Lecture 4: Trees, Rooted Trees, Graph Isomorphism

Dr. George Mertzios

george.mertzios@durham.ac.uk

Reminder from last lecture

- A tree is a connected acyclic graph.
- Each tree contains at least one leaf, i.e. a vertex of degree 1.
- A tree on n vertices has exactly $n-1$ edges.
- Conversely, a connected graph with n vertices and $n-1$ edges is a tree.
- A tree contains a unique path between any two vertices in it.
- A rooted tree has one distinguished vertex, the root, and all edges are directed away from the root.

Contents for today's lecture

- Exercises involving trees;
- Examples of proof techniques;
- Properties of rooted trees.

The number of leaves in a tree

The number of leaves in a tree

What is the minimum number of leaves in a tree on at least 2 vertices?

The number of leaves in a tree

What is the minimum number of leaves in a tree on at least 2 vertices?

Lemma

A tree with at least 2 vertices, n_3 of which have degree at least 3, has at least $n_3 + 2$ leaves.

The number of leaves in a tree

What is the minimum number of leaves in a tree on at least 2 vertices?

Lemma

A tree with at least 2 vertices, n_3 of which have degree at least 3, has at least $n_3 + 2$ leaves.

Proof: Let T be a tree on $n \geq 2$ vertices. We use induction on n.

The number of leaves in a tree

What is the minimum number of leaves in a tree on at least 2 vertices?

Lemma

A tree with at least 2 vertices, n_3 of which have degree at least 3, has at least $n_3 + 2$ leaves.

Proof: Let T be a tree on $n \geq 2$ vertices. We use induction on n.

- Let $\ell(T)$ denote the number of leaves in T, and $n_3(T)$ denote the number of vertices of degree at least 3 in T.

The number of leaves in a tree

What is the minimum number of leaves in a tree on at least 2 vertices?

Lemma

A tree with at least 2 vertices, n_3 of which have degree at least 3, has at least $n_3 + 2$ leaves.

Proof: Let T be a tree on $n \geq 2$ vertices. We use induction on n.

- Let $\ell(T)$ denote the number of leaves in T, and $n_3(T)$ denote the number of vertices of degree at least 3 in T.
- Induction Base : If $n = 2$, then $n_3 = 0$ and T has 2 leaves.

The number of leaves in a tree

What is the minimum number of leaves in a tree on at least 2 vertices?

Lemma

A tree with at least 2 vertices, n_3 of which have degree at least 3, has at least $n_3 + 2$ leaves.

Proof: Let T be a tree on $n \geq 2$ vertices. We use induction on n.

- Let $\ell(T)$ denote the number of leaves in T, and $n_3(T)$ denote the number of vertices of degree at least 3 in T.
- Induction Base : If $n = 2$, then $n_3 = 0$ and T has 2 leaves.
- Step: Now suppose that every tree on $< n$ vertices has at least $n_3 + 2$ leaves (induction hypothesis), and consider a tree T on $n \geq 3$ vertices.

The number of leaves in a tree

What is the minimum number of leaves in a tree on at least 2 vertices?

Lemma

A tree with at least 2 vertices, n_3 of which have degree at least 3, has at least $n_3 + 2$ leaves.

Proof: Let T be a tree on $n \geq 2$ vertices. We use induction on n.

- Let $\ell(T)$ denote the number of leaves in T, and $n_3(T)$ denote the number of vertices of degree at least 3 in T.
- Induction Base : If $n = 2$, then $n_3 = 0$ and T has 2 leaves.
- Step: Now suppose that every tree on $< n$ vertices has at least $n_3 + 2$ leaves (induction hypothesis), and consider a tree T on $n \geq 3$ vertices.
- Since T is a tree on at least 3 vertices, T has a leaf u.

The number of leaves in a tree

What is the minimum number of leaves in a tree on at least 2 vertices?

Lemma

A tree with at least 2 vertices, n_3 of which have degree at least 3, has at least $n_3 + 2$ leaves.

Proof: Let T be a tree on $n \geq 2$ vertices. We use induction on n.

- Let $\ell(T)$ denote the number of leaves in T, and $n_3(T)$ denote the number of vertices of degree at least 3 in T.
- Induction Base : If $n = 2$, then $n_3 = 0$ and T has 2 leaves.
- Step: Now suppose that every tree on $< n$ vertices has at least $n_3 + 2$ leaves (induction hypothesis), and consider a tree T on $n \geq 3$ vertices.
- Since T is a tree on at least 3 vertices, T has a leaf u.
- Then $T' = T - u$ is a tree on $n - 1$ vertices. By the induction hypothesis, we have $\ell(T') \geq n_3(T') + 2$.

The number of leaves in a tree

What is the minimum number of leaves in a tree on at least 2 vertices?

Lemma

A tree with at least 2 vertices, n_3 of which have degree at least 3, has at least $n_3 + 2$ leaves.

Proof: Let T be a tree on $n \geq 2$ vertices. We use induction on n.

- Let $\ell(T)$ denote the number of leaves in T, and $n_3(T)$ denote the number of vertices of degree at least 3 in T.
- Induction Base : If $n = 2$, then $n_3 = 0$ and T has 2 leaves.
- Step: Now suppose that every tree on $< n$ vertices has at least $n_3 + 2$ leaves (induction hypothesis), and consider a tree T on $n \geq 3$ vertices.
- Since T is a tree on at least 3 vertices, T has a leaf u.
- Then $T' = T - u$ is a tree on $n - 1$ vertices. By the induction hypothesis, we have $\ell(T') \geq n_3(T') + 2$.
- The rest of the proof is on the next slide.

Proof continued

- We have: a leaf u in T, a tree $T' = T - u$, $\ell(T') \geq n_3(T') + 2$.
- Let v be the (unique) neighbour of u in T.
- T is connected and has at least 3 vertices, so v has at least 2 neighbors in T.

Proof continued

- We have: a leaf u in T, a tree $T' = T - u$, $\ell(T') \geq n_3(T') + 2$.
- Let v be the (unique) neighbour of u in T.
- T is connected and has at least 3 vertices, so v has at least 2 neighbors in T.
- The rest of the proof is by case analysis.

Proof continued

- We have: a leaf u in T, a tree $T' = T - u$, $\ell(T') \geq n_3(T') + 2$.
- Let v be the (unique) neighbour of u in T.
- T is connected and has at least 3 vertices, so v has at least 2 neighbors in T.
- The rest of the proof is by case analysis.

1. Suppose that v has exactly two neighbors in T.
   - Then $n_3(T') = n_3(T)$ and $\ell(T') = \ell(T)$.
   - Hence, $\ell(T) = \ell(T') \geq n_3(T') + 2 = n_3(T) + 2$.

Proof continued

- We have: a leaf u in T, a tree $T' = T - u$, $\ell(T') \geq n_3(T') + 2$.
- Let v be the (unique) neighbour of u in T.
- T is connected and has at least 3 vertices, so v has at least 2 neighbors in T.
- The rest of the proof is by case analysis.

1. Suppose that v has exactly two neighbors in T.
   - Then $n_3(T') = n_3(T)$ and $\ell(T') = \ell(T)$.
   - Hence, $\ell(T) = \ell(T') \geq n_3(T') + 2 = n_3(T) + 2$.
2. Suppose that v has exactly three neighbors in T.
   - Then $n_3(T') = n_3(T) - 1$ and $\ell(T') = \ell(T) - 1$.
   - Hence, $\ell(T) = \ell(T') + 1 \geq n_3(T') + 2 + 1 = n_3(T) - 1 + 2 + 1 = n_3(T) + 2$.

Proof continued

- We have: a leaf u in T, a tree $T' = T - u$, $\ell(T') \geq n_3(T') + 2$.
- Let v be the (unique) neighbour of u in T.
- T is connected and has at least 3 vertices, so v has at least 2 neighbors in T.
- The rest of the proof is by case analysis.

1. Suppose that v has exactly two neighbors in T.
   - Then $n_3(T') = n_3(T)$ and $\ell(T') = \ell(T)$.
   - Hence, $\ell(T) = \ell(T') \geq n_3(T') + 2 = n_3(T) + 2$.
2. Suppose that v has exactly three neighbors in T.
   - Then $n_3(T') = n_3(T) - 1$ and $\ell(T') = \ell(T) - 1$.
   - Hence, $\ell(T) = \ell(T') + 1 \geq n_3(T') + 2 + 1 = n_3(T) - 1 + 2 + 1 = n_3(T) + 2$.
3. Suppose that v has at least four neighbors in T.
   - Then, $n_3(T) = n_3(T')$ and $\ell(T') = \ell(T) - 1$.
   - Hence, $\ell(T) = \ell(T') + 1 \geq n_3(T') + 2 + 1 = n_3(T) + 2 + 1 \geq n_3(T) + 2$.

Proof continued

- We have: a leaf u in T, a tree $T' = T - u$, $\ell(T') \geq n_3(T') + 2$.
- Let v be the (unique) neighbour of u in T.
- T is connected and has at least 3 vertices, so v has at least 2 neighbors in T.
- The rest of the proof is by case analysis.

1. Suppose that v has exactly two neighbors in T.
   - Then $n_3(T') = n_3(T)$ and $\ell(T') = \ell(T)$.
   - Hence, $\ell(T) = \ell(T') \geq n_3(T') + 2 = n_3(T) + 2$.
2. Suppose that v has exactly three neighbors in T.
   - Then $n_3(T') = n_3(T) - 1$ and $\ell(T') = \ell(T) - 1$.
   - Hence, $\ell(T) = \ell(T') + 1 \geq n_3(T') + 2 + 1 = n_3(T) - 1 + 2 + 1 = n_3(T) + 2$.
3. Suppose that v has at least four neighbors in T.
   - Then, $n_3(T) = n_3(T')$ and $\ell(T') = \ell(T) - 1$.
   - Hence, $\ell(T) = \ell(T') + 1 \geq n_3(T') + 2 + 1 = n_3(T) + 2 + 1 \geq n_3(T) + 2$.

This finishes the proof.

Note that induction on n_3 is also possible.

Every tree is a bipartite graph

Theorem

Every tree is a bipartite graph.

Every tree is a bipartite graph

Theorem

Every tree is a bipartite graph.

Proof.

We give a direct proof. We can use the known result on unique paths in a tree T to define a bipartition of its vertex set $V(T)$.

Every tree is a bipartite graph

Theorem

Every tree is a bipartite graph.

Proof.

We give a direct proof. We can use the known result on unique paths in a tree T to define a bipartition of its vertex set $V(T)$.

- Choose any vertex v and put this vertex in the set V_1.
- For every vertex $u \neq v$, there is a unique path from v to u in T, consider the length of this path.
- If the length is odd, put u in V_2; otherwise put u in V_1.

Every tree is a bipartite graph

Theorem

Every tree is a bipartite graph.

Proof.

We give a direct proof. We can use the known result on unique paths in a tree T to define a bipartition of its vertex set $V(T)$.

- Choose any vertex v and put this vertex in the set V_1.
- For every vertex $u \neq v$, there is a unique path from v to u in T, consider the length of this path.
- If the length is odd, put u in V_2; otherwise put u in V_1.
- We have to show that this is a valid bipartition.
- V_1 and V_2 are disjoint and together make up $V(T)$. (Why?)
- Every edge has end vertices in both V_1 and V_2. (Why?)
- This completes the proof.

□

Graph isomorphism

Definition

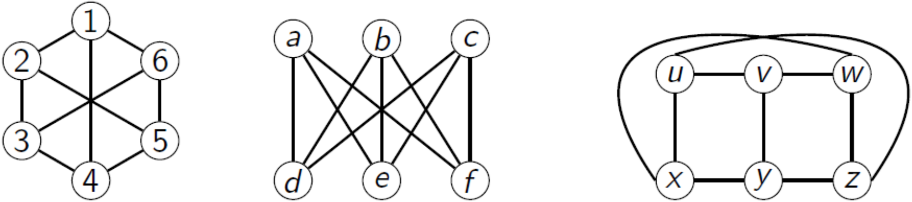
Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijective function $f : V \to V'$ such that for every $u, v \in V$ we have: $uv \in E$ if and only if $f(u)f(v) \in E'$. Then we write: $G \cong G'$.

Graph isomorphism

Definition

Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijective function $f : V \to V'$ such that for every $u, v \in V$ we have: $uv \in E$ if and only if $f(u)f(v) \in E'$. Then we write: $G \cong G'$.

Example: which of these graphs are isomorphic?

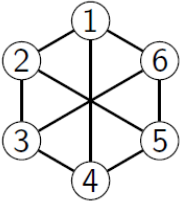


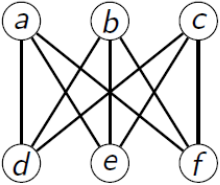
Graph isomorphism

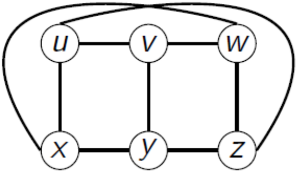
Definition

Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijective function $f : V \to V'$ such that for every $u, v \in V$ we have: $uv \in E$ if and only if $f(u)f(v) \in E'$. Then we write: $G \cong G'$.

Example: which of these graphs are isomorphic?







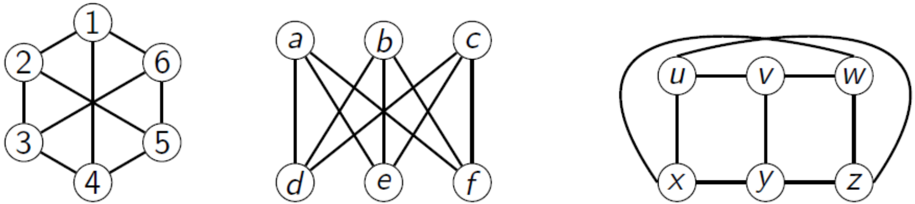
bijective function f for the first and second graph:
$1 \mapsto a,\ 2 \mapsto d,\ 3 \mapsto b,\ 4 \mapsto e,\ 5 \mapsto c,\ 6 \mapsto f$

Graph isomorphism

Definition

Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijective function $f : V \to V'$ such that for every $u, v \in V$ we have: $uv \in E$ if and only if $f(u)f(v) \in E'$. Then we write: $G \cong G'$.

Example: which of these graphs are isomorphic?



bijective function f for the first and second graph:
$1 \mapsto a,\ 2 \mapsto d,\ 3 \mapsto b,\ 4 \mapsto e,\ 5 \mapsto c,\ 6 \mapsto f$
bijective function f for the second and third graph:
$a \mapsto x,\ b \mapsto v,\ c \mapsto z,\ d \mapsto u,\ e \mapsto w,\ f \mapsto y$

Graph isomorphism

Definition

Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijective function $f : V \to V'$ such that for every $u, v \in V$ we have: $uv \in E$ if and only if $f(u)f(v) \in E'$. Then we write: $G \cong G'$.

In other words, $G \cong G'$ when:

- there exists a correspondence between vertices of G and G' which maintains the "neighborhood property"

Graph isomorphism

Definition

Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijective function $f : V \to V'$ such that for every $u, v \in V$ we have: $uv \in E$ if and only if $f(u)f(v) \in E'$. Then we write: $G \cong G'$.

In other words, $G \cong G'$ when:

- there exists a correspondence between vertices of G and G' which maintains the "neighborhood property", or equivalently
- there is a "renaming" of G' such that it coincides with G.

Graph isomorphism

Definition

Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijective function $f : V \to V'$ such that for every $u, v \in V$ we have: $uv \in E$ if and only if $f(u)f(v) \in E'$. Then we write: $G \cong G'$.

In other words, $G \cong G'$ when:

- there exists a correspondence between vertices of G and G' which maintains the "neighborhood property", or equivalently
- there is a "renaming" of G' such that it coincides with G.

Isomorphism is an equivalence relationship:

- $G \cong G$, i.e. every graph is isomorphic to itself, and
- if $G_1 \cong G_2$ and $G_2 \cong G_3$, then also $G_1 \cong G_3$.

Graph isomorphism

Definition

Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijective function $f : V \to V'$ such that for every $u, v \in V$ we have: $uv \in E$ if and only if $f(u)f(v) \in E'$. Then we write: $G \cong G'$.

In other words, $G \cong G'$ when:

- there exists a correspondence between vertices of G and G' which maintains the "neighborhood property", or equivalently
- there is a "renaming" of G' such that it coincides with G.

Isomorphism is an equivalence relationship:

- $G \cong G$, i.e. every graph is isomorphic to itself, and
- if $G_1 \cong G_2$ and $G_2 \cong G_3$, then also $G_1 \cong G_3$.

Can we quickly decide whether G and G' are isomorphic?

Graph isomorphism

Definition

Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijective function $f : V \to V'$ such that for every $u, v \in V$ we have: $uv \in E$ if and only if $f(u)f(v) \in E'$. Then we write: $G \cong G'$.

In other words, $G \cong G'$ when:

- there exists a correspondence between vertices of G and G' which maintains the "neighborhood property", or equivalently
- there is a "renaming" of G' such that it coincides with G.

Isomorphism is an equivalence relationship:

- $G \cong G$, i.e. every graph is isomorphic to itself, and
- if $G_1 \cong G_2$ and $G_2 \cong G_3$, then also $G_1 \cong G_3$.

Can we quickly decide whether G and G' are isomorphic?

- one of the few most tantalising and tricky questions in Computer Science
- presumably not an "easy" problem (i.e. polynomial-time) and not a "hard" problem (i.e. NP-complete), but "somewhere in the middle"

Graph isomorphism

If G and G' are isomorphic, they shared all their structural characteristics, e.g.:

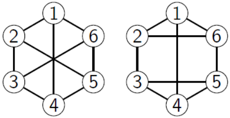
- number of vertices and edges, degree sequence, (strong) connectivity

Graph isomorphism

If G and G' are isomorphic, they shared all their structural characteristics, e.g.:

- number of vertices and edges, degree sequence, (strong) connectivity
- Euler circuit, Hamiltonian Circuit, chromatic number, size of largest independent set, ...

But none of these alone determines isomorphism:

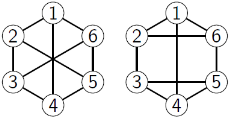


Graph isomorphism

If G and G' are isomorphic, they shared all their structural characteristics, e.g.:

- number of vertices and edges, degree sequence, (strong) connectivity
- Euler circuit, Hamiltonian Circuit, chromatic number, size of largest independent set, ...

But none of these alone determines isomorphism:



Not isomorphic:

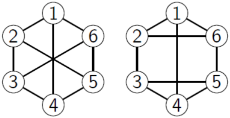
- although same number of vertices / edges, degree sequence

Graph isomorphism

If G and G' are isomorphic, they shared all their structural characteristics, e.g.:

- number of vertices and edges, degree sequence, (strong) connectivity
- Euler circuit, Hamiltonian Circuit, chromatic number, size of largest independent set, ...

But none of these alone determines isomorphism:



Not isomorphic:

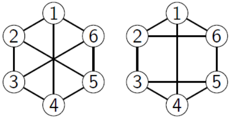
- although same number of vertices / edges, degree sequence
- Euler circuit, Hamiltonian Circuit:

Graph isomorphism

If G and G' are isomorphic, they shared all their structural characteristics, e.g.:

- number of vertices and edges, degree sequence, (strong) connectivity
- Euler circuit, Hamiltonian Circuit, chromatic number, size of largest independent set, ...

But none of these alone determines isomorphism:



Not isomorphic:

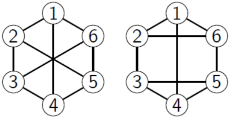
- although same number of vertices / edges, degree sequence
- Euler circuit, Hamiltonian Circuit: yes

Graph isomorphism

If G and G' are isomorphic, they shared all their structural characteristics, e.g.:

- number of vertices and edges, degree sequence, (strong) connectivity
- Euler circuit, Hamiltonian Circuit, chromatic number, size of largest independent set, ...

But none of these alone determines isomorphism:



Not isomorphic:

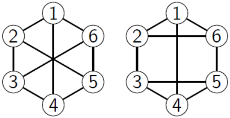
- although same number of vertices / edges, degree sequence
- Euler circuit, Hamiltonian Circuit: yes
- chromatic number, size of largest independent set:

Graph isomorphism

If G and G' are isomorphic, they shared all their structural characteristics, e.g.:

- number of vertices and edges, degree sequence, (strong) connectivity
- Euler circuit, Hamiltonian Circuit, chromatic number, size of largest independent set, ...

But none of these alone determines isomorphism:



Not isomorphic:

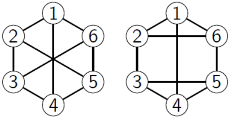
- although same number of vertices / edges, degree sequence
- Euler circuit, Hamiltonian Circuit: yes
- chromatic number, size of largest independent set: no

Graph isomorphism

If G and G' are isomorphic, they shared all their structural characteristics, e.g.:

- number of vertices and edges, degree sequence, (strong) connectivity
- Euler circuit, Hamiltonian Circuit, chromatic number, size of largest independent set, ...

But none of these alone determines isomorphism:



Not isomorphic:

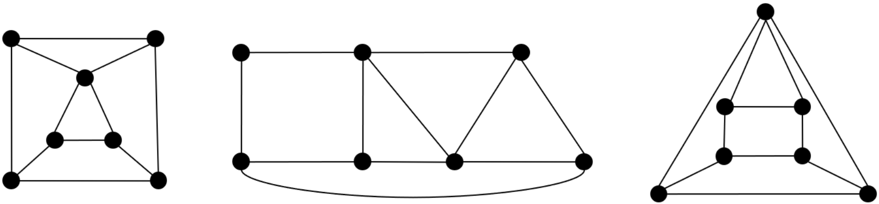
- although same number of vertices / edges, degree sequence
- Euler circuit, Hamiltonian Circuit: yes
- chromatic number, size of largest independent set: no

All these (and all other characteristics):

- can only be used to show non-isomorphism

Graph isomorphism

What about these graphs?



Graph isomorphism on rooted trees

We have seen:

- rooted <u>directed</u> trees, where all paths depart from the root

Graph isomorphism on rooted trees

We have seen:

- rooted directed trees, where all paths depart from the root

However, undirected rooted trees exist also:

- a tree T and a "specially designated" vertex r (the root)

Graph isomorphism on rooted trees

We have seen:

- rooted directed trees, where all paths depart from the root

However, undirected rooted trees exist also:

- a tree T and a "specially designated" vertex r (the root)

Definition (Isomorphism of rooted trees)

Two rooted trees $T_1(V_1, E_1, r_1)$ and $T_2(V_2, E_2, r_2)$ are called isomorphic if there exists an isomorphism bijection $f : V_1 \mapsto V_2$ such that $f(r_1) = r_2$.

Graph isomorphism on rooted trees

We have seen:

- rooted directed trees, where all paths depart from the root

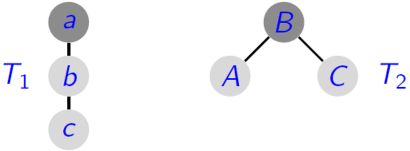
However, undirected rooted trees exist also:

- a tree T and a "specially designated" vertex r (the root)

Definition (Isomorphism of rooted trees)

Two rooted trees $T_1(V_1, E_1, r_1)$ and $T_2(V_2, E_2, r_2)$ are called isomorphic if there exists an isomorphism bijection $f : V_1 \mapsto V_2$ such that $f(r_1) = r_2$.

Example: T_1 and T_2 are isomorphic as graphs, but not as rooted trees !



Graph isomorphism on rooted trees

Definition

In a rooted tree T with root r, the level $L(i)$ is the set of vertices at distance i from the root r.

Graph isomorphism on rooted trees

Definition

In a rooted tree T with root r, the level $L(i)$ is the set of vertices at distance i from the root r.

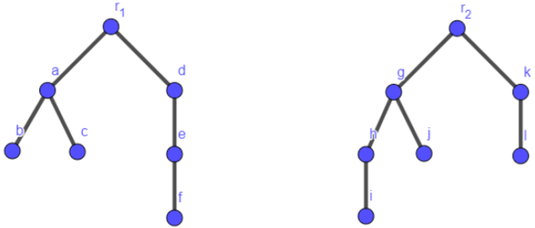
Exercise: Give an example of two rooted trees with the same number of levels, same number of vertices per level, and degree sequence per level, but are not isomorphic to each other

Graph isomorphism on rooted trees

Definition

In a rooted tree T with root r, the level $L(i)$ is the set of vertices at distance i from the root r.

Exercise: Give an example of two rooted trees with the same number of levels, same number of vertices per level, and degree sequence per level, but are not isomorphic to each other



Graph isomorphism on rooted trees

An isomorphism algorithm for rooted trees (Algorithm 2):

Algorithm 1 LabelRootedTree(T, v)

```
if v is a leaf of T then
    label(v) ← "10"
else
    for every child w of v do
        ℓ(w) ← label(w)
        Sort the labels of the children of v decreasingly: ℓ_1 ≥ ℓ_2 ≥ ... ≥ ℓ_k
        label(v) ← "1" ℓ_1 ℓ_2 ... ℓ_k "0"
return label(v)
```

Graph isomorphism on rooted trees

An isomorphism algorithm for rooted trees (Algorithm 2):

Algorithm 1 $\textsc{LabelRootedTree}(T, v)$

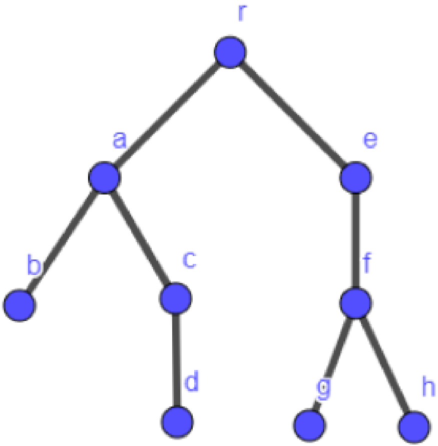
if v is a leaf of T **then**
    $label(v) \leftarrow$ "10"
else
    for every child w of v **do**
        $\ell(w) \leftarrow label(w)$
        Sort the labels of the children of v decreasingly: $\ell_1 \geq \ell_2 \geq \ldots \geq \ell_k$
        $label(v) \leftarrow$ "1" $\ell_1\ \ell_2 \ldots \ell_k$ "0"
return $label(v)$

Algorithm 2 $\textsc{Tree Isomorphism}((T_1, r_1), (T_2, r_2))$

$label(r_1) \leftarrow \textsc{LabelRootedTree}(T_1, r_1)$
$label(r_2) \leftarrow \textsc{LabelRootedTree}(T_2, r_2)$
if $label(r_1) = label(r_2)$ **then**
    return YES
else
    return NO

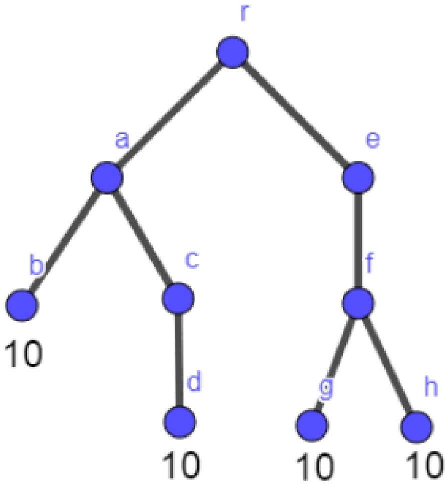
Graph isomorphism on rooted trees

Example:



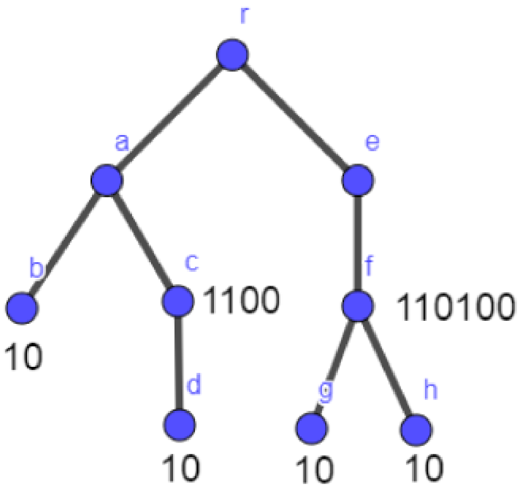
Graph isomorphism on rooted trees

Example:



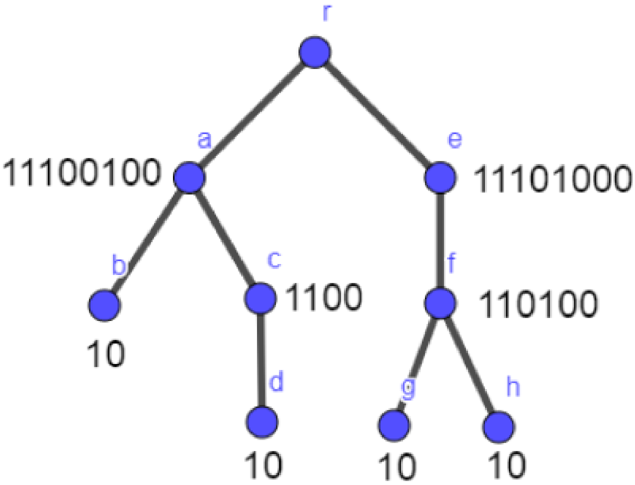
Graph isomorphism on rooted trees

Example:



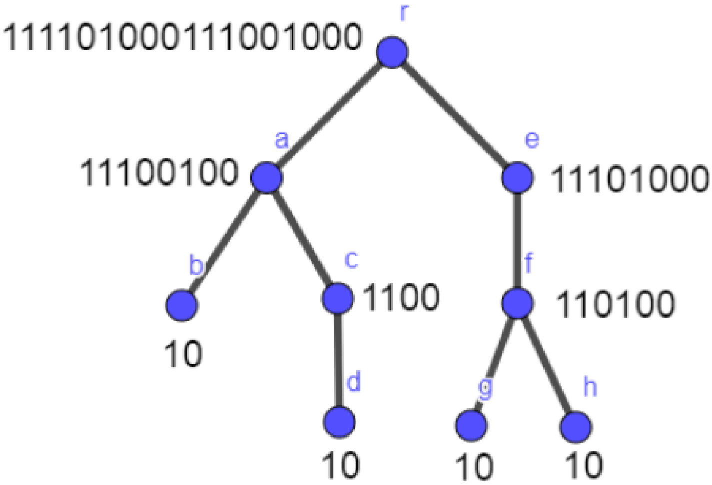
Graph isomorphism on rooted trees

Example:



Graph isomorphism on rooted trees

Example:



Graph isomorphism on rooted trees

Theorem

If (T_1, r_1) is isomorphic to (T_2, r_2) then Algorithm 2 returns YES.

Graph isomorphism on rooted trees

Theorem

If (T_1, r_1) is isomorphic to (T_2, r_2) then Algorithm 2 returns YES.

Proof by induction on the number k of levels.

Graph isomorphism on rooted trees

Theorem

If (T_1, r_1) is isomorphic to (T_2, r_2) then Algorithm 2 returns YES.

Proof by induction on the number k of levels.

Induction basis ($k = 1$): the two trees have only their root (thus isomorphic), which gets label 10. Thus the algorithm returns YES.

Graph isomorphism on rooted trees

Theorem

If (T_1, r_1) is isomorphic to (T_2, r_2) then Algorithm 2 returns YES.

Proof by induction on the number k of levels.

Induction basis ($k = 1$): the two trees have only their root (thus isomorphic), which gets label 10. Thus the algorithm returns YES.

Induction hypothesis: If two rooted trees (X_1, a_1) and (X_2, a_2) are isomorphic and have both k levels then the algorithm returns YES on input $((X_1, a_1), (X_2, a_2))$.

Graph isomorphism on rooted trees

Theorem

If (T_1, r_1) is isomorphic to (T_2, r_2) then Algorithm 2 returns YES.

Proof by induction on the number k of levels.

Induction basis ($k = 1$): the two trees have only their root (thus isomorphic), which gets label 10. Thus the algorithm returns YES.

Induction hypothesis: If two rooted trees (X_1, a_1) and (X_2, a_2) are isomorphic and have both k levels then the algorithm returns YES on input $((X_1, a_1), (X_2, a_2))$.

Induction step: Let (T_1, r_1) and (T_2, r_2) be two isomorphic rooted trees, each with $k + 1$ levels.

Let f be the bijection between T_1 and T_2 (from the definition). Then $f(r_1) = f(r_2)$ and, for every child a_1 of r_1 there exists a child a_2 of r_2 such that:

- the subtrees $(T_1(a_1), a_1)$ and $(T_2(a_2), a_2)$ are isomorphic, and
- $f(a_1) = f(a_2)$

Graph isomorphism on rooted trees

Proof (cont.):

Since every such pair of trees $(T_1(a_1), a_1)$ and $(T_2(a_2), a_2)$ is isomorphic, they have both k levels. Thus, by the induction hypothesis, Algorithm 1 returns the same labels for their roots, i.e. $label(a_1) = label(a_2)$.

Graph isomorphism on rooted trees

Proof (cont.):

Since every such pair of trees $(T_1(a_1), a_1)$ and $(T_2(a_2), a_2)$ is isomorphic, they have both k levels. Thus, by the induction hypothesis, Algorithm 1 returns the same labels for their roots, i.e. $label(a_1) = label(a_2)$.

Algorithm 1 now computes $label(r_1)$ by decreasingly sorting the labels of the children of r_1, say $\ell_1 \geq \ldots \geq \ell_p$, and then it sets $label(r_1) =$ "1" $\ell_1 \ell_2 \ldots \ell_p$ "0".

Graph isomorphism on rooted trees

Proof (cont.):

Since every such pair of trees $(T_1(a_1), a_1)$ and $(T_2(a_2), a_2)$ is isomorphic, they have both k levels. Thus, by the induction hypothesis, Algorithm 1 returns the same labels for their roots, i.e. $label(a_1) = label(a_2)$.

Algorithm 1 now computes $label(r_1)$ by decreasingly sorting the labels of the children of r_1, say $\ell_1 \geq \ldots \geq \ell_p$, and then it sets $label(r_1) =$ "1" $\ell_1 \ell_2 \ldots \ell_p$ "0".

Exactly the same happens for $label(r_2)$, i.e. $label(r_2) =$ "1" $\ell'_1 \ell'_2 \ldots \ell'_p$ "0".

Graph isomorphism on rooted trees

Proof (cont.):

Since every such pair of trees $(T_1(a_1), a_1)$ and $(T_2(a_2), a_2)$ is isomorphic, they have both k levels. Thus, by the induction hypothesis, Algorithm 1 returns the same labels for their roots, i.e. $label(a_1) = label(a_2)$.

Algorithm 1 now computes $label(r_1)$ by decreasingly sorting the labels of the children of r_1, say $\ell_1 \geq \ldots \geq \ell_p$, and then it sets $label(r_1) =$ "1" $\ell_1 \ell_2 \ldots \ell_p$ "0".

Exactly the same happens for $label(r_2)$, i.e. $label(r_2) =$ "1" $\ell'_1 \ell'_2 \ldots \ell'_p$ "0".

Since all these labels are the same (by the induction hypothesis): $label(r_1) = label(r_2)$, and thus Algorithm 2 returns YES. $\square$

Graph isomorphism on rooted trees

Proof (cont.):

Since every such pair of trees $(T_1(a_1), a_1)$ and $(T_2(a_2), a_2)$ is isomorphic, they have both k levels. Thus, by the induction hypothesis, Algorithm 1 returns the same labels for their roots, i.e. $label(a_1) = label(a_2)$.

Algorithm 1 now computes $label(r_1)$ by decreasingly sorting the labels of the children of r_1, say $\ell_1 \geq \ldots \geq \ell_p$, and then it sets $label(r_1) =$ "1" $\ell_1 \ell_2 \ldots \ell_p$ "0".

Exactly the same happens for $label(r_2)$, i.e. $label(r_2) =$ "1" $\ell'_1 \ell'_2 \ldots \ell'_p$ "0".

Since all these labels are the same (by the induction hypothesis):
$label(r_1) = label(r_2)$, and thus Algorithm 2 returns YES. □

Does this theorem show that Algorithm 2 is a correct isomorphism algorithm for rooted trees?

Graph isomorphism on rooted trees

Proof (cont.):

Since every such pair of trees $(T_1(a_1), a_1)$ and $(T_2(a_2), a_2)$ is isomorphic, they have both k levels. Thus, by the induction hypothesis, Algorithm 1 returns the same labels for their roots, i.e. $label(a_1) = label(a_2)$.

Algorithm 1 now computes $label(r_1)$ by decreasingly sorting the labels of the children of r_1, say $\ell_1 \geq \ldots \geq \ell_p$, and then it sets $label(r_1) =$ "1" $\ell_1 \ell_2 \ldots \ell_p$ "0".

Exactly the same happens for $label(r_2)$, i.e. $label(r_2) =$ "1" $\ell'_1 \ell'_2 \ldots \ell'_p$ "0".

Since all these labels are the same (by the induction hypothesis):
$label(r_1) = label(r_2)$, and thus Algorithm 2 returns YES. □

Does this theorem show that Algorithm 2 is a correct isomorphism algorithm for rooted trees?

- we also need the reverse direction: if two trees are not isomorphic, then the algorithm returns NO
- this can be also proved (a bit more tricky)

Additional slides

How to find and write down proofs?

Let's point out relevant stages in the process of proving statements by going in detail through one example.

Before doing so, let's point out the following questions you could ask yourself to help finding a possible proof approach.

How to find and write down proofs?

Let's point out relevant stages in the process of proving statements by going in detail through one example.

Before doing so, let's point out the following questions you could ask yourself to help finding a possible proof approach.

- What do I have to prove? Is it one statement, or several; is it an implication or an equivalence; can I rephrase it; does it resemble other statements?
- What do I know? What are the assumptions; do I know the relevant definitions; is there any known theory related to the statement?

How to find and write down proofs?

Let's point out relevant stages in the process of proving statements by going in detail through one example.

Before doing so, let's point out the following questions you could ask yourself to help finding a possible proof approach.

- What do I have to prove? Is it one statement, or several; is it an implication or an equivalence; can I rephrase it; does it resemble other statements?
- What do I know? What are the assumptions; do I know the relevant definitions; is there any known theory related to the statement?
- Can I get more insight? Can I sketch the situation, the assumptions, the question; are there special (small) cases to check; can I break it into several subcases?
- How to approach/attack the question? Can I use induction; does a direct proof have any chance; or does it help to use contradiction?

How to find and write down proofs?

Let's point out relevant stages in the process of proving statements by going in detail through one example.

Before doing so, let's point out the following questions you could ask yourself to help finding a possible proof approach.

- What do I have to prove? Is it one statement, or several; is it an implication or an equivalence; can I rephrase it; does it resemble other statements?
- What do I know? What are the assumptions; do I know the relevant definitions; is there any known theory related to the statement?
- Can I get more insight? Can I sketch the situation, the assumptions, the question; are there special (small) cases to check; can I break it into several subcases?
- How to approach/attack the question? Can I use induction; does a direct proof have any chance; or does it help to use contradiction?
- Is my solution valid and convincing? Write a draft first; check all the steps; critically examine the steps for errors or counterexamples; modify and revise the solution and write it down in a clear way.

The start: Write down what you see

We will consider the process of finding the proof on the following example:

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

The start: Write down what you see

We will consider the process of finding the proof on the following example:

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

- Clearly, you have to know what a tree is, what a forest is, and what the notations $e \in E(T)$ and $T - e$ mean.

The start: Write down what you see

We will consider the process of finding the proof on the following example:

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

- Clearly, you have to know what a tree is, what a forest is, and what the notations $e \in E(T)$ and $T - e$ mean.
- In fact, you have to prove two (or perhaps even three) statements: $T - e$ is a forest, and this forest consists of precisely two trees (so not ≤ 1 and not ≥ 3 trees).

The start: Write down what you see

We will consider the process of finding the proof on the following example:

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

- Clearly, you have to know what a tree is, what a forest is, and what the notations $e \in E(T)$ and $T - e$ mean.
- In fact, you have to prove two (or perhaps even three) statements: $T - e$ is a forest, and this forest consists of precisely two trees (so not ≤ 1 and not ≥ 3 trees).
- Here it (probably) helps to draw a picture that roughly sketches the situation and concepts. I will not include it on the slides.

Example continued

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

Example continued

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

- If you draw the general situation, and know the definitions and notations, then you more or less see the solution in the picture.
- The question is how to write it down (and check that the picture did not fool you).
- This requires certain skills and experience.

Example continued

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

- If you draw the general situation, and know the definitions and notations, then you more or less see the solution in the picture.
- The question is how to write it down (and check that the picture did not fool you).
- This requires certain skills and experience.
- You can only learn this by doing it yourself.
- You do not learn it by reading (although this helps if you force yourself to understand every step you read).

Example continued

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

- A tree is a connected graph without cycles.
- A forest is a graph without cycles.
- Since a tree is a connected graph, between any two vertices there is a path in a tree.
- We even know from the previous lecture that this path is unique.

Example continued

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

- A tree is a connected graph without cycles.
- A forest is a graph without cycles.
- Since a tree is a connected graph, between any two vertices there is a path in a tree.
- We even know from the previous lecture that this path is unique.

How to use (some of) the above facts to prove that $T - e$ is a forest consisting of precisely two trees?

Let us consider the first part of the statement first. Can we prove that $T - e$ is a forest?

Example: the first part

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest.

Example: the first part

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest.

This is an easy consequence of the definitions and the observation that removing edges from a tree, we cannot introduce cycles. So if T is a tree, then T contains no cycles and $T - e$ contains no cycles either, so $T - e$ is a forest. (This is a direct proof.)

Example: the first part

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest.

This is an easy consequence of the definitions and the observation that removing edges from a tree, we cannot introduce cycles. So if T is a tree, then T contains no cycles and $T - e$ contains no cycles either, so $T - e$ is a forest. (This is a direct proof.)

It remains to show that $T - e$ consists of precisely 2 trees, i.e., at least 2 and at most 2 trees. How to prove this?

Example: the first part

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest.

This is an easy consequence of the definitions and the observation that removing edges from a tree, we cannot introduce cycles. So if T is a tree, then T contains no cycles and $T - e$ contains no cycles either, so $T - e$ is a forest. (This is a direct proof.)

It remains to show that $T - e$ consists of precisely 2 trees, i.e., at least 2 and at most 2 trees. How to prove this?

At least 2: you have to show that $T - e$ is not connected (not 1 tree). This is easy: if u and v are the end vertices of the edge e, then in $T - e$ there is no path between u and v. (Why not?) (This is also a direct proof.)

Example: the first part

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest.

This is an easy consequence of the definitions and the observation that removing edges from a tree, we cannot introduce cycles. So if T is a tree, then T contains no cycles and $T - e$ contains no cycles either, so $T - e$ is a forest. (This is a direct proof.)

It remains to show that $T - e$ consists of precisely 2 trees, i.e., at least 2 and at most 2 trees. How to prove this?

At least 2: you have to show that $T - e$ is not connected (not 1 tree). This is easy: if u and v are the end vertices of the edge e, then in $T - e$ there is no path between u and v. (Why not?) (This is also a direct proof.)

At most 2: you have to show that $T - e$ does not consist of 3 or more trees. This is easy, using the observation that the edge e can only connect 2 trees into one. (Why is this?) So, if $T - e$ would consist of 3 or more trees, then T is not connected, a contradiction. (This is a proof by contradiction or contraposition.)

Example: the first part

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest.

This is an easy consequence of the definitions and the observation that removing edges from a tree, we cannot introduce cycles. So if T is a tree, then T contains no cycles and $T - e$ contains no cycles either, so $T - e$ is a forest. (This is a direct proof.)

It remains to show that $T - e$ consists of precisely 2 trees, i.e., at least 2 and at most 2 trees. How to prove this?

At least 2: you have to show that $T - e$ is not connected (not 1 tree). This is easy: if u and v are the end vertices of the edge e, then in $T - e$ there is no path between u and v. (Why not?) (This is also a direct proof.)

At most 2: you have to show that $T - e$ does not consist of 3 or more trees. This is easy, using the observation that the edge e can only connect 2 trees into one. (Why is this?) So, if $T - e$ would consist of 3 or more trees, then T is not connected, a contradiction. (This is a proof by contradiction or contraposition.)

The proof seems to be complete. Now you have to write it down and carefully check the details.

Example: a solution

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

Example: a solution

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

Proof.

Since T is a tree, $T - e$ has no cycles, so $T - e$ is a forest. Since in $T - e$ there is no path between the two end vertices of e, $T - e$ is not connected, hence $T - e$ consists of at least 2 trees. If $T - e$ consists of at least 3 trees, then T cannot be connected, since a single edge can only connect two connected components into one. Hence $T - e$ is a forest consisting of precisely two trees. $\square$

Example: a solution

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

Proof.

Since T is a tree, $T - e$ has no cycles, so $T - e$ is a forest. Since in $T - e$ there is no path between the two end vertices of e, $T - e$ is not connected, hence $T - e$ consists of at least 2 trees. If $T - e$ consists of at least 3 trees, then T cannot be connected, since a single edge can only connect two connected components into one. Hence $T - e$ is a forest consisting of precisely two trees. □

There are probably many different correct ways to prove the lemma. For instance, for the last part you could use the fact that a tree on n vertices has $n - 1$ edges.

Example: a solution

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

Proof.

Since T is a tree, $T - e$ has no cycles, so $T - e$ is a forest. Since in $T - e$ there is no path between the two end vertices of e, $T - e$ is not connected, hence $T - e$ consists of at least 2 trees. If $T - e$ consists of at least 3 trees, then T cannot be connected, since a single edge can only connect two connected components into one. Hence $T - e$ is a forest consisting of precisely two trees. $\square$

There are probably many different correct ways to prove the lemma. For instance, for the last part you could use the fact that a tree on n vertices has $n - 1$ edges.

So suppose that $T - e$ consists of trees on $n_1, \ldots, n_k$ vertices for some integer $k \geq 1$. Now we count the number of edges of T in two ways:

Example: a solution

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

Proof.

Since T is a tree, $T - e$ has no cycles, so $T - e$ is a forest. Since in $T - e$ there is no path between the two end vertices of e, $T - e$ is not connected, hence $T - e$ consists of at least 2 trees. If $T - e$ consists of at least 3 trees, then T cannot be connected, since a single edge can only connect two connected components into one. Hence $T - e$ is a forest consisting of precisely two trees. □

There are probably many different correct ways to prove the lemma. For instance, for the last part you could use the fact that a tree on n vertices has $n - 1$ edges.

So suppose that $T - e$ consists of trees on $n_1, \ldots, n_k$ vertices for some integer $k \geq 1$. Now we count the number of edges of T in two ways: As T has $n_1 + \ldots + n_k$ vertices, T has $n_1 + \ldots + n_k - 1$ edges.

Example: a solution

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

Proof.

Since T is a tree, $T - e$ has no cycles, so $T - e$ is a forest. Since in $T - e$ there is no path between the two end vertices of e, $T - e$ is not connected, hence $T - e$ consists of at least 2 trees. If $T - e$ consists of at least 3 trees, then T cannot be connected, since a single edge can only connect two connected components into one. Hence $T - e$ is a forest consisting of precisely two trees. □

There are probably many different correct ways to prove the lemma. For instance, for the last part you could use the fact that a tree on n vertices has $n - 1$ edges.

So suppose that $T - e$ consists of trees on $n_1, \ldots, n_k$ vertices for some integer $k \geq 1$. Now we count the number of edges of T in two ways: As T has $n_1 + \ldots + n_k$ vertices, T has $n_1 + \ldots + n_k - 1$ edges. On the other hand, T has $(n_1 - 1) + \ldots + (n_k - 1) + 1$ edges.

Example: a solution

Lemma

Let T be a tree on $n \geq 2$ vertices, and let $e \in E(T)$. Then $T - e$ is a forest consisting of precisely two trees.

Proof.

Since T is a tree, $T - e$ has no cycles, so $T - e$ is a forest. Since in $T - e$ there is no path between the two end vertices of e, $T - e$ is not connected, hence $T - e$ consists of at least 2 trees. If $T - e$ consists of at least 3 trees, then T cannot be connected, since a single edge can only connect two connected components into one. Hence $T - e$ is a forest consisting of precisely two trees. □

There are probably many different correct ways to prove the lemma. For instance, for the last part you could use the fact that a tree on n vertices has $n - 1$ edges.

So suppose that $T - e$ consists of trees on $n_1, \ldots, n_k$ vertices for some integer $k \geq 1$. Now we count the number of edges of T in two ways: As T has $n_1 + \ldots + n_k$ vertices, T has $n_1 + \ldots + n_k - 1$ edges. On the other hand, T has $(n_1 - 1) + \ldots + (n_k - 1) + 1$ edges. The two expressions can only be equal if $k = 2$, so $T - e$ consists of precisely 2 trees.

Full m-ary trees

Definitions

A rooted tree is called an m-ary tree if each internal vertex has at most m children.

Full m-ary trees

Definitions

A rooted tree is called an m-ary tree if each internal vertex has at most m children.
It is a full m-ary tree if each internal vertex has exactly m children.

Full m-ary trees

Definitions

A rooted tree is called an *m-ary tree* if each internal vertex has at most m children.
It is a full m-ary tree if each internal vertex has exactly m children.

A (full) 2-ary tree is usually called a (full) binary tree.

Often, the children of each node are assumed to be ordered: 1st, 2nd, etc
(or left and right for binary trees)

Full m-ary trees

Definitions

A rooted tree is called an m-ary tree if each internal vertex has at most m children.
It is a full m-ary tree if each internal vertex has exactly m children.

A (full) 2-ary tree is usually called a (full) binary tree.

Often, the children of each node are assumed to be ordered: 1st, 2nd, etc (or left and right for binary trees)

Lemma

A full m-ary tree with i internal nodes has $n = m \cdot i + 1$ vertices.

Full m-ary trees

Definitions

A rooted tree is called an *m-ary tree* if each internal vertex has at most m children.
It is a full m-ary tree if each internal vertex has exactly m children.

A (full) 2-ary tree is usually called a (full) binary tree.

Often, the children of each node are assumed to be ordered: 1st, 2nd, etc
(or left and right for binary trees)

Lemma

A full m-ary tree with i internal nodes has $n = m \cdot i + 1$ *vertices.*

Proof.

Every node except the root is one of m children of a unique internal vertex. □

Full m-ary trees

Definitions

A rooted tree is called an m-ary tree if each internal vertex has at most m children.
It is a full m-ary tree if each internal vertex has exactly m children.

A (full) 2-ary tree is usually called a (full) binary tree.

Often, the children of each node are assumed to be ordered: 1st, 2nd, etc
(or left and right for binary trees)

Lemma

A full m-ary tree with i internal nodes has $n = m \cdot i + 1$ vertices.

Proof.

Every node except the root is one of m children of a unique internal vertex. □

Let ℓ be the number of leaves a full m-ary tree. Since $n = i + \ell$ and $n = m \cdot i + 1$, if we know any of n, i, ℓ then we can find all of them.

Exercise

- Suppose someone starts a chain letter. Each person who receives it is asked to send it on to four other people. Some people do this, some don't.
- It ended after there have been 100 people who have seen the letter, but did not send it out.
- How many people have seen this letter, including the first person, if no-one received it more than once?
- How many people sent out the letter?

Solution: model the situation as

Exercise

- Suppose someone starts a chain letter. Each person who receives it is asked to send it on to four other people. Some people do this, some don't.
- It ended after there have been 100 people who have seen the letter, but did not send it out.
- How many people have seen this letter, including the first person, if no-one received it more than once?
- How many people sent out the letter?

Solution: model the situation as a full 4-ary tree.

Exercise

- Suppose someone starts a chain letter. Each person who receives it is asked to send it on to four other people. Some people do this, some don't.
- It ended after there have been 100 people who have seen the letter, but did not send it out.
- How many people have seen this letter, including the first person, if no-one received it more than once?
- How many people sent out the letter?

Solution: model the situation as a full 4-ary tree.

- The person who started the chain letter is the root.

Exercise

- Suppose someone starts a chain letter. Each person who receives it is asked to send it on to four other people. Some people do this, some don't.
- It ended after there have been 100 people who have seen the letter, but did not send it out.
- How many people have seen this letter, including the first person, if no-one received it more than once?
- How many people sent out the letter?

Solution: model the situation as a full 4-ary tree.

- The person who started the chain letter is the root.
- Each person who sends it out is an internal node (with 4 children).

Exercise

- Suppose someone starts a chain letter. Each person who receives it is asked to send it on to four other people. Some people do this, some don't.
- It ended after there have been 100 people who have seen the letter, but did not send it out.
- How many people have seen this letter, including the first person, if no-one received it more than once?
- How many people sent out the letter?

Solution: model the situation as a full 4-ary tree.

- The person who started the chain letter is the root.
- Each person who sends it out is an internal node (with 4 children).
- Each person who receives it, but does not send it out, is a leaf.

Exercise

- Suppose someone starts a chain letter. Each person who receives it is asked to send it on to four other people. Some people do this, some don't.
- It ended after there have been 100 people who have seen the letter, but did not send it out.
- How many people have seen this letter, including the first person, if no-one received it more than once?
- How many people sent out the letter?

Solution: model the situation as a full 4-ary tree.

- The person who started the chain letter is the root.
- Each person who sends it out is an internal node (with 4 children).
- Each person who receives it, but does not send it out, is a leaf.
- We know that $\ell = 100$ and need to find n and i.

Exercise

- Suppose someone starts a chain letter. Each person who receives it is asked to send it on to four other people. Some people do this, some don't.
- It ended after there have been 100 people who have seen the letter, but did not send it out.
- How many people have seen this letter, including the first person, if no-one received it more than once?
- How many people sent out the letter?

Solution: model the situation as a full 4-ary tree.

- The person who started the chain letter is the root.
- Each person who sends it out is an internal node (with 4 children).
- Each person who receives it, but does not send it out, is a leaf.
- We know that $\ell = 100$ and need to find n and i.
- We have $n = 4i + 1$ and $n = i + 100$.

Exercise

- Suppose someone starts a chain letter. Each person who receives it is asked to send it on to four other people. Some people do this, some don't.
- It ended after there have been 100 people who have seen the letter, but did not send it out.
- How many people have seen this letter, including the first person, if no-one received it more than once?
- How many people sent out the letter?

Solution: model the situation as a full 4-ary tree.

- The person who started the chain letter is the root.
- Each person who sends it out is an internal node (with 4 children).
- Each person who receives it, but does not send it out, is a leaf.
- We know that $\ell = 100$ and need to find n and i.
- We have $n = 4i + 1$ and $n = i + 100$.
- Solving this, get $n = 133$ and $i = 33$.

The height of a rooted tree

Definitions

In a rooted tree, the level of a vertex u is the length of the (unique) path from the root to u. (The level of the root is 0).

The height of a rooted tree

Definitions

In a rooted tree, the level of a vertex u is the length of the (unique) path from the root to u. (The level of the root is 0).
The height of a rooted tree is the maximum level of a vertex in it.

The height of a rooted tree

Definitions

In a rooted tree, the level of a vertex u is the length of the (unique) path from the root to u. (The level of the root is 0).
The height of a rooted tree is the maximum level of a vertex in it.

Theorem

There are at most m^h leaves in an m-ary tree of height h.

The height of a rooted tree

Definitions

In a rooted tree, the level of a vertex u is the length of the (unique) path from the root to u. (The level of the root is 0).
The height of a rooted tree is the maximum level of a vertex in it.

Theorem

There are at most m^h leaves in an m-ary tree of height h.

Proof.

Induction on the height h.

- Base: If $h = 1$ then the claim is obvious.

The height of a rooted tree

Definitions

In a rooted tree, the level of a vertex u is the length of the (unique) path from the root to u. (The level of the root is 0).
The height of a rooted tree is the maximum level of a vertex in it.

Theorem

There are at most m^h leaves in an m-ary tree of height h.

Proof.

Induction on the height h.

- Base: If $h = 1$ then the claim is obvious.
- Step: Assume the claim is true for m-ary trees of height at most $h - 1$.

The height of a rooted tree

Definitions

In a rooted tree, the level of a vertex u is the length of the (unique) path from the root to u. (The level of the root is 0).
The height of a rooted tree is the maximum level of a vertex in it.

Theorem

There are at most m^h leaves in an m-ary tree of height h.

Proof.

Induction on the height h.

- Base: If $h = 1$ then the claim is obvious.
- Step: Assume the claim is true for m-ary trees of height at most $h - 1$.
- Take an m-ary tree T of height $h \geq 2$, with root r

The height of a rooted tree

Definitions

In a rooted tree, the level of a vertex u is the length of the (unique) path from the root to u. (The level of the root is 0).
The height of a rooted tree is the maximum level of a vertex in it.

Theorem

There are at most m^h leaves in an m-ary tree of height h.

Proof.

Induction on the height h.

- Base: If $h = 1$ then the claim is obvious.
- Step: Assume the claim is true for m-ary trees of height at most $h - 1$.
- Take an m-ary tree T of height $h \geq 2$, with root r
- Consider the subtrees of T rooted at children of r.

The height of a rooted tree

Definitions

In a rooted tree, the level of a vertex u is the length of the (unique) path from the root to u. (The level of the root is 0).
The height of a rooted tree is the maximum level of a vertex in it.

Theorem

There are at most m^h leaves in an m-ary tree of height h.

Proof.

Induction on the height h.

- Base: If $h = 1$ then the claim is obvious.
- Step: Assume the claim is true for m-ary trees of height at most $h - 1$.
- Take an m-ary tree T of height $h \geq 2$, with root r
- Consider the subtrees of T rooted at children of r.
- There are at most m of them, and, by induction hypothesis, each has at most m^{h-1} leaves.

The height of a rooted tree

Definitions

In a rooted tree, the level of a vertex u is the length of the (unique) path from the root to u. (The level of the root is 0).
The height of a rooted tree is the maximum level of a vertex in it.

Theorem

There are at most m^h leaves in an m-ary tree of height h.

Proof.

Induction on the height h.

- Base: If $h = 1$ then the claim is obvious.
- Step: Assume the claim is true for m-ary trees of height at most $h - 1$.
- Take an m-ary tree T of height $h \geq 2$, with root r
- Consider the subtrees of T rooted at children of r.
- There are at most m of them, and, by induction hypothesis, each has at most m^{h-1} leaves.
- Hence, T has at most $m \cdot m^{h-1} = m^h$ leaves.

Balanced m-ary trees

Definition

An m-ary tree of height h is balanced if all leaves in it have height $h-1$ or h.

Balanced m-ary trees

Definition

An m-ary tree of height h is balanced if all leaves in it have height $h-1$ or h.

Theorem

If an m-ary tree of height h has ℓ leaves then $h \geq \lceil \log_m \ell \rceil$.

Balanced m-ary trees

Definition

An m-ary tree of height h is balanced if all leaves in it have height $h-1$ or h.

Theorem

If an m-ary tree of height h has ℓ leaves then $h \geq \lceil \log_m \ell \rceil$.
If the tree is full and balanced then $h = \lceil \log_m \ell \rceil$.

Balanced m-ary trees

Definition

An m-ary tree of height h is balanced if all leaves in it have height $h-1$ or h.

Theorem

If an m-ary tree of height h has ℓ leaves then $h \geq \lceil \log_m \ell \rceil$.
If the tree is full and balanced then $h = \lceil \log_m \ell \rceil$.

(For a real number x, $\lceil x \rceil$ is the smallest integer greater than or equal to x).

Balanced m-ary trees

Definition

An m-ary tree of height h is balanced if all leaves in it have height $h-1$ or h.

Theorem

If an m-ary tree of height h has ℓ leaves then $h \geq \lceil \log_m \ell \rceil$.
If the tree is full and balanced then $h = \lceil \log_m \ell \rceil$.

(For a real number x, $\lceil x \rceil$ is the smallest integer greater than or equal to x).

Proof.

- The first part immediately follows from the previous theorem:

Balanced m-ary trees

Definition

An m-ary tree of height h is balanced if all leaves in it have height $h-1$ or h.

Theorem

If an m-ary tree of height h has ℓ leaves then $h \geq \lceil \log_m \ell \rceil$.
If the tree is full and balanced then $h = \lceil \log_m \ell \rceil$.

(For a real number x, $\lceil x \rceil$ is the smallest integer greater than or equal to x).

Proof.

- The first part immediately follows from the previous theorem:
  We know that $\ell \leq m^h$, so $h \geq \log_m \ell$.

Balanced m-ary trees

Definition

An m-ary tree of height h is balanced if all leaves in it have height $h-1$ or h.

Theorem

If an m-ary tree of height h has ℓ leaves then $h \geq \lceil \log_m \ell \rceil$.
If the tree is full and balanced then $h = \lceil \log_m \ell \rceil$.

(For a real number x, $\lceil x \rceil$ is the smallest integer greater than or equal to x).

Proof.

- The first part immediately follows from the previous theorem:
  We know that $\ell \leq m^h$, so $h \geq \log_m \ell$. Since h is an integer, $h \geq \lceil \log_m \ell \rceil$.

Balanced m-ary trees

Definition

An m-ary tree of height h is balanced if all leaves in it have height $h-1$ or h.

Theorem

If an m-ary tree of height h has ℓ leaves then $h \geq \lceil \log_m \ell \rceil$.
If the tree is full and balanced then $h = \lceil \log_m \ell \rceil$.

(For a real number x, $\lceil x \rceil$ is the smallest integer greater than or equal to x).

Proof.

- The first part immediately follows from the previous theorem:
  We know that $\ell \leq m^h$, so $h \geq \log_m \ell$. Since h is an integer, $h \geq \lceil \log_m \ell \rceil$.
- For the second part, note that there is at least one leaf of level h.

Balanced m-ary trees

Definition

An m-ary tree of height h is balanced if all leaves in it have height $h-1$ or h.

Theorem

If an m-ary tree of height h has ℓ leaves then $h \geq \lceil \log_m \ell \rceil$.
If the tree is full and balanced then $h = \lceil \log_m \ell \rceil$.

(For a real number x, $\lceil x \rceil$ is the smallest integer greater than or equal to x).

Proof.

- The first part immediately follows from the previous theorem:
  We know that $\ell \leq m^h$, so $h \geq \log_m \ell$. Since h is an integer, $h \geq \lceil \log_m \ell \rceil$.
- For the second part, note that there is at least one leaf of level h.
- It follows that there are at least m^{h-1} leaves (why?).

Balanced m-ary trees

Definition

An m-ary tree of height h is balanced if all leaves in it have height $h-1$ or h.

Theorem

If an m-ary tree of height h has ℓ leaves then $h \geq \lceil \log_m \ell \rceil$.
If the tree is full and balanced then $h = \lceil \log_m \ell \rceil$.

(For a real number x, $\lceil x \rceil$ is the smallest integer greater than or equal to x).

Proof.

- The first part immediately follows from the previous theorem:
  We know that $\ell \leq m^h$, so $h \geq \log_m \ell$. Since h is an integer, $h \geq \lceil \log_m \ell \rceil$.
- For the second part, note that there is at least one leaf of level h.
- It follows that there are at least m^{h-1} leaves (why?).
- So, we have $m^{h-1} < \ell \leq m^h$, or taking logarithm to the base m,
  $h-1 < \log_m \ell \leq h$.

Balanced m-ary trees

Definition

An m-ary tree of height h is balanced if all leaves in it have height $h-1$ or h.

Theorem

If an m-ary tree of height h has ℓ leaves then $h \geq \lceil \log_m \ell \rceil$.
If the tree is full and balanced then $h = \lceil \log_m \ell \rceil$.

(For a real number x, $\lceil x \rceil$ is the smallest integer greater than or equal to x).

Proof.

- The first part immediately follows from the previous theorem:
  We know that $\ell \leq m^h$, so $h \geq \log_m \ell$. Since h is an integer, $h \geq \lceil \log_m \ell \rceil$.
- For the second part, note that there is at least one leaf of level h.
- It follows that there are at least m^{h-1} leaves (why?).
- So, we have $m^{h-1} < \ell \leq m^h$, or taking logarithm to the base m,
  $h - 1 < \log_m \ell \leq h$.
- Since h is an integer, $h = \lceil \log_m \ell \rceil$.

□

Constructing trees

Constructing trees

Every tree $T \neq K_1$ has a leaf. We know that $T - v$ is also a tree.

Constructing trees

Every tree $T \neq K_1$ has a leaf. We know that $T - v$ is also a tree. This shows that T can be constructed from a smaller tree $T' = T - v$ by adding a vertex to T' and joining it by one edge to a vertex in T'.

Constructing trees

Every tree $T \neq K_1$ has a leaf. We know that $T - v$ is also a tree. This shows that T can be constructed from a smaller tree $T' = T - v$ by adding a vertex to T' and joining it by one edge to a vertex in T'. This also proves the following statement.

Constructing trees

Every tree $T \neq K_1$ has a leaf. We know that $T - v$ is also a tree. This shows that T can be constructed from a smaller tree $T' = T - v$ by adding a vertex to T' and joining it by one edge to a vertex in T'. This also proves the following statement.

Lemma

We can construct all different trees on $n \geq 2$ vertices from all trees on $n - 1$ vertices, by adding one vertex and joining it by one edge to a vertex in one of the trees, in all possible ways, and deleting multiple copies of the same tree.

Constructing trees

Every tree $T \neq K_1$ has a leaf. We know that $T - v$ is also a tree. This shows that T can be constructed from a smaller tree $T' = T - v$ by adding a vertex to T' and joining it by one edge to a vertex in T'. This also proves the following statement.

Lemma

We can construct all different trees on $n \geq 2$ vertices from all trees on $n - 1$ vertices, by adding one vertex and joining it by one edge to a vertex in one of the trees, in all possible ways, and deleting multiple copies of the same tree.

- We can use the above result and proceed to obtain all different trees on n vertices, starting with K_1 (or we can give a recursive definition for the class of all trees).

Constructing trees

Every tree $T \neq K_1$ has a leaf. We know that $T - v$ is also a tree. This shows that T can be constructed from a smaller tree $T' = T - v$ by adding a vertex to T' and joining it by one edge to a vertex in T'. This also proves the following statement.

Lemma

We can construct all different trees on $n \geq 2$ vertices from all trees on $n - 1$ vertices, by adding one vertex and joining it by one edge to a vertex in one of the trees, in all possible ways, and deleting multiple copies of the same tree.

- We can use the above result and proceed to obtain all different trees on n vertices, starting with K_1 (or we can give a recursive definition for the class of all trees).
- Check that there are, respectively, 1, 1, 1, 2, 3, and 6 different trees on 1, 2, 3, 4, 5, and 6 vertices.

Constructing trees

Every tree $T \neq K_1$ has a leaf. We know that $T - v$ is also a tree. This shows that T can be constructed from a smaller tree $T' = T - v$ by adding a vertex to T' and joining it by one edge to a vertex in T'. This also proves the following statement.

Lemma

We can construct all different trees on $n \geq 2$ vertices from all trees on $n - 1$ vertices, by adding one vertex and joining it by one edge to a vertex in one of the trees, in all possible ways, and deleting multiple copies of the same tree.

- We can use the above result and proceed to obtain all different trees on n vertices, starting with K_1 (or we can give a recursive definition for the class of all trees).
- Check that there are, respectively, 1, 1, 1, 2, 3, and 6 different trees on 1, 2, 3, 4, 5, and 6 vertices.
- **Exercise:**: how many are there on 7 vertices?