why?)*
- Removing a leaf from a tree of size ≥ 2 leaves the resulting graph a tree. *(Why?)*

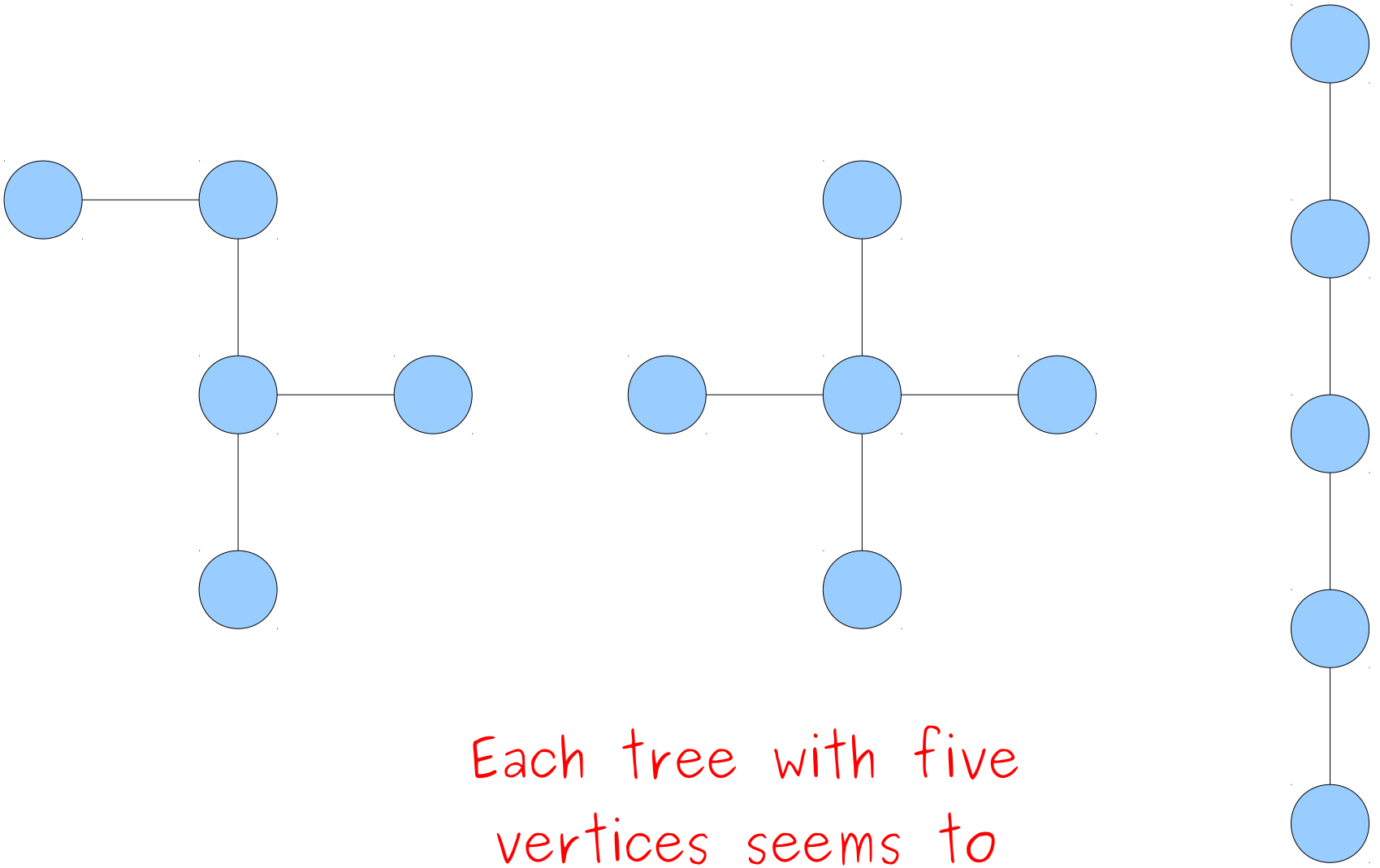
Some Interesting Totals







Some Interesting Totals



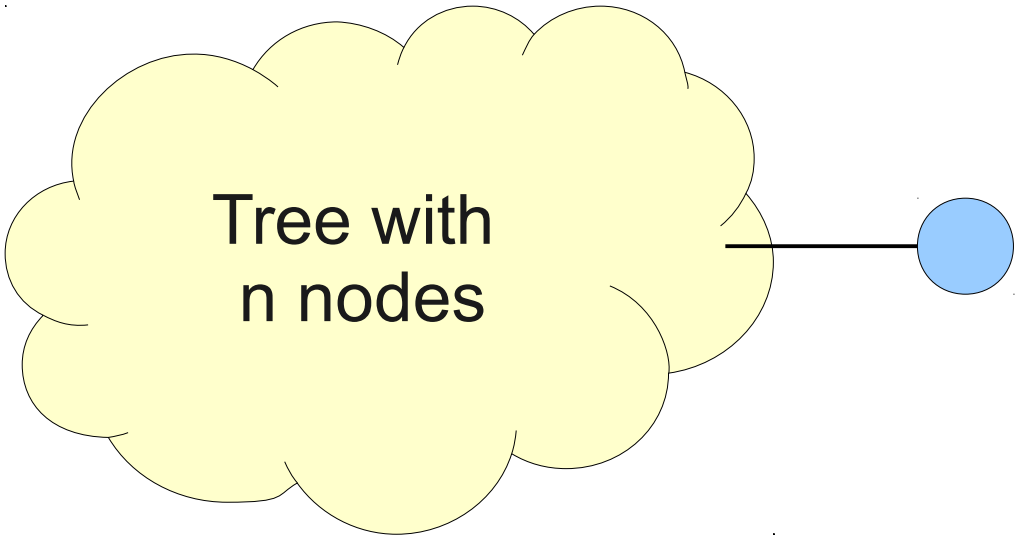
Each tree with five vertices seems to have four edges...

Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.

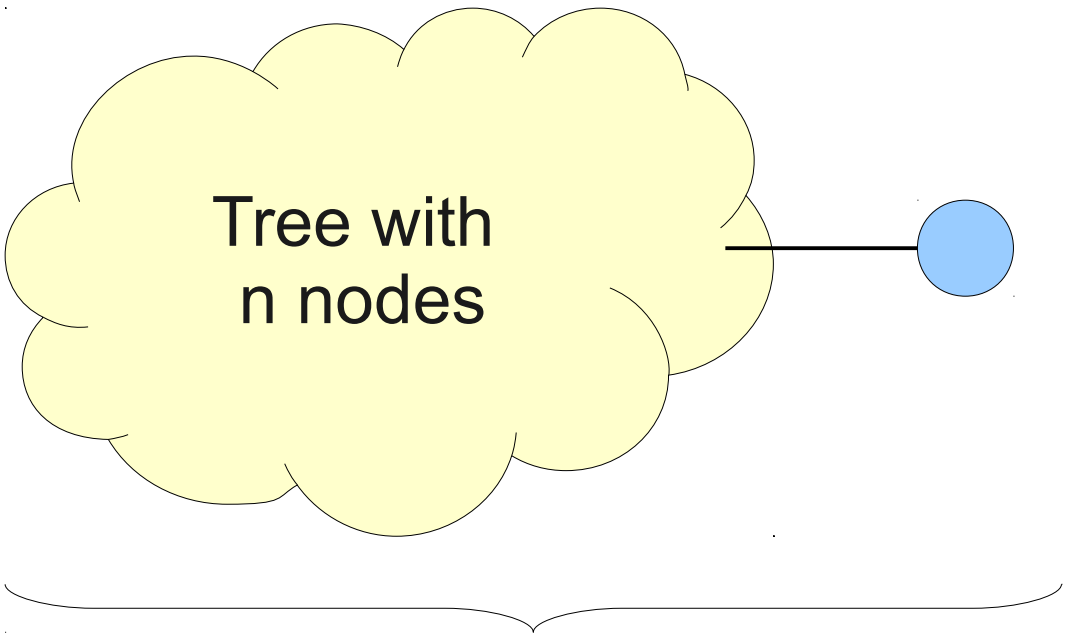
Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.



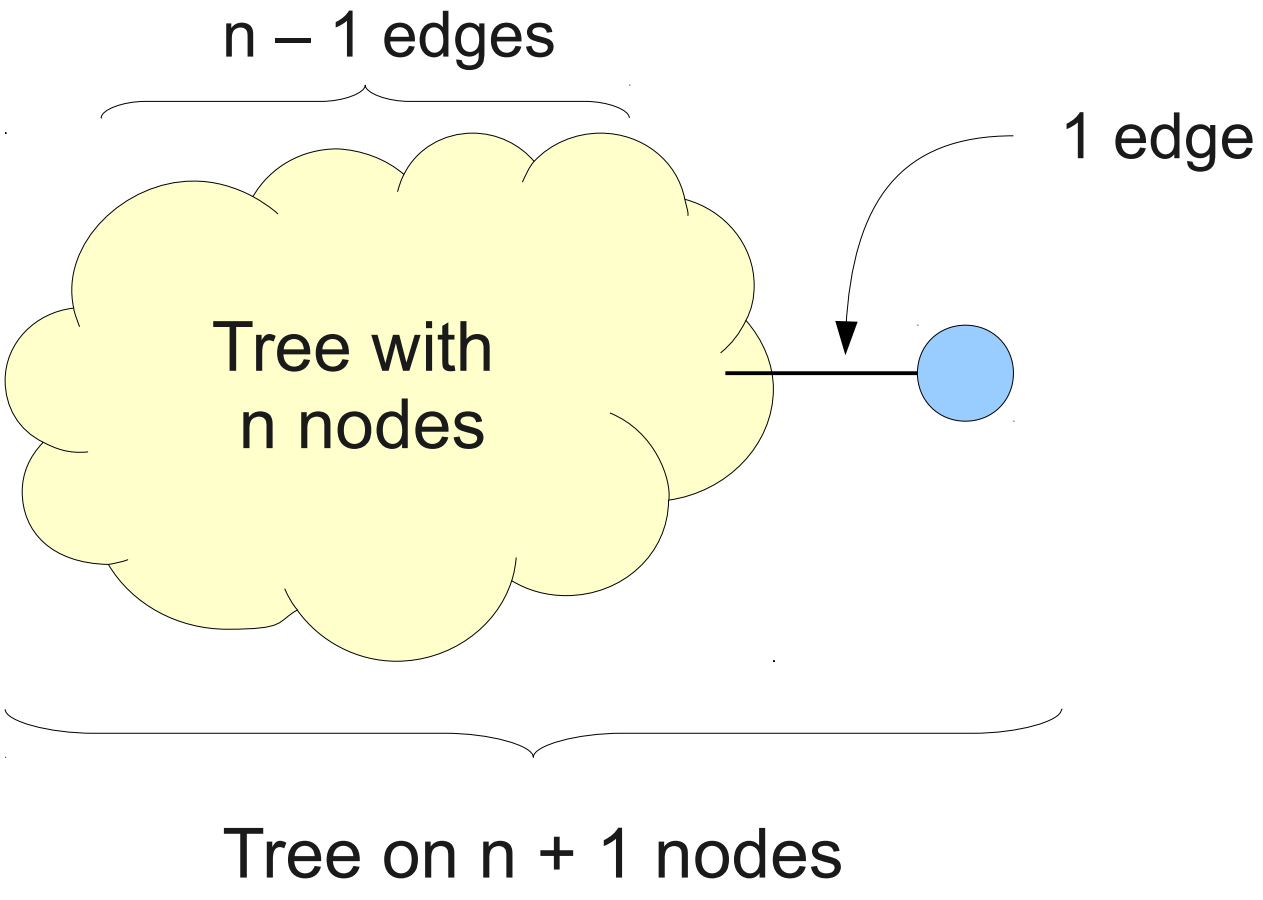
Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.



Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.



Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.

Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.

Proof: By induction.

Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.

Proof: By induction. As our base case, if n = 1, then there can be no edges in the graph, since the only possible edge would be from the single node to itself, contradicting the fact that trees do not contain cycles.

Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.

Proof: By induction. As our base case, if n = 1, then there can be no edges in the graph, since the only possible edge would be from the single node to itself, contradicting the fact that trees do not contain cycles.

For the inductive step, assume that for some n ≥ 1 the theorem is true. We show it holds for any tree with n + 1 nodes (that is, the tree has n edges).

Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.

Proof: By induction. As our base case, if n = 1, then there can be no edges in the graph, since the only possible edge would be from the single node to itself, contradicting the fact that trees do not contain cycles.

For the inductive step, assume that for some n ≥ 1 the theorem is true. We show it holds for any tree with n + 1 nodes (that is, the tree has n edges). If T is a tree with n + 1 nodes, then it must have some leaf node v.

Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.

Proof: By induction. As our base case, if n = 1, then there can be no edges in the graph, since the only possible edge would be from the single node to itself, contradicting the fact that trees do not contain cycles.

For the inductive step, assume that for some n ≥ 1 the theorem is true. We show it holds for any tree with n + 1 nodes (that is, the tree has n edges). If T is a tree with n + 1 nodes, then it must have some leaf node v. Let T' be the tree formed by removing v from T.

Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.

Proof: By induction. As our base case, if n = 1, then there can be no edges in the graph, since the only possible edge would be from the single node to itself, contradicting the fact that trees do not contain cycles.

For the inductive step, assume that for some n ≥ 1 the theorem is true. We show it holds for any tree with n + 1 nodes (that is, the tree has n edges). If T is a tree with n + 1 nodes, then it must have some leaf node v. Let T' be the tree formed by removing v from T. Then T is a tree with n nodes, so by the inductive hypothesis it contains n – 1 edges.

Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.

Proof: By induction. As our base case, if n = 1, then there can be no edges in the graph, since the only possible edge would be from the single node to itself, contradicting the fact that trees do not contain cycles.

For the inductive step, assume that for some n ≥ 1 the theorem is true. We show it holds for any tree with n + 1 nodes (that is, the tree has n edges). If T is a tree with n + 1 nodes, then it must have some leaf node v. Let T' be the tree formed by removing v from T. Then T is a tree with n nodes, so by the inductive hypothesis it contains n – 1 edges. Combined with the extra edge we deleted when we removed v from the graph, this gives a total of n edges, completing the induction.

Properties of Trees

Theorem: A nonempty tree with n vertices has n – 1 edges.

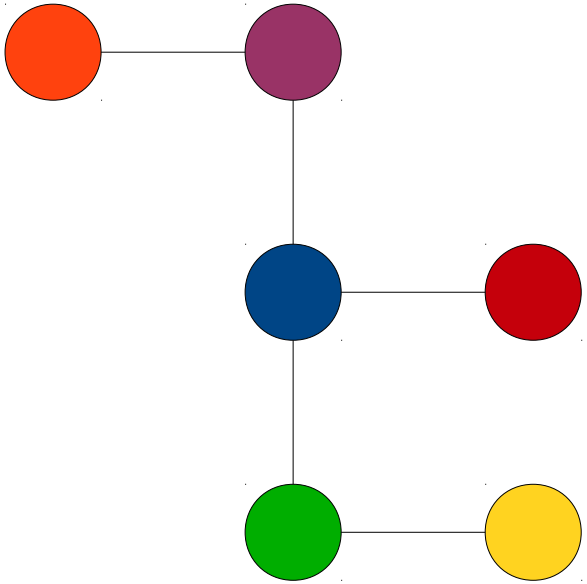
Proof: By induction. As our base case, if n = 1, then there can be no edges in the graph, since the only possible edge would be from the single node to itself, contradicting the fact that trees do not contain cycles.

For the inductive step, assume that for some n ≥ 1 the theorem is true. We show it holds for any tree with n + 1 nodes (that is, the tree has n edges). If T is a tree with n + 1 nodes, then it must have some leaf node v. Let T' be the tree formed by removing v from T. Then T' is a tree with n nodes, so by the inductive hypothesis it contains n – 1 edges. Combined with the extra edge we deleted when we removed v from the graph, this gives a total of n edges, completing the induction. ■

Special Trees

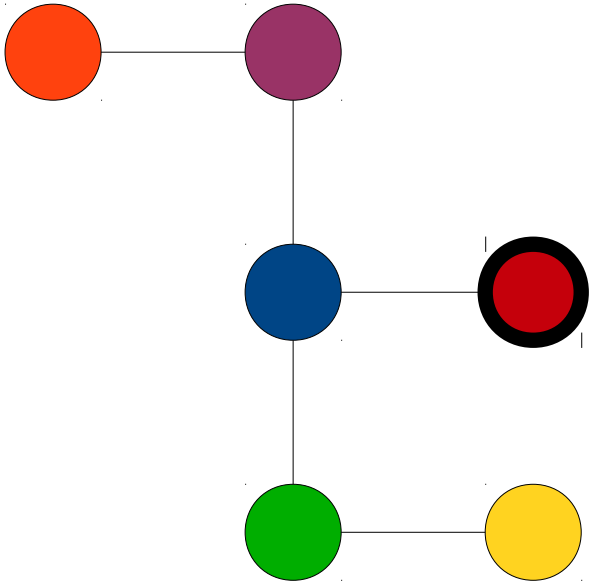
Rooted Trees

- Any tree can be thought of as a **rooted tree** with a special node called the **root**.
- The root is at the top of the tree with all elements below it (just like real trees!)



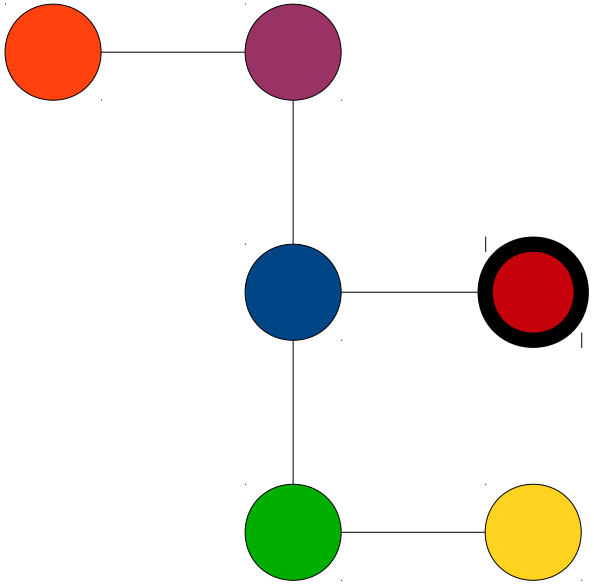
Rooted Trees

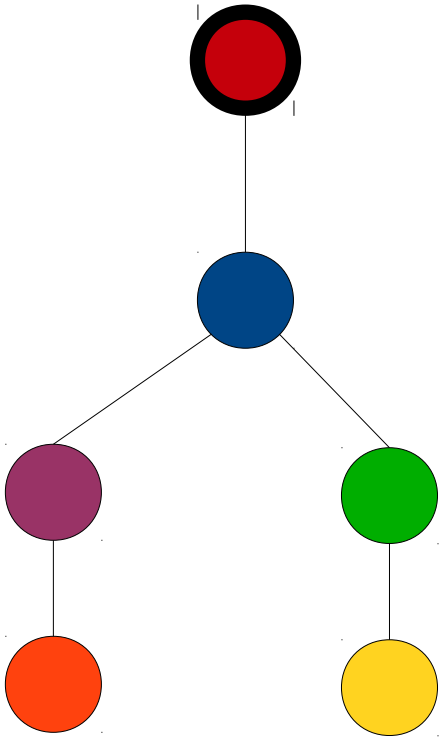
- Any tree can be thought of as a **rooted tree** with a special node called the **root**.
- The root is at the top of the tree with all elements below it (just like real trees!)



Rooted Trees

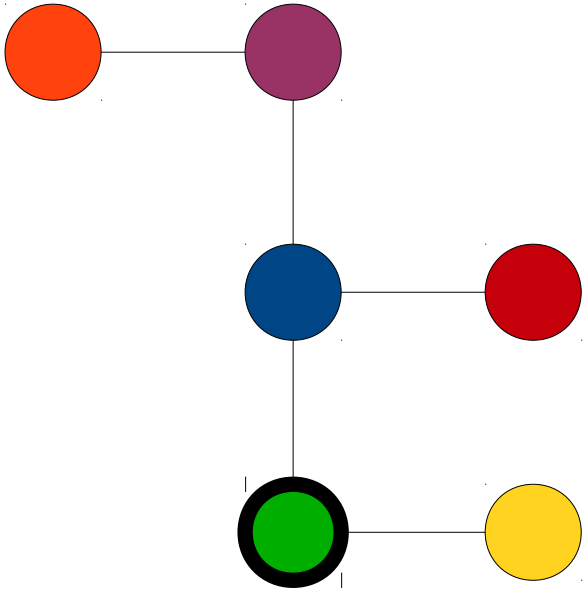
- Any tree can be thought of as a **rooted tree** with a special node called the **root**.
- The root is at the top of the tree with all elements below it (just like real trees!)

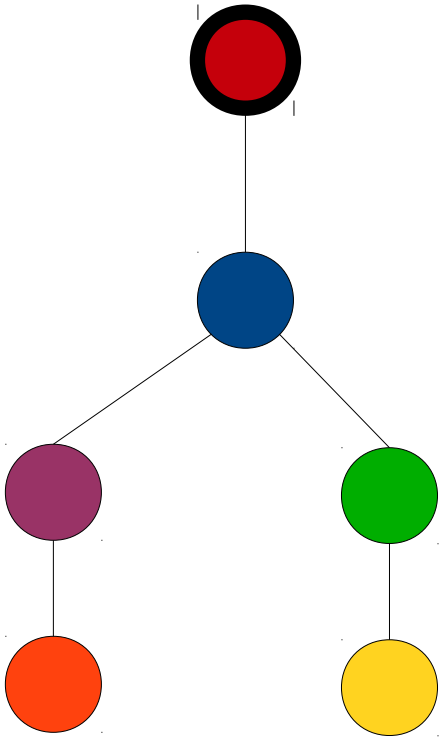




Rooted Trees

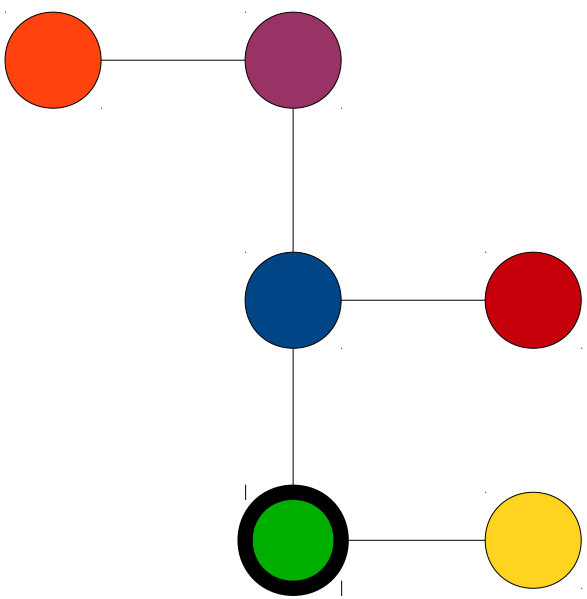
- Any tree can be thought of as a **rooted tree** with a special node called the **root**.
- The root is at the top of the tree with all elements below it (just like real trees!)

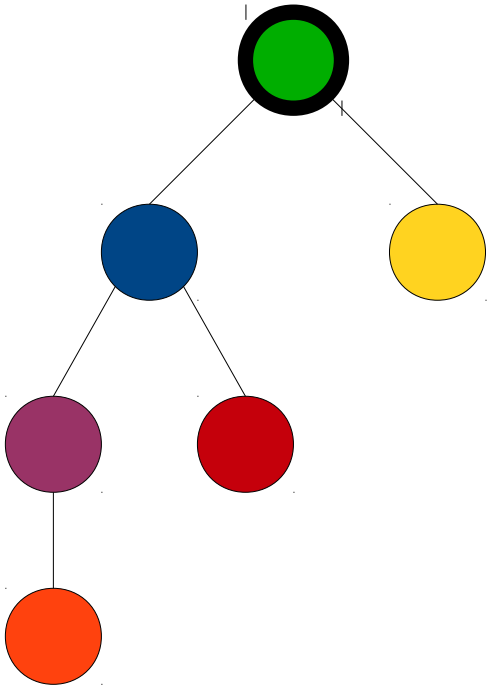




Rooted Trees

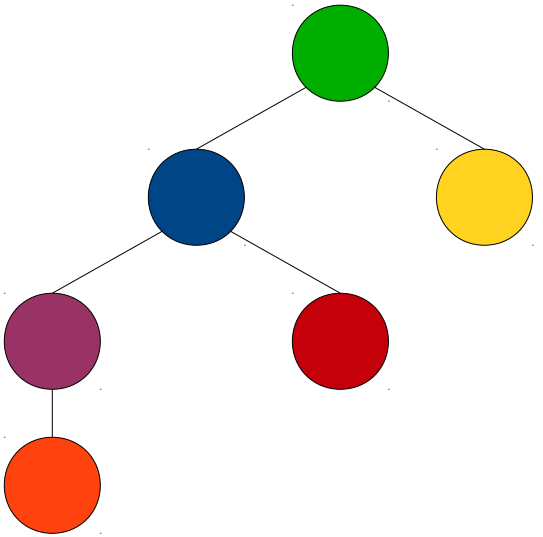
- Any tree can be thought of as a **rooted tree** with a special node called the **root**.
- The root is at the top of the tree with all elements below it (just like real trees!)





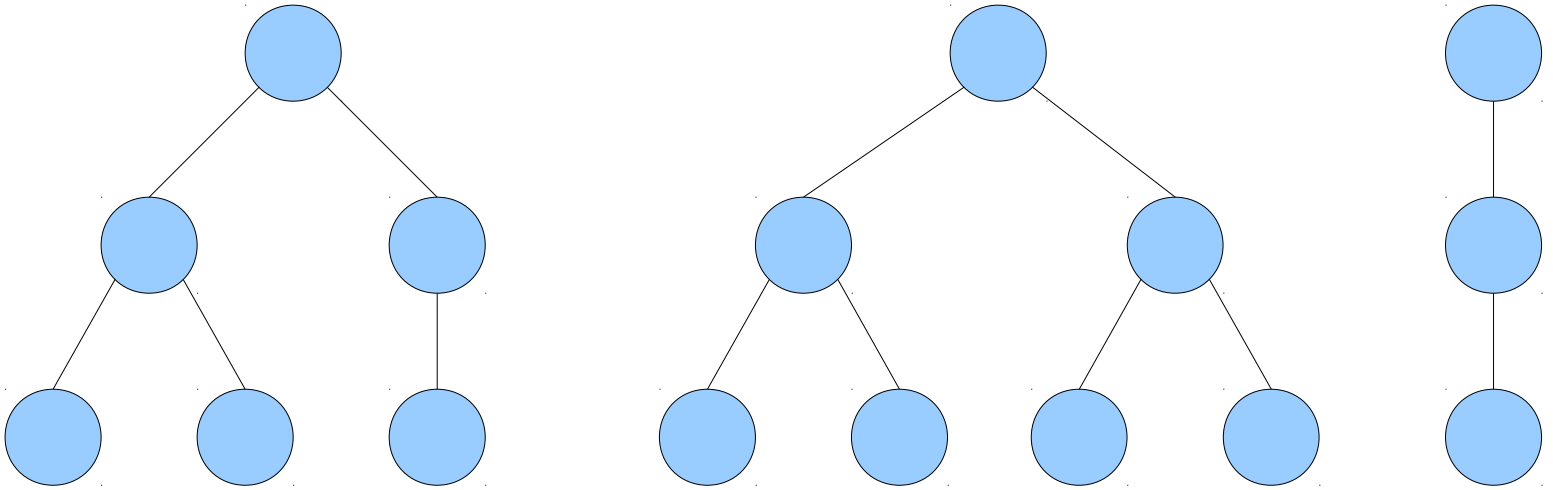
Rooted Trees

- In a rooted tree, a node u is an **ancestor** of v if the path from v to the root includes u. We say that v is a **descendant** of u.
- If {u, v} is an edge on the path from u to the root, we say that v is the **parent** of u and u is the **child** of v.
- The **height** of a tree is the length of the longest path from the root to a leaf node.



Binary Trees

- A **binary tree** is a rooted tree in which each node has at most two children.



Another Definition of Binary Trees

- The set of all binary trees is defined as follows:
  - A single node is a binary tree.
  - A node with one child that is a binary tree is a binary tree.
  - A node with two children that are both binary trees is a binary tree.
  - No other graphs are binary trees.

Another Definition of Binary Trees

- The set of all binary trees is defined as follows:
  - A single node is a binary tree.
  - A node with one child that is a binary tree is a binary tree.
  - A node with two children that are both binary trees is a binary tree.
  - No other graphs are binary trees.

This is a recursive definition!

Recursively-Defined Sets

- A **recursively-defined set** is a set S specified as a combination of
  - some initial elements of the set, and
  - some way of combining elements of the set together to form new elements, along with
  - a restriction saying nothing else is in the set.
- This last restriction is called a **limiting clause** or **extremal clause** and is often omitted.

Some Recursively-Defined Sets

- The set S where
  - $1 \in S$, and
  - If $n \in S$, then $2n \in S$
- The set S where
  - $\top \in S$.
  - $\bot \in S$
  - For any variable p, $p \in S$
  - If $\varphi \in S$, then $\neg\varphi \in S$ and $(\varphi) \in S$.
  - If $\varphi \in S$ and $\psi \in S$, then $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, and $\varphi \leftrightarrow \psi$ are all in S.

Proofs on Recursively-Defined Sets

- Given a recursively-defined set S, how do we prove properties about the elements of S?
- **The Principle of Structural Induction!**
- To show that P(s) is true for all s in S:
  - Show that P(s) is true for all elements initially known to be in S.
  - Show that for any $s = f(s_1, \ldots, s_n)$, where f is a rule defining elements of S from older elements of S, that if $P(s_1)$, $P(s_2)$, …, $P(s_n)$ all hold, then P(s) holds.
- Note the similarity to weak induction.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

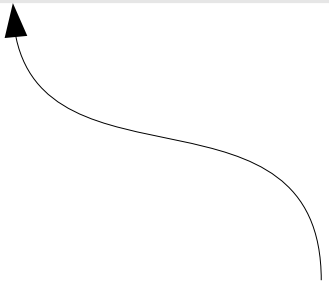
Proof: By structural induction.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction.



As with all other proof techniques, be sure to mention what proof technique you're using!

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

Note that we have multiple base cases here, since there are multiple elements that in the set by definition. Also note that neither of these base cases are P(0), since 0 is not defined to be in the set.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

For the inductive step, we have two cases to consider:

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

For the inductive step, we have two cases to consider:

Case 1: We need to show that if P(x) and P(y), then P(x + y).

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

For the inductive step, we have two cases to consider:

Case 1: We need to show that if P(x) and P(y), then P(x + y).

Note that the assumptions we make depend on what rule we're using. Since we need to show the property is true when taking the sum of x and y, we assume x and y have the property we want and show their sum does as well.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

For the inductive step, we have two cases to consider:

Case 1: We need to show that if P(x) and P(y), then P(x + y). If P(x) and P(y) are true, then $x = 4n$ for some $n \in \mathbb{Z}$ and $y = 4m$ for some $m \in \mathbb{Z}$.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

For the inductive step, we have two cases to consider:

Case 1: We need to show that if P(x) and P(y), then P(x + y). If P(x) and P(y) are true, then $x = 4n$ for some $n \in \mathbb{Z}$ and $y = 4m$ for some $m \in \mathbb{Z}$. Then $x + y = 4(n + m)$, so $x + y$ is divisible by four.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

For the inductive step, we have two cases to consider:

Case 1: We need to show that if P(x) and P(y), then P(x + y). If P(x) and P(y) are true, then $x = 4n$ for some $n \in \mathbb{Z}$ and $y = 4m$ for some $m \in \mathbb{Z}$. Then $x + y = 4(n + m)$, so $x + y$ is divisible by four. Thus P(x + y) holds.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

For the inductive step, we have two cases to consider:

Case 1: We need to show that if P(x) and P(y), then P(x + y). If P(x) and P(y) are true, then $x = 4n$ for some $n \in \mathbb{Z}$ and $y = 4m$ for some $m \in \mathbb{Z}$. Then $x + y = 4(n + m)$, so $x + y$ is divisible by four. Thus P(x + y) holds.

Case 2: We need to show that if P(x) and P(y), then P(xy).

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

For the inductive step, we have two cases to consider:

Case 1: We need to show that if P(x) and P(y), then P(x + y). If P(x) and P(y) are true, then $x = 4n$ for some $n \in \mathbb{Z}$ and $y = 4m$ for some $m \in \mathbb{Z}$. Then $x + y = 4(n + m)$, so x + y is divisible by four. Thus P(x + y) holds.

Case 2: We need to show that if P(x) and P(y), then P(xy).

We have to consider all rules, not just the first!

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

For the inductive step, we have two cases to consider:

Case 1: We need to show that if P(x) and P(y), then P(x + y). If P(x) and P(y) are true, then $x = 4n$ for some $n \in \mathbb{Z}$ and $y = 4m$ for some $m \in \mathbb{Z}$. Then $x + y = 4(n + m)$, so $x + y$ is divisible by four. Thus P(x + y) holds.

Case 2: We need to show that if P(x) and P(y), then P(xy). If P(x) and P(y) are true, then $x = 4n$ for some $n \in \mathbb{Z}$ and $y = 4m$ for some $m \in \mathbb{Z}$.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

For the inductive step, we have two cases to consider:

Case 1: We need to show that if P(x) and P(y), then P(x + y). If P(x) and P(y) are true, then $x = 4n$ for some $n \in \mathbb{Z}$ and $y = 4m$ for some $m \in \mathbb{Z}$. Then $x + y = 4(n + m)$, so x + y is divisible by four. Thus P(x + y) holds.

Case 2: We need to show that if P(x) and P(y), then P(xy). If P(x) and P(y) are true, then $x = 4n$ for some $n \in \mathbb{Z}$ and $y = 4m$ for some $m \in \mathbb{Z}$. Then $xy = 4(4mn)$, so xy is divisible by four, so P(xy) holds.

Theorem: Let S be defined recursively as follows:

$4 \in S$
$8 \in S$
If $x \in S$ and $y \in S$, then $x + y \in S$
If $x \in S$ and $y \in S$, then $xy \in S$

Then if $x \in S$, then is divisible by four.

Proof: By structural induction. Let P(s) be "s is divisible by four." We prove that P(s) is true for all $s \in S$. As our base cases, note that P(4) and P(8) are true, since each are divisible by four.

For the inductive step, we have two cases to consider:

Case 1: We need to show that if P(x) and P(y), then P(x + y). If P(x) and P(y) are true, then $x = 4n$ for some $n \in \mathbb{Z}$ and $y = 4m$ for some $m \in \mathbb{Z}$. Then $x + y = 4(n + m)$, so x + y is divisible by four. Thus P(x + y) holds.

Case 2: We need to show that if P(x) and P(y), then P(xy). If P(x) and P(y) are true, then $x = 4n$ for some $n \in \mathbb{Z}$ and $y = 4m$ for some $m \in \mathbb{Z}$. Then $xy = 4(4mn)$, so xy is divisible by four, so P(xy) holds. ■

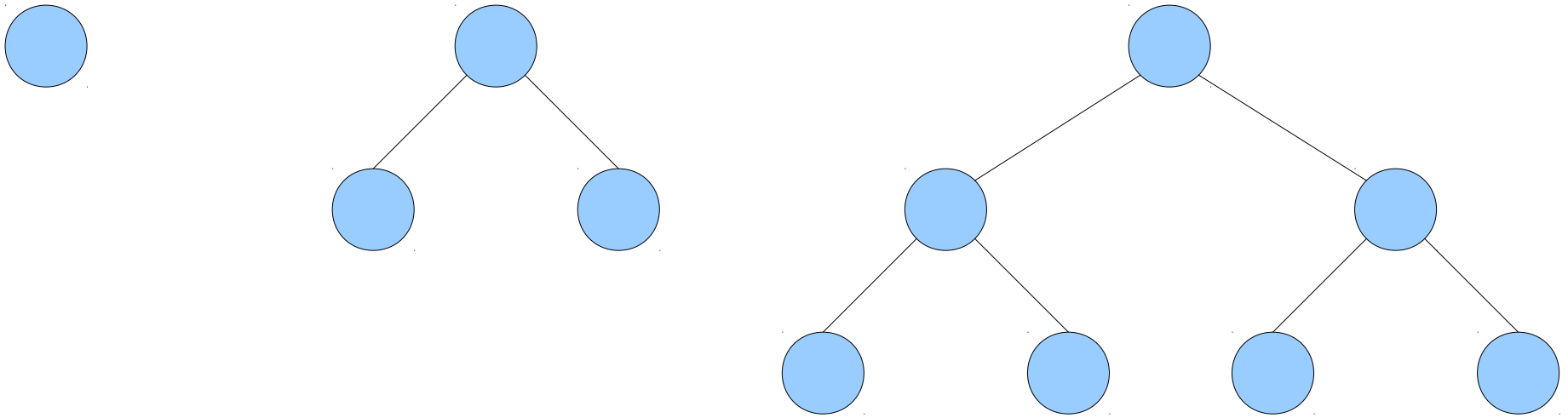
A Recursive Definition of Binary Trees

- The set of all binary trees is defined as follows:
  - A single node is a binary tree.
  - A node with one child that is a binary tree is a binary tree.
  - A node with two children that are both binary trees is a binary tree.
  - No other graphs are binary trees.

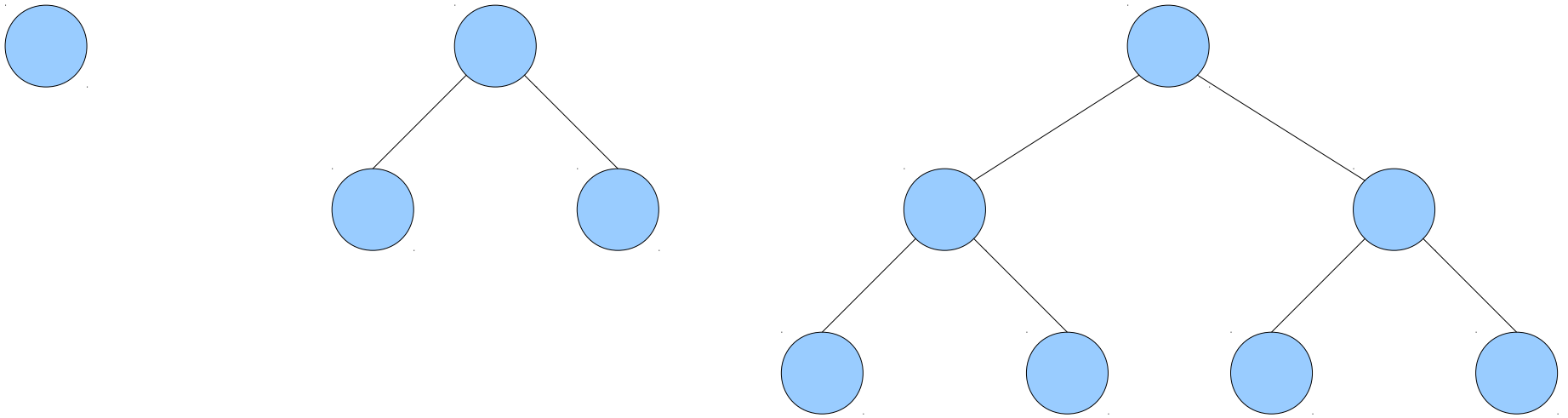
A Recursive Definition of Binary Trees

- The set of all binary trees is defined as follows:
  - A single node is a binary tree.
  - A node with one child that is a binary tree is a binary tree.
  - A node with two children that are both binary trees is a binary tree.
  - No other graphs are binary trees.
- **What is the maximum number of nodes in a binary tree of height h?**

Counting Nodes



Counting Nodes

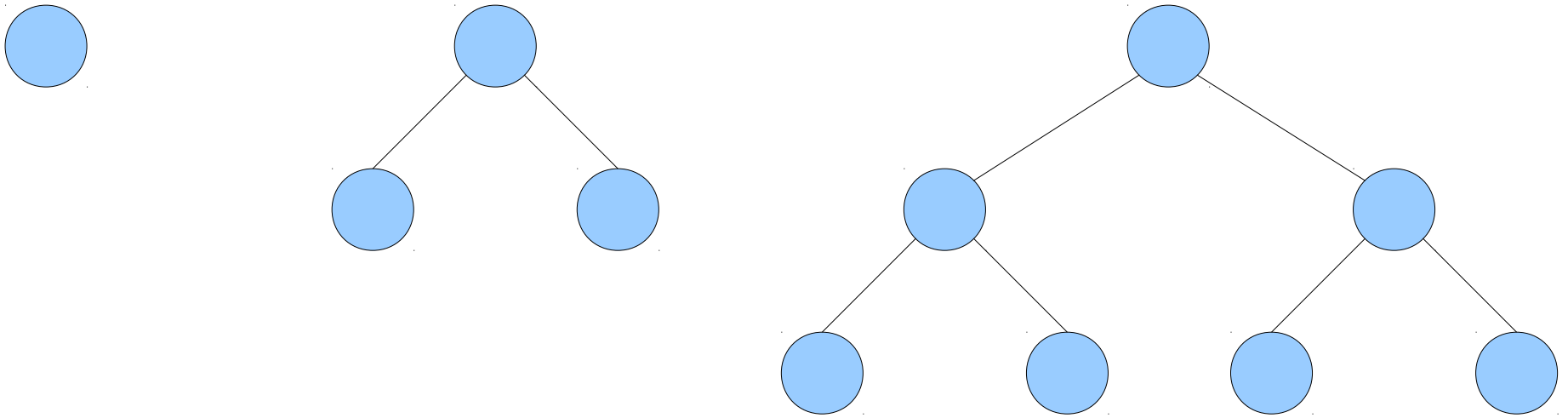


Height: 0

Height: 1

Height: 2

Counting Nodes



Height: 0

Nodes: 1

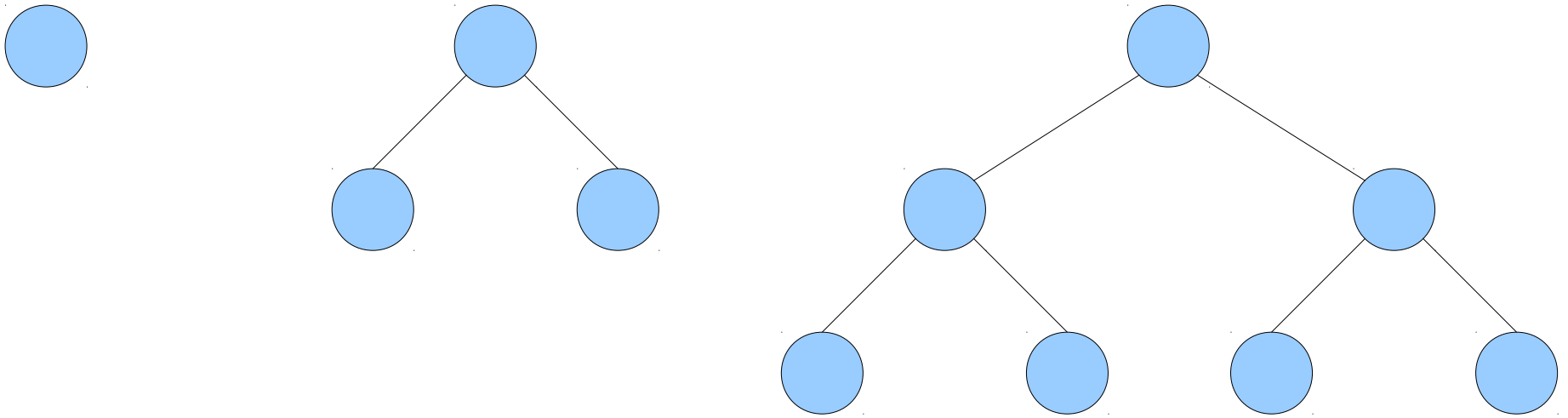
Height: 1

Nodes: 3

Height: 2

Nodes: 7

Counting Nodes



Height: 0

Nodes: $2^1 - 1$

Height: 1

Nodes: $2^2 - 1$

Height: 2

Nodes: $2^3 - 1$

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1. By the inductive hypothesis, the tree of height h has up to $2^{h+1} - 1$ nodes in it.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1. By the inductive hypothesis, the tree of height h has up to $2^{h+1} - 1$ nodes in it. The new tree thus has at most $2^{h+1} - 1 + 1 = 2^{h+1}$ nodes.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1. By the inductive hypothesis, the tree of height h has up to $2^{h+1} - 1$ nodes in it. The new tree thus has at most $2^{h+1} - 1 + 1 = 2^{h+1}$ nodes. Since $2^{h+1} \geq 1$, $2^{h+1} - 1 \geq 0$, so the number of nodes is at most $2^{h+1} \leq 2^{h+1} + 2^{h+1} - 1 = 2(2^{h+1}) - 1 = 2^{h+2} - 1$, so the claim holds for the new tree.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1. By the inductive hypothesis, the tree of height h has up to $2^{h+1} - 1$ nodes in it. The new tree thus has at most $2^{h+1} - 1 + 1 = 2^{h+1}$ nodes. Since $2^{h+1} \geq 1$, $2^{h+1} - 1 \geq 0$, so the number of nodes is at most $2^{h+1} \leq 2^{h+1} + 2^{h+1} - 1 = 2(2^{h+1}) - 1 = 2^{h+2} - 1$, so the claim holds for the new tree.

Case 2: Binary trees with a root node with two children.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1. By the inductive hypothesis, the tree of height h has up to $2^{h+1} - 1$ nodes in it. The new tree thus has at most $2^{h+1} - 1 + 1 = 2^{h+1}$ nodes. Since $2^{h+1} \geq 1$, $2^{h+1} - 1 \geq 0$, so the number of nodes is at most $2^{h+1} \leq 2^{h+1} + 2^{h+1} - 1 = 2(2^{h+1}) - 1 = 2^{h+2} - 1$, so the claim holds for the new tree.

Case 2: Binary trees with a root node with two children. Suppose the two children have heights of h and k, respectively.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1. By the inductive hypothesis, the tree of height h has up to $2^{h+1} - 1$ nodes in it. The new tree thus has at most $2^{h+1} - 1 + 1 = 2^{h+1}$ nodes. Since $2^{h+1} \geq 1$, $2^{h+1} - 1 \geq 0$, so the number of nodes is at most $2^{h+1} \leq 2^{h+1} + 2^{h+1} - 1 = 2(2^{h+1}) - 1 = 2^{h+2} - 1$, so the claim holds for the new tree.

Case 2: Binary trees with a root node with two children. Suppose the two children have heights of h and k, respectively. Assume, without loss of generality, that $h \geq k$.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1. By the inductive hypothesis, the tree of height h has up to $2^{h+1} - 1$ nodes in it. The new tree thus has at most $2^{h+1} - 1 + 1 = 2^{h+1}$ nodes. Since $2^{h+1} \geq 1$, $2^{h+1} - 1 \geq 0$, so the number of nodes is at most $2^{h+1} \leq 2^{h+1} + 2^{h+1} - 1 = 2(2^{h+1}) - 1 = 2^{h+2} - 1$, so the claim holds for the new tree.

Case 2: Binary trees with a root node with two children. Suppose the two children have heights of h and k, respectively. Assume, without loss of generality, that $h \geq k$.

Here, "assume without loss of generality" means that the cases where $h \leq k$ and $k \geq h$ are symmetric and neither is special. We can always rename the variables if $k \geq h$.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1. By the inductive hypothesis, the tree of height h has up to $2^{h+1} - 1$ nodes in it. The new tree thus has at most $2^{h+1} - 1 + 1 = 2^{h+1}$ nodes. Since $2^{h+1} \geq 1$, $2^{h+1} - 1 \geq 0$, so the number of nodes is at most $2^{h+1} \leq 2^{h+1} + 2^{h+1} - 1 = 2(2^{h+1}) - 1 = 2^{h+2} - 1$, so the claim holds for the new tree.

Case 2: Binary trees with a root node with two children. Suppose the two children have heights of h and k, respectively. Assume, without loss of generality, that $h \geq k$. Then the new tree has height h + 1, and the subtrees have at most $2^{h+1} - 1$ and $2^{k+1} - 1$ nodes.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1. By the inductive hypothesis, the tree of height h has up to $2^{h+1} - 1$ nodes in it. The new tree thus has at most $2^{h+1} - 1 + 1 = 2^{h+1}$ nodes. Since $2^{h+1} \geq 1$, $2^{h+1} - 1 \geq 0$, so the number of nodes is at most $2^{h+1} \leq 2^{h+1} + 2^{h+1} - 1 = 2(2^{h+1}) - 1 = 2^{h+2} - 1$, so the claim holds for the new tree.

Case 2: Binary trees with a root node with two children. Suppose the two children have heights of h and k, respectively. Assume, without loss of generality, that $h \geq k$. Then the new tree has height h + 1, and the subtrees have at most $2^{h+1} - 1$ and $2^{k+1} - 1$ nodes. Then the total number of nodes is at most $2^{h+1} - 1 + 2^{k+1} - 1 + 1 = 2^{h+1} + 2^{k+1} - 1$.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1. By the inductive hypothesis, the tree of height h has up to $2^{h+1} - 1$ nodes in it. The new tree thus has at most $2^{h+1} - 1 + 1 = 2^{h+1}$ nodes. Since $2^{h+1} \geq 1$, $2^{h+1} - 1 \geq 0$, so the number of nodes is at most $2^{h+1} \leq 2^{h+1} + 2^{h+1} - 1 = 2(2^{h+1}) - 1 = 2^{h+2} - 1$, so the claim holds for the new tree.

Case 2: Binary trees with a root node with two children. Suppose the two children have heights of h and k, respectively. Assume, without loss of generality, that $h \geq k$. Then the new tree has height h + 1, and the subtrees have at most $2^{h+1} - 1$ and $2^{k+1} - 1$ nodes. Then the total number of nodes is at most $2^{h+1} - 1 + 2^{k+1} - 1 + 1 = 2^{h+1} + 2^{k+1} - 1$. Since $k \leq h$, $2^{h+1} + 2^{k+1} - 1 \leq 2^{h+1} + 2^{h+1} - 1 = 2^{h+2} - 1$ nodes, the claim holds for the new tree.

Theorem: A binary tree of height h has at most $2^{h+1} - 1$ nodes.

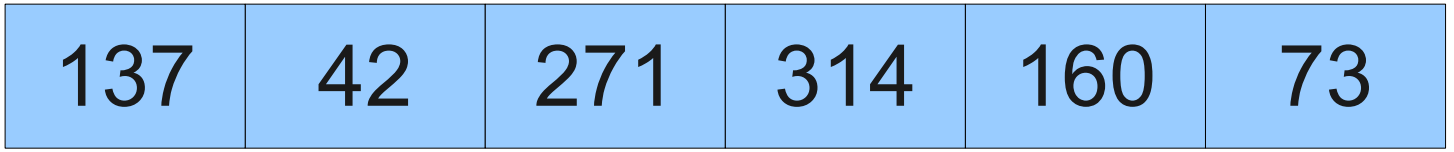
Proof: By structural induction. As our base case, a single node is a binary tree of height 0, and it contains $1 = 2^1 - 1$ nodes. For the inductive step, we consider two types of binary trees:

Case 1: Binary trees with a root node with exactly one child. Then if that one child has height h, the new tree has height h + 1. By the inductive hypothesis, the tree of height h has up to $2^{h+1} - 1$ nodes in it. The new tree thus has at most $2^{h+1} - 1 + 1 = 2^{h+1}$ nodes. Since $2^{h+1} \geq 1$, $2^{h+1} - 1 \geq 0$, so the number of nodes is at most $2^{h+1} \leq 2^{h+1} + 2^{h+1} - 1 = 2(2^{h+1}) - 1 = 2^{h+2} - 1$, so the claim holds for the new tree.

Case 2: Binary trees with a root node with two children. Suppose the two children have heights of h and k, respectively. Assume, without loss of generality, that $h \geq k$. Then the new tree has height h + 1, and the subtrees have at most $2^{h+1} - 1$ and $2^{k+1} - 1$ nodes. Then the total number of nodes is at most $2^{h+1} - 1 + 2^{k+1} - 1 + 1 = 2^{h+1} + 2^{k+1} - 1$. Since $k \leq h$, $2^{h+1} + 2^{k+1} - 1 \leq 2^{h+1} + 2^{h+1} - 1 = 2^{h+2} - 1$ nodes, the claim holds for the new tree. ■

Application: **The Sorting Lower Bound**

Sorting Algorithms



Sorting Algorithms

137	42	271	314	160	73

42	73	137	160	271	314

The Sorting Problem

- Given a list $x_1, x_2, \ldots, x_n$ of values drawn from a totally ordered set S, output a **permutation** of the sequence that is in ascending order.
  - A **permutation** is a rearrangement of the same elements that doesn't discard or introduce anything new.
- There are **many** sorting algorithms:

Bubble Sort	Merge Sort	Counting Sort	Tournament Sort
Insertion Sort	Heap Sort	Radix Sort	Cartesian Tree Sort
Selection Sort	Quicksort	Bucket Sort	Natural Merge Sort
Gnome Sort	Smoothsort	vEB-Sort	Strand Sort
Stoogesort	Treesort	Bead Sort	Pancake Sort
Bogosort	Flashsort	Cycle Sort	Introsort

Is there an inherent limitation in how fast a list of n elements can be sorted?

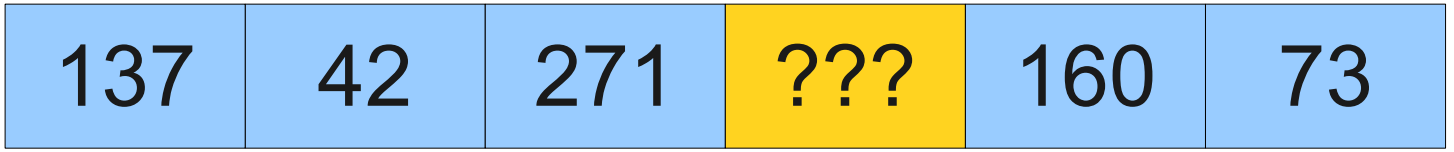
The Adversarial Approach

- One way to find lower bounds on algorithms is to consider an **adversary** who tries to break the algorithm.
- The adversary can watch what the algorithm does, then tries to find some input where the algorithm outputs the wrong answer.

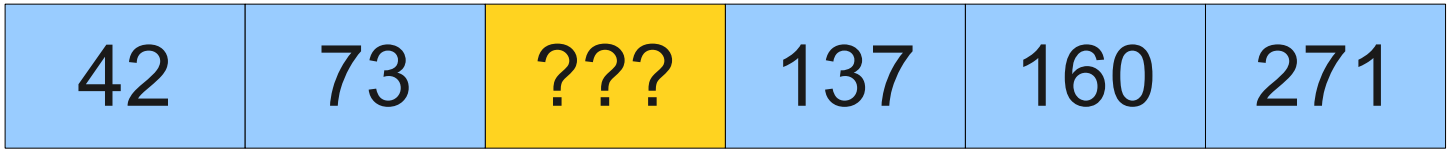
A Weak Lower Bound

- **Theorem:** Every sorting algorithm must do at least as much work as is necessary to look at each element of the input.
  - Depending on how the input is represented and what model of computation is used, this can differ greatly.
- **Intuition**: If some algorithm doesn't look at each element of the input sequence, then some element wasn't inspected. How can the algorithm know where to put it?

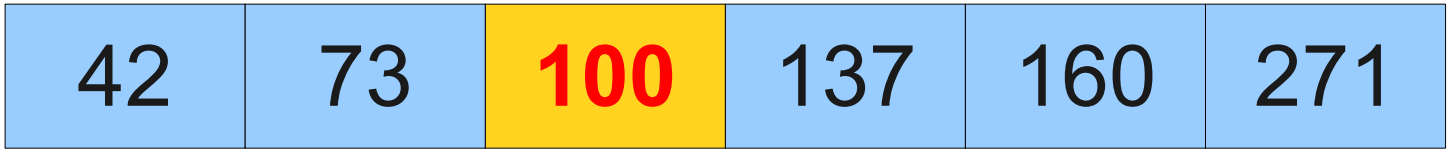
A Weak Lower Bound: n work



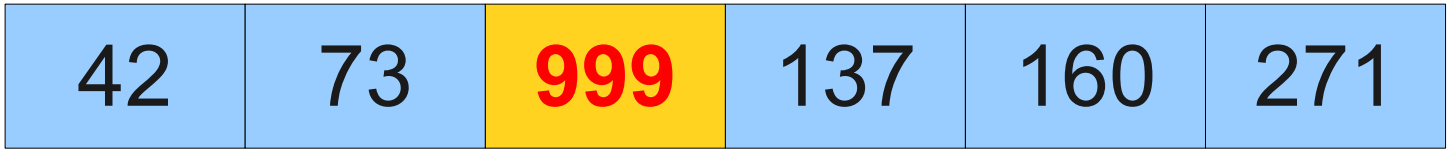
A Weak Lower Bound: n work



A Weak Lower Bound: n work



A Weak Lower Bound: n work



The Adversarial Approach

- To show that some problem cannot be solved without doing f(n) work:
  - Assume, for the sake of contradiction, that the problem can be solved doing at most f(n) work.
  - Show that because the algorithm cannot do f(n) work, there must be two inputs that it cannot distinguish.
  - Since the algorithm cannot tell them apart, the algorithm must treat both inputs identically.
  - Thus an adversary could run the algorithm on one input, then change it in a way that the algorithm can't notice but causes the result to be incorrect.
  - Conclude, by contradiction, that the algorithm must use at least f(n) work.

Comparison Sorting

- A sorting algorithm is called a **comparison sort** if the only information it can use about its elements is whether they are greater than or less than one another.
- For example, comparison sorts cannot
  - Inspect the bits of the objects to sort to make decisions about them.
  - Store the elements in hash tables (can't assume a hash function exists)
- Advantage of comparison sorts: Given a totally ordered set $(S, \leq_S)$, comparison sorts can sort just by using $\leq_S$ with no other information.

An Adversarial Analysis

- Suppose we want to determine the minimum amount of work required to sort a list of n elements with a comparison sort.
- **Idea**: Use the adversarial approach. Show that if the algorithm doesn't do “enough work,” then there must be two different input lists it can't distinguish.
- If the algorithm can't distinguish the two input lists, then it must sort one of them wrong (an adversary could tamper with the sequence to produce the wrong answer).

The Key Insight

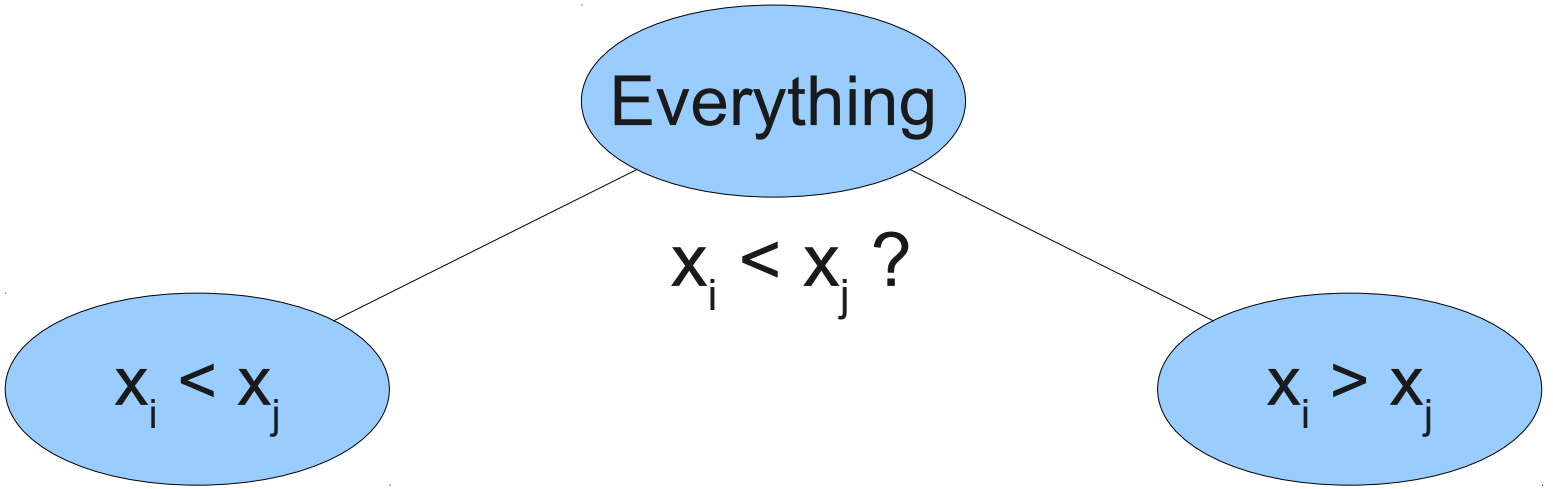
- Before doing **any** comparisons, the algorithm cannot distinguish any of the possible input sequences to the problem.
- After doing the first comparison (say, comparing x_i and x_j), the algorithm can split the set of all input sequences into two groups – a group where the comparison said that $x_i < x_j$ and a group where the comparison said that $x_i > x_j$.
- More generally, at each point, each comparison splits some group of inputs into at most two groups based on the result of the comparison.

Making Comparisons

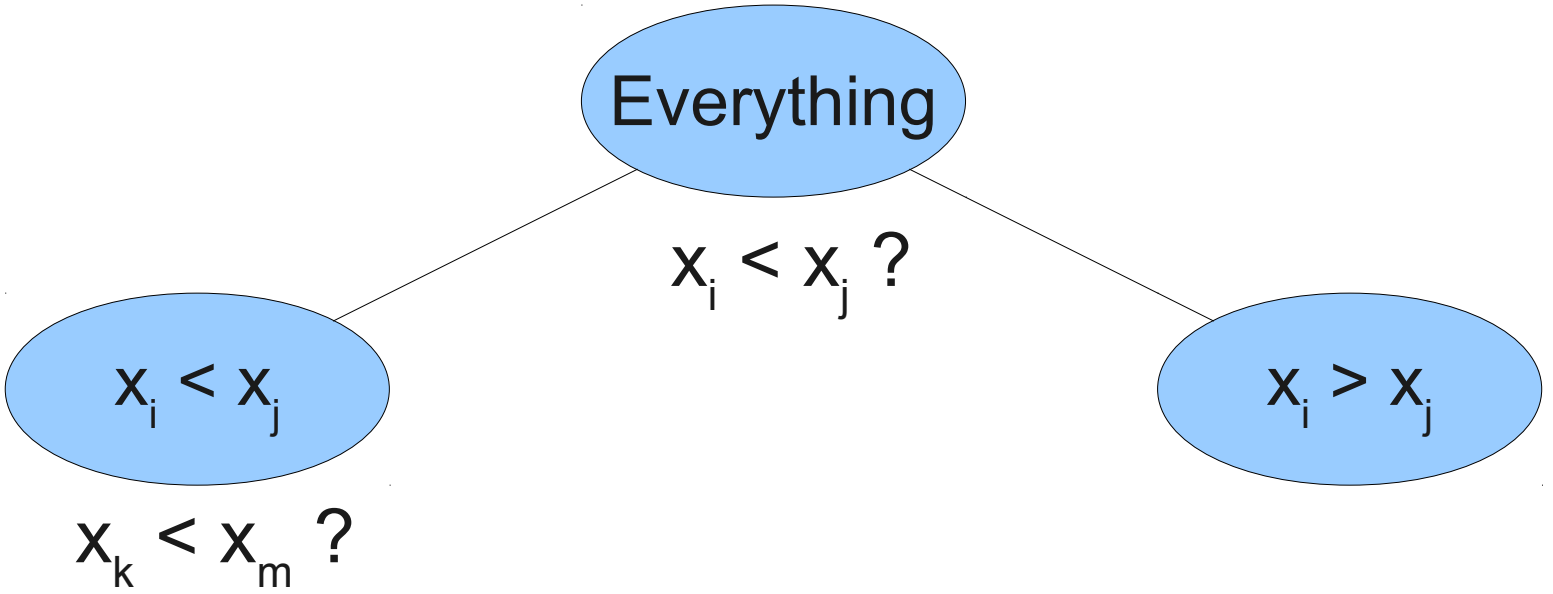
Making Comparisons

$x_i < x_j$?

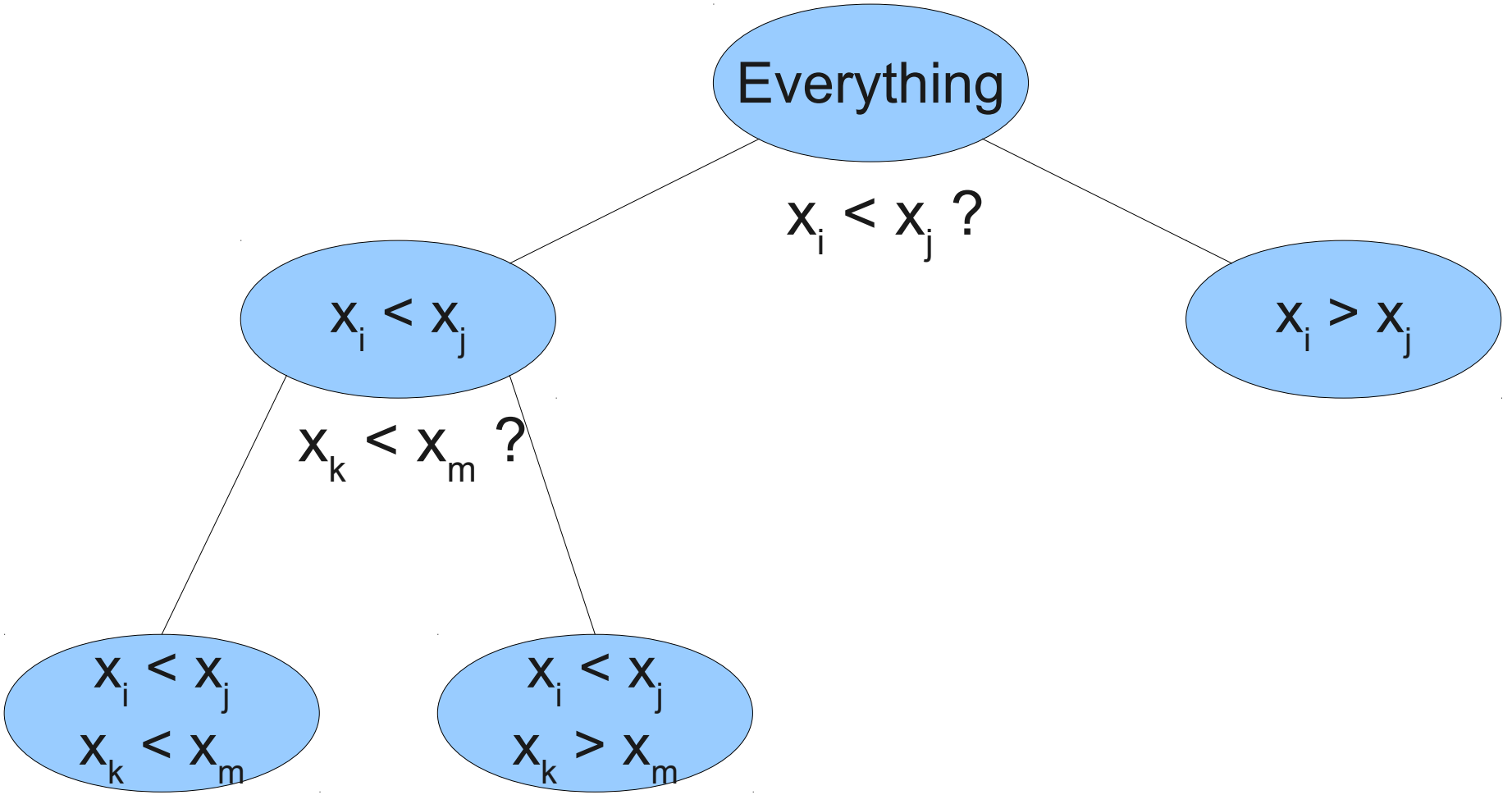
Making Comparisons



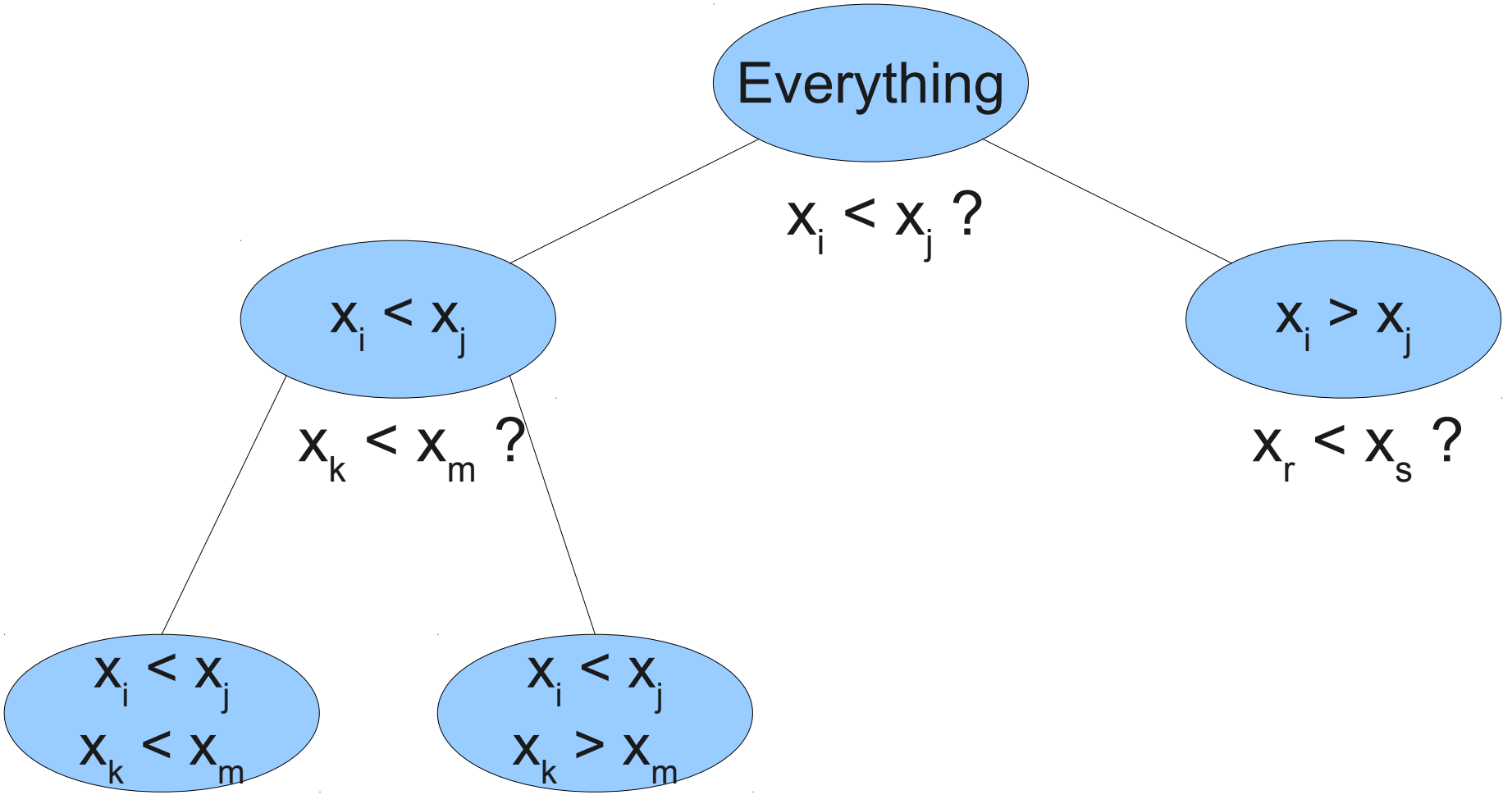
Making Comparisons



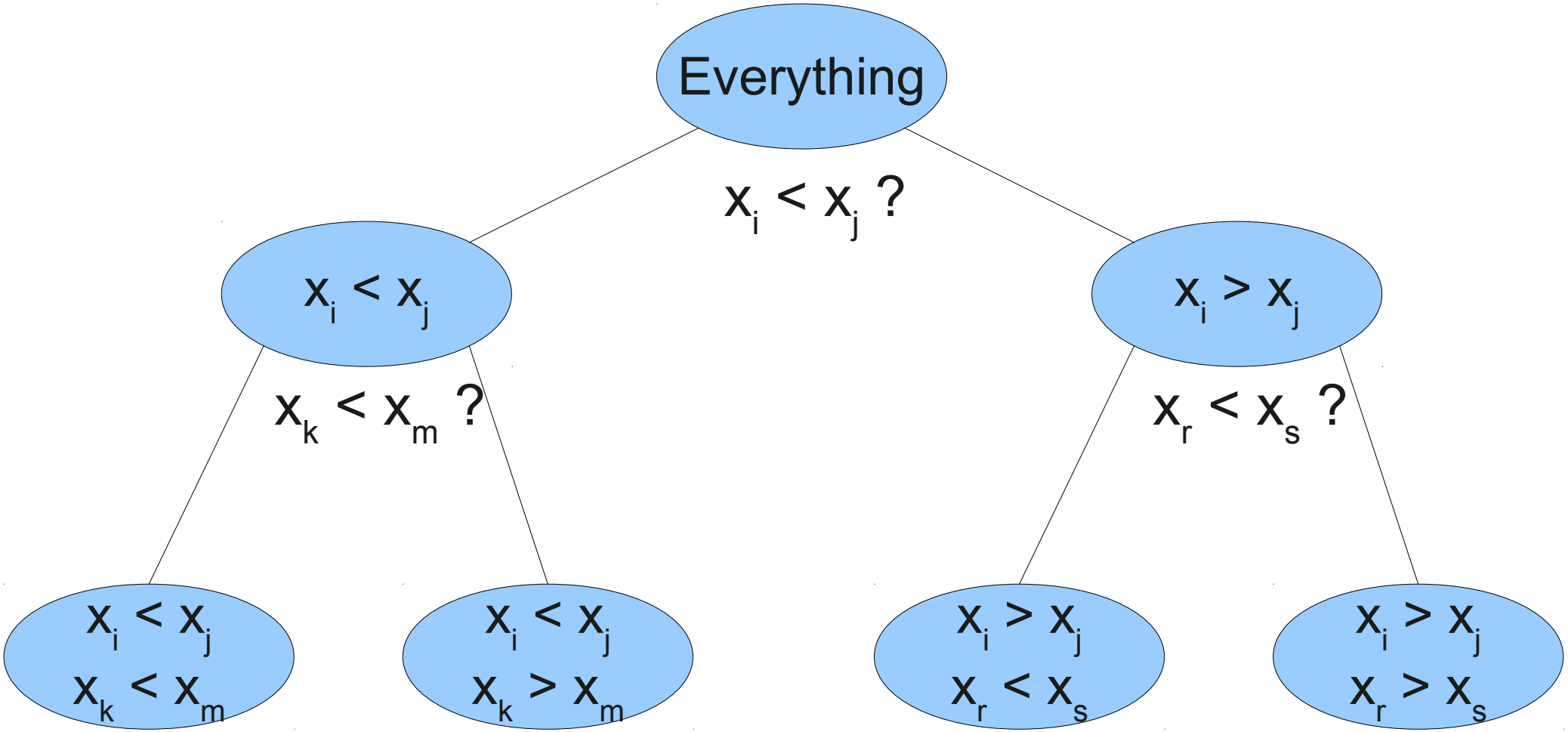
Making Comparisons



Making Comparisons



Making Comparisons



The Decision Tree

- Let the **state** of the algorithm be the information the algorithm has at any point in time about the input.
- Each time a comparison is made, the current state is split into at most two states, based on the result of the comparision.
- We can build a **binary tree** out of the results of these comparisons.
- Each path of length h in the tree corresponds to making h different comparisons.
- The height of the tree therefore corresponds to the maximum number of comparisons the algorithm ever makes.

A Fun Trick

- In order for all inputs to be distinguishable from one another, we need to have at least one state for each possible input.
- In the best case, if we make f(n) comparisons, then the binary tree will have height f(n).
- Thus the binary tree will contain $2^{f(n) + 1} - 1$ nodes.
- If $2^{f(n) + 1} - 1$ is less than the number of possible inputs to the problem, the sorting algorithm cannot possibly output the right answer each time.
  - Reason: Two different inputs must be treated the same way, and they can't both have the same sorted order.

Inputs to Sorting

- If we want to sort n distinct elements, then the actual values of those elements doesn't matter.
  - Since all we can do is compare elements, we can never actually tell what elements were sorted.
- What does matter is the **relative ordering** of those elements.
- Given n elements, there are n! possible relative orderings of those elements, one for each permutation.

The Sorting Lower Bound

- If we have n elements to sort, there are n! different permutations we might get as inputs.
- We need to have at least $2^{f(n)+1} - 1$ different states we get into.
- So $2^{f(n)+1} - 1 \geq n!$
- So $f(n) \geq \lg(n! + 1) - 1$
- **We cannot sort without making at least $\lg(n! + 1) - 1$ comparisons**.

Stirling's Approximation

- As n gets large, lg n! becomes approximately n lg n – n.
- **Theorem**: Comparison sorts cannot perform fewer than **n lg n – n** comparisons without being incorrect at least once.

So What?

- This result can be used to show how close various sorting algorithms come to the theoretical limit.
  - An analysis of **quicksort** can be used to show that it is always within 40% of the optimal number of comparisons.
- This result motivates a search for **non-comparison sorts** that perform better by using operations other than comparisons.
  - **Radix sort** can often outperform quicksort when run on on array of integers.

An Important Milestone

Recap: **Discrete Mathematics**

- The past four weeks have focused exclusively on discrete mathematics:
  - Relations
  - Graphs
  - Trees
  - Induction
  - Functions
  - Logic
- These are the building blocks we will use throughout the rest of the quarter.
- There are the building blocks you will use throughout the rest of your CS career.

Next Up: **Computability Theory**

- It's time to switch gears and address the limits of what can be computed.
- We'll explore
  - What is the formal definition of a computer?
  - What might computers look like with various resource constraints?
  - What problems can be solved by these computers?
  - What problems **can't** be solved by these computers?
- **Get ready to explore the boundaries of what computers could ever be made to do.**