Lecture 3: Trees and Isomorphism

Dr. Amitabh Trehan

amitabh.trehan@durham.ac.uk

**Based on the slides of ADS-21/22 by Dr. George Mertzios*

Reminder from last lecture

- Eulerian circuit: If we can travel along the edges of a given graph G so that we start and finish at the same vertex and traverse each edge exactly once.

Reminder from last lecture

- Eulerian circuit: If we can travel along the edges of a given graph G so that we start and finish at the same vertex and traverse each edge exactly once.

Theorem

A connected graph with at least two vertices has an Eulerian curcuit iff each of its vertices has even degree.

Reminder from last lecture

- Eulerian circuit: If we can travel along the edges of a given graph G so that we start and finish at the same vertex and traverse each edge exactly once.

Theorem

A connected graph with at least two vertices has an Eulerian curcuit iff each of its vertices has even degree.

- Hamiltonian cycle (after William Hamilton (1805-65): if we can travel along the edges of a given graph so that we start and finish at the same vertex and visit each vertex exactly once.

Reminder from last lecture

- Eulerian circuit: If we can travel along the edges of a given graph G so that we start and finish at the same vertex and traverse each edge exactly once.

Theorem

A connected graph with at least two vertices has an Eulerian curcuit iff each of its vertices has even degree.

- Hamiltonian cycle (after William Hamilton (1805-65): if we can travel along the edges of a given graph so that we start and finish at the same vertex and visit each vertex exactly once.
- Usually no ‘easy’/‘efficient’ way to know if a graph of general topology has a Hamiltonian cycle besides enumerating cycles one by one.

Contents for today's lecture

- Trees and their properties;
- Applications of trees;
- Exercises involving trees;
- Isomorphism
- Examples of proof techniques;
- Properties of rooted trees.

Trees

We now turn to a special graph class that has many applications in many areas.

Definitions

A forest is an acyclic graph, i.e. graph without cycles.

Trees

We now turn to a special graph class that has many applications in many areas.

Definitions

A forest is an acyclic graph, i.e. graph without cycles.
A tree is a connected forest, i.e. a connected acyclic graph.

Trees

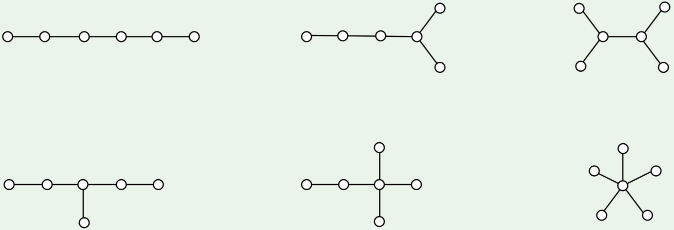
We now turn to a special graph class that has many applications in many areas.

Definitions

A forest is an acyclic graph, i.e. graph without cycles.
A tree is a connected forest, i.e. a connected acyclic graph.

Examples

The different trees on 6 vertices are shown below.



We can also consider this as a forest on 36 vertices.

Spanning trees

A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ is spanning if $V' = V$.

Spanning trees

A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ is spanning if $V' = V$.

Theorem

Every connected graph contains a spanning tree (a spanning subgraph that is a tree).

Spanning trees

A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ is spanning if $V' = V$.

Theorem

Every connected graph contains a spanning tree (a spanning subgraph that is a tree).

An algorithmic proof.

Let G be a connected graph.

- If G contains no cycles, it is a tree, and hence a spanning tree of itself.

Spanning trees

A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ is spanning if $V' = V$.

Theorem

Every connected graph contains a spanning tree (a spanning subgraph that is a tree).

An algorithmic proof.

Let G be a connected graph.

- If G contains no cycles, it is a tree, and hence a spanning tree of itself.
- If G contains a cycle, we can remove one edge from the cycle.
- The new graph is still connected. (Why?)

Spanning trees

A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ is spanning if $V' = V$.

Theorem

Every connected graph contains a spanning tree (a spanning subgraph that is a tree).

An algorithmic proof.

Let G be a connected graph.

- If G contains no cycles, it is a tree, and hence a spanning tree of itself.
- If G contains a cycle, we can remove one edge from the cycle.
- The new graph is still connected. (Why?)
- Repeating this, we can destroy all cycles and end up with a spanning tree. $\square$

Spanning trees

A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ is spanning if $V' = V$.

Theorem

Every connected graph contains a spanning tree (a spanning subgraph that is a tree).

An algorithmic proof.

Let G be a connected graph.

- If G contains no cycles, it is a tree, and hence a spanning tree of itself.
- If G contains a cycle, we can remove one edge from the cycle.
- The new graph is still connected. (Why?)
- Repeating this, we can destroy all cycles and end up with a spanning tree. □

How many repetitions do we need for the above algorithm?

Spanning trees

A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ is spanning if $V' = V$.

Theorem

Every connected graph contains a spanning tree (a spanning subgraph that is a tree).

An algorithmic proof.

Let G be a connected graph.

- If G contains no cycles, it is a tree, and hence a spanning tree of itself.
- If G contains a cycle, we can remove one edge from the cycle.
- The new graph is still connected. (Why?)
- Repeating this, we can destroy all cycles and end up with a spanning tree. □

How many repetitions do we need for the above algorithm?

It follows that trees are the smallest connected structures.

Spanning trees

A subgraph $G' = (V', E')$ of a graph $G = (V, E)$ is spanning if $V' = V$.

Theorem

Every connected graph contains a spanning tree (a spanning subgraph that is a tree).

An algorithmic proof.

Let G be a connected graph.

- If G contains no cycles, it is a tree, and hence a spanning tree of itself.
- If G contains a cycle, we can remove one edge from the cycle.
- The new graph is still connected. (Why?)
- Repeating this, we can destroy all cycles and end up with a spanning tree. $\square$

How many repetitions do we need for the above algorithm?

It follows that trees are the smallest connected structures.

Finding minimum-weight spanning trees in edge-weighted graphs is an important task in practice: we will learn fast algorithms for it in a few lectures.

Leaves in trees

A leaf in a tree is a vertex of degree 1.

Lemma

Every tree on at least two vertices contains a leaf.

Leaves in trees

A leaf in a tree is a vertex of degree 1.

Lemma

Every tree on at least two vertices contains a leaf.

Proof.

By contradiction:

- Assuming that every vertex has degree 0 or at least 2, we will show that the graph is not a tree.

Leaves in trees

A leaf in a tree is a vertex of degree 1.

Lemma

Every tree on at least two vertices contains a leaf.

Proof.

By contradiction:

- Assuming that every vertex has degree 0 or at least 2, we will show that the graph is not a tree.
- If a vertex has degree 0, then:

Leaves in trees

A leaf in a tree is a vertex of degree 1.

Lemma

Every tree on at least two vertices contains a leaf.

Proof.

By contradiction:

- Assuming that every vertex has degree 0 or at least 2, we will show that the graph is not a tree.
- If a vertex has degree 0, then: the graph (which contains at least two vertices) is not connected, hence not a tree.
- If every vertex has degree at least 2:

Leaves in trees

A leaf in a tree is a vertex of degree 1.

Lemma

Every tree on at least two vertices contains a leaf.

Proof.

By contradiction:

- Assuming that every vertex has degree 0 or at least 2, we will show that the graph is not a tree.
- If a vertex has degree 0, then: the graph (which contains at least two vertices) is not connected, hence not a tree.
- If every vertex has degree at least 2: just start at a vertex, go to one of its neighbours, from there go to another neighbour, etc.
- Since the vertex set is finite, at some stage we encounter a vertex we have already visited.

Leaves in trees

A leaf in a tree is a vertex of degree 1.

Lemma

Every tree on at least two vertices contains a leaf.

Proof.

By contradiction:

- Assuming that every vertex has degree 0 or at least 2, we will show that the graph is not a tree.
- If a vertex has degree 0, then: the graph (which contains at least two vertices) is not connected, hence not a tree.
- If every vertex has degree at least 2: just start at a vertex, go to one of its neighbours, from there go to another neighbour, etc.
- Since the vertex set is finite, at some stage we encounter a vertex we have already visited.
- This implies that the graph contains a cycle, so is not a tree, contradiction. □

Leaves in trees

A leaf in a tree is a vertex of degree 1.

Lemma

Every tree on at least two vertices contains a leaf.

Proof.

By contradiction:

- Assuming that every vertex has degree 0 or at least 2, we will show that the graph is not a tree.
- If a vertex has degree 0, then: the graph (which contains at least two vertices) is not connected, hence not a tree.
- If every vertex has degree at least 2: just start at a vertex, go to one of its neighbours, from there go to another neighbour, etc.
- Since the vertex set is finite, at some stage we encounter a vertex we have already visited.
- This implies that the graph contains a cycle, so is not a tree, contradiction. □

Can a tree have exactly one leaf?

Leaves in trees

A leaf in a tree is a vertex of degree 1.

Lemma

Every tree on at least two vertices contains a leaf.

Alternative proof (when every vertex has degree at least 2):

Proof.

- Consider a longest path P and an end vertex v of P.

Leaves in trees

A leaf in a tree is a vertex of degree 1.

Lemma

Every tree on at least two vertices contains a leaf.

Alternative proof (when every vertex has degree at least 2):

Proof.

- Consider a longest path P and an end vertex v of P.
- All neighbours of v are on P. (why?)

Leaves in trees

A leaf in a tree is a vertex of degree 1.

Lemma

Every tree on at least two vertices contains a leaf.

Alternative proof (when every vertex has degree at least 2):

Proof.

- Consider a longest path P and an end vertex v of P.
- All neighbours of v are on P. (why?)
- If $\deg(v) \geq 2$, then there is a cycle.
- The same also holds for the other end vertex u of P

$\Rightarrow \deg(u) = \deg(v) = 1$

□

Edges of trees

How many edges does a tree on n vertices have?

Edges of trees

How many edges does a tree on n vertices have?

Theorem

A connected graph on n vertices is a tree iff it has $n - 1$ edges.

Edges of trees

How many edges does a tree on n vertices have?

Theorem

A connected graph on n vertices is a tree iff it has $n - 1$ edges.

Proof.

($\Rightarrow$). Show, by induction on n, that a tree on n vertices has $n - 1$ edges.

Edges of trees

How many edges does a tree on n vertices have?

Theorem

A connected graph on n vertices is a tree iff it has $n - 1$ edges.

Proof.

($\Rightarrow$). Show, by induction on n, that a tree on n vertices has $n - 1$ edges.

- For small n the lemma holds: a tree on one vertex has no edges; a tree on two vertices has one edge.

Edges of trees

How many edges does a tree on n vertices have?

Theorem

A connected graph on n vertices is a tree iff it has $n-1$ edges.

Proof.

($\Rightarrow$). Show, by induction on n, that a tree on n vertices has $n-1$ edges.

- For small n the lemma holds: a tree on one vertex has no edges; a tree on two vertices has one edge.
- Suppose each tree on $n-1$ vertices has $n-2$ edges (induction hypothesis).
- Take a tree T on n vertices, for some $n \geq 3$.

Edges of trees

How many edges does a tree on n vertices have?

Theorem

A connected graph on n vertices is a tree iff it has $n-1$ edges.

Proof.

($\Rightarrow$). Show, by induction on n, that a tree on n vertices has $n-1$ edges.

- For small n the lemma holds: a tree on one vertex has no edges; a tree on two vertices has one edge.
- Suppose each tree on $n-1$ vertices has $n-2$ edges (induction hypothesis).
- Take a tree T on n vertices, for some $n \geq 3$.
- T contains a leaf v. Consider the graph $T-v$, it has one vertex less and one edge less than T.

Edges of trees

How many edges does a tree on n vertices have?

Theorem

A connected graph on n vertices is a tree iff it has $n-1$ edges.

Proof.

($\Rightarrow$). Show, by induction on n, that a tree on n vertices has $n-1$ edges.

- For small n the lemma holds: a tree on one vertex has no edges; a tree on two vertices has one edge.
- Suppose each tree on $n-1$ vertices has $n-2$ edges (induction hypothesis).
- Take a tree T on n vertices, for some $n \geq 3$.
- T contains a leaf v. Consider the graph $T - v$, it has one vertex less and one edge less than T.
- $T - v$ is still connected and (still) acyclic.

Edges of trees

How many edges does a tree on n vertices have?

Theorem

A connected graph on n vertices is a tree iff it has $n-1$ edges.

Proof.

($\Rightarrow$). Show, by induction on n, that a tree on n vertices has $n-1$ edges.

- For small n the lemma holds: a tree on one vertex has no edges; a tree on two vertices has one edge.
- Suppose each tree on $n-1$ vertices has $n-2$ edges (induction hypothesis).
- Take a tree T on n vertices, for some $n \geq 3$.
- T contains a leaf v. Consider the graph $T-v$, it has one vertex less and one edge less than T.
- $T-v$ is still connected and (still) acyclic.
- $T-v$ is a tree with $n-1$ vertices, by induction hypothesis it has $n-2$ edges.
- T has one edge more, so $n-1$ edges.

Edges of trees, cont'd

How many edges does a tree on n vertices have?

Theorem

A connected graph on n vertices is a tree if and only if it has $n - 1$ edges.

Proof.

$(\Leftarrow)$.

Edges of trees, cont'd

How many edges does a tree on n vertices have?

Theorem

A connected graph on n vertices is a tree if and only if it has $n-1$ edges.

Proof.

$(\Leftarrow)$.

- Assume that G is a connected graph with n vertices and $n-1$ edges.
- Then, as we proved before, G contains a spanning tree T.

Edges of trees, cont'd

How many edges does a tree on n vertices have?

Theorem

A connected graph on n vertices is a tree if and only if it has $n-1$ edges.

Proof.

($\Leftarrow$).

- Assume that G is a connected graph with n vertices and $n-1$ edges.
- Then, as we proved before, G contains a spanning tree T.
- By the first part of the proof, T contains exactly $n-1$ edges.

Edges of trees, cont'd

How many edges does a tree on n vertices have?

Theorem

A connected graph on n vertices is a tree if and only if it has $n-1$ edges.

Proof.

($\Leftarrow$).

- Assume that G is a connected graph with n vertices and $n-1$ edges.
- Then, as we proved before, G contains a spanning tree T.
- By the first part of the proof, T contains exactly $n-1$ edges.
- T is a subgraph of G, and it has the same number of edges as G.
- Hence, T and G are the same.
- In particular, G is a tree.

$\square$

Paths in trees

Paths in trees

Since a tree is a connected graph, between any two vertices in a tree there is a path. Can there be more than one path between two vertices in a tree?

Paths in trees

Since a tree is a connected graph, between any two vertices in a tree there is a path. Can there be more than one path between two vertices in a tree?

Lemma

Let T be a tree and $u, v \in V(T)$ with $u \neq v$.
Then there is a unique path in T between u and v.

Paths in trees

Since a tree is a connected graph, between any two vertices in a tree there is a path. Can there be more than one path between two vertices in a tree?

Lemma

Let T be a tree and $u, v \in V(T)$ with $u \neq v$.
Then there is a unique path in T between u and v.

Proof.

By contradiction.

- There is a path between u and v in T, since T is connected.
- Suppose there are two paths P and Q in T between u and v, and derive a contradiction.

Paths in trees

Since a tree is a connected graph, between any two vertices in a tree there is a path. Can there be more than one path between two vertices in a tree?

Lemma

Let T be a tree and $u, v \in V(T)$ with $u \neq v$.
Then there is a unique path in T between u and v.

Proof.

By contradiction.

- There is a path between u and v in T, since T is connected.
- Suppose there are two paths P and Q in T between u and v, and derive a contradiction.
- Let x and y in $V(T)$ be distinct and chosen in such a way that x and y are on both P and Q, but between x and y the vertices on P and Q are disjoint. (It is possible that $x = u$ and $y = v$, but this is not necessarily the case.)

Paths in trees

Since a tree is a connected graph, between any two vertices in a tree there is a path. Can there be more than one path between two vertices in a tree?

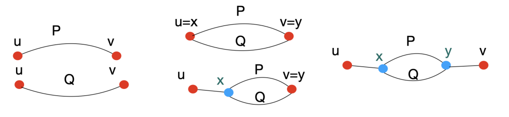
Lemma

Let T be a tree and $u, v \in V(T)$ with $u \neq v$.
Then there is a unique path in T between u and v.

Proof.

By contradiction.

- There is a path between u and v in T, since T is connected.
- Suppose there are two paths P and Q in T between u and v, and derive a contradiction.
- Let x and y in $V(T)$ be distinct and chosen in such a way that x and y are on both P and Q, but between x and y the vertices on P and Q are disjoint. (It is possible that $x = u$ and $y = v$, but this is not necessarily the case.)



Paths in trees

Since a tree is a connected graph, between any two vertices in a tree there is a path. Can there be more than one path between two vertices in a tree?

Lemma

Let T be a tree and $u, v \in V(T)$ with $u \neq v$.
Then there is a unique path in T between u and v.

Proof.

By contradiction.

- There is a path between u and v in T, since T is connected.
- Suppose there are two paths P and Q in T between u and v, and derive a contradiction.
- Let x and y in $V(T)$ be distinct and chosen in such a way that x and y are on both P and Q, but between x and y the vertices on P and Q are disjoint. (It is possible that $x = u$ and $y = v$, but this is not necessarily the case.)
- Then the segments of P and Q between x and y together form a cycle.
- This contradicts that T is a tree. Hence there is a unique (u, v)-path in T.

$\square$

Exercises

We have shown that, for a graph G on n vertices, the following conditions are equivalent:

1. G is tree;
2. G is connected and has $n-1$ edges.

Exercise 2: Show that these conditions are also equivalent to each of the following:

3. G is acyclic and has $n-1$ edges;
4. any two distinct vertices of G are connected by a unique path;
5. for any distinct $u, v \in V$, if $uv \notin E(G)$ then the graph $G + uv$ contains a unique cycle.

Rooted trees

Rooted trees

Definition

A (directed) rooted tree is a tree in which one vertex is fixed as the root (vertex) (and every edge is directed away from this root).

Rooted trees

Definition

A (directed) rooted tree is a tree in which one vertex is fixed as the root (vertex) (and every edge is directed away from this root).

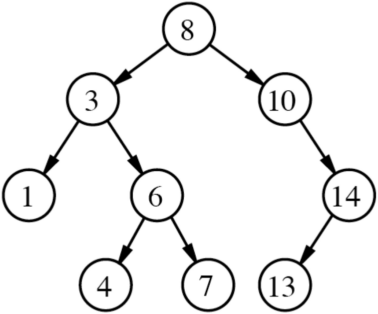
We usually draw a rooted tree in (horizontal) levels, starting with the root (level 0), then the neighbours of the root (level 1), etc.

Rooted trees

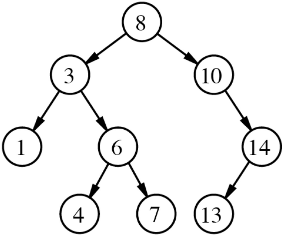
Definition

A (directed) rooted tree is a tree in which one vertex is fixed as the root (vertex) (and every edge is directed away from this root).

We usually draw a rooted tree in (horizontal) levels, starting with the root (level 0), then the neighbours of the root (level 1), etc.



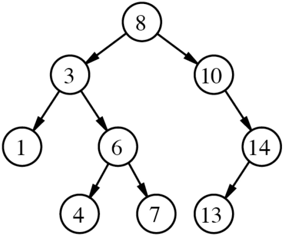
Rooted trees, children and parents



Definitions

Let v be a vertex in a rooted tree T.

Rooted trees, children and parents

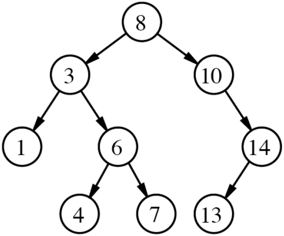


Definitions

Let v be a vertex in a rooted tree T.

- The neighbours of v in the next level are called the children of v.

Rooted trees, children and parents

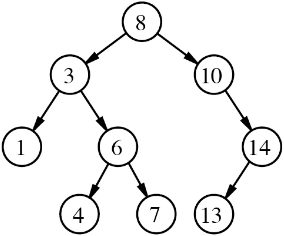


Definitions

Let v be a vertex in a rooted tree T.

- The neighbours of v in the next level are called the children of v.
- the (unique) neighbour of v in the previous level (if v is not the root) is called the parent of v.

Rooted trees, children and parents

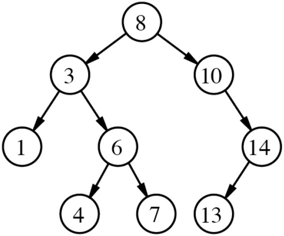


Definitions

Let v be a vertex in a rooted tree T.

- The neighbours of v in the next level are called the children of v.
- the (unique) neighbour of v in the previous level (if v is not the root) is called the parent of v.
- If v has no children then it is called a leaf of T;

Rooted trees, children and parents



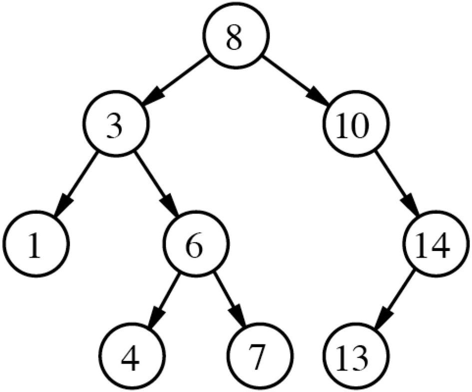
Definitions

Let v be a vertex in a rooted tree T.

- The neighbours of v in the next level are called the children of v.
- the (unique) neighbour of v in the previous level (if v is not the root) is called the parent of v.
- If v has no children then it is called a leaf of T;
- If v has children, then it is an internal vertex.

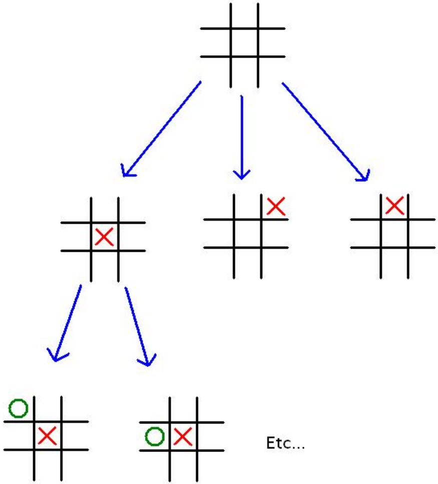
Some applications of trees

- Binary search trees (we have seen these earlier in ADS)



Some applications of trees

- Search trees (more on this in AI Search)



Some applications of trees

- Phylogenetic trees (Bioinformatics)

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.

$\square$

Graph isomorphism

Definition

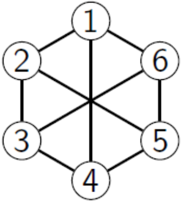
Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijective function $f : V \to V'$ such that for every $u, v \in V$ we have: $uv \in E$ if and only if $f(u)f(v) \in E'$. Then we write: $G \cong G'$.

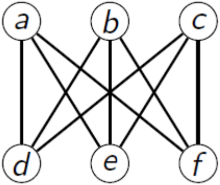
Graph isomorphism

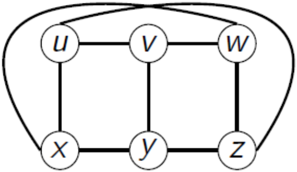
Definition

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?





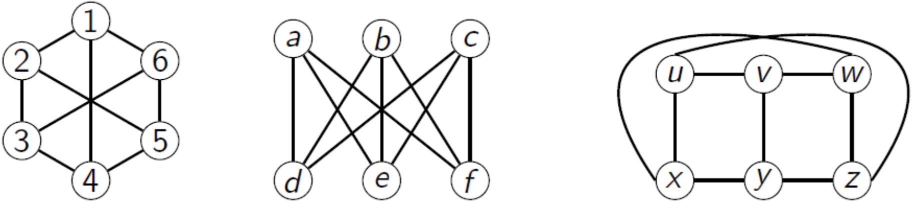


Graph isomorphism

Definition

Two graphs $G = (V, E)$ and $G' = (V', E')$ are isomorphic if there exists a bijective function $f : V \rightarrow V'$ 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?



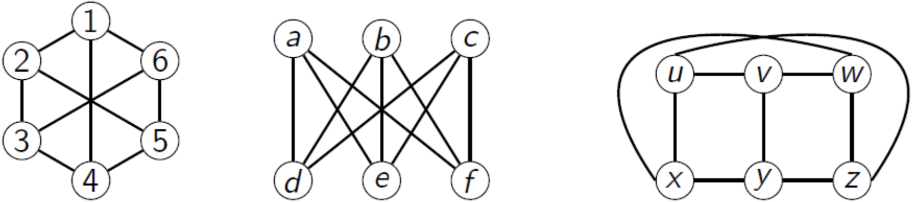
bijective function f for the first and second graph:
$1 \mapsto a,\ 2 \mapsto d,\ 3 \mapsto b,\ 4 \mapsto e,\ 5 \mapsto c,\ 6 \mapsto f$

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 $f : V \rightarrow V'$ 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.:

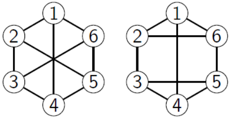
- number of vertices and edges, degree sequence, (strong) connectivity

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:

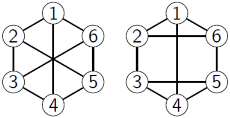


Graph isomorphism

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:

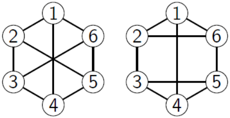
- although same number of vertices / edges, degree sequence

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:

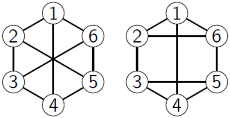
- although same number of vertices / edges, degree sequence
- Euler circuit, Hamiltonian Circuit:

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:

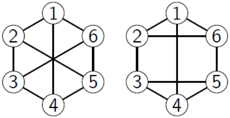
- although same number of vertices / edges, degree sequence
- Euler circuit, Hamiltonian Circuit: yes

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:

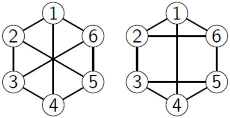
- although same number of vertices / edges, degree sequence
- Euler circuit, Hamiltonian Circuit: yes
- chromatic number, size of largest independent set:

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:

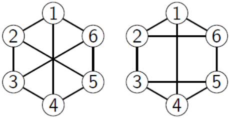
- although same number of vertices / edges, degree sequence
- Euler circuit, Hamiltonian Circuit: yes
- chromatic number, size of largest independent set: no

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:

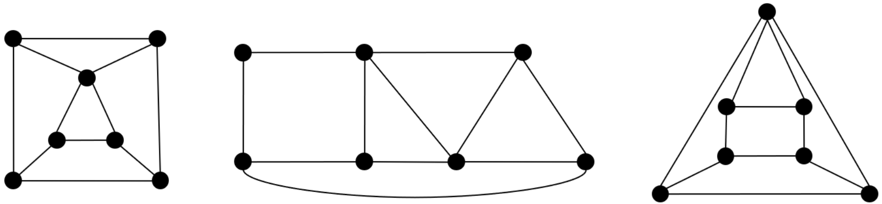
- although same number of vertices / edges, degree sequence
- Euler circuit, Hamiltonian Circuit: yes
- chromatic number, size of largest independent set: no

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 directed 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

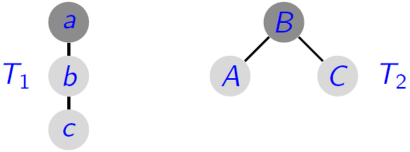
However, undirected rooted trees exist also:

- 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.

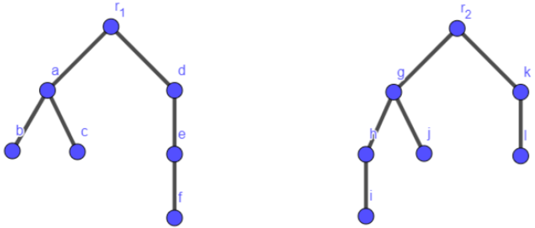
Exercise: Give an example of two rooted trees with the same number of levels, same number of vertices per level, and degree sequence per level, but are not isomorphic to each other

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 LabelVertex(T, v)

```
if v is a leaf of T then
    label(v) ← "10"
else
    for every child w of v do
        ℓ(w) ← LabelVertex(T, w)
    Sort the labels of the children of v decreasingly: ℓ(w_1) ≥ ℓ(w_2) ≥ ... ≥ ℓ(w_k)
    label(v) ← "1" ℓ(w_1) ℓ(w_2) ... ℓ(w_k) "0"
return label(v)
```

Graph isomorphism on rooted trees

An isomorphism algorithm for rooted trees (Algorithm 2):

Algorithm 1 LabelVertex(T, v)

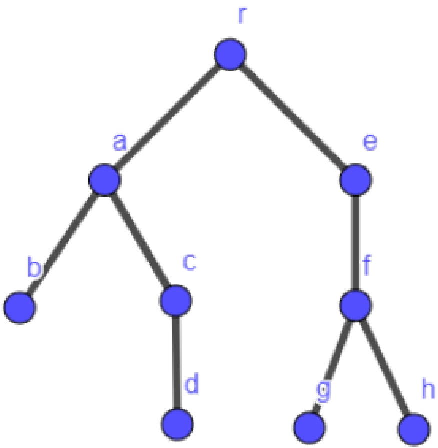
if v is a leaf of T **then**
    $label(v) \leftarrow$ "10"
else
    for every child w of v **do**
        $\ell(w) \leftarrow$ LabelVertex(T, w)
    Sort the labels of the children of v decreasingly: $\ell(w_1) \geq \ell(w_2) \geq \ldots \geq \ell(w_k)$
    $label(v) \leftarrow$ "1" $\ell(w_1)\ \ell(w_2)\ \ldots \ell(w_k)$ "0"
return $label(v)$

Algorithm 2 Labeled Tree Isomorphism$((T_1, r_1), (T_2, r_2))$

$label(r_1) \leftarrow$ LabelVertex(T_1, r_1)
$label(r_2) \leftarrow$ LabelVertex(T_2, r_2)
if $label(r_1) = label(r_2)$ **then**
    return YES
else
    return NO

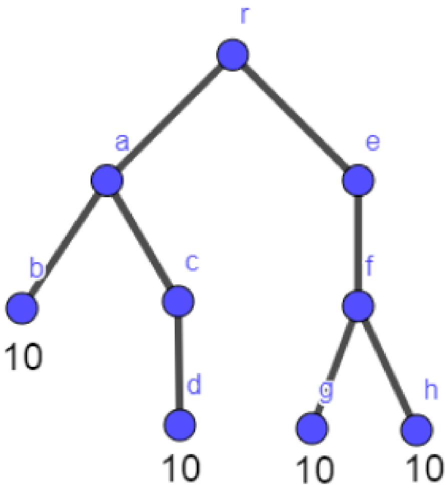
Graph isomorphism on rooted trees

Example:



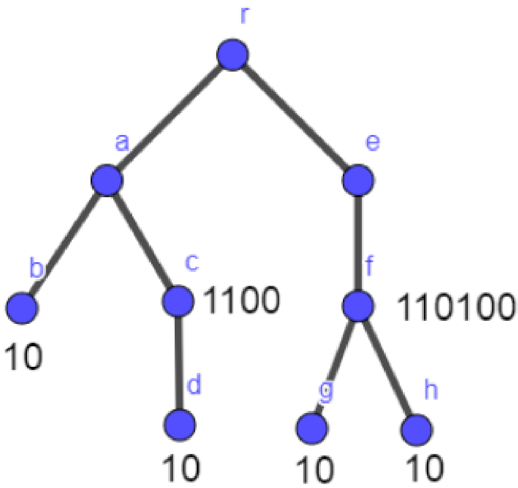
Graph isomorphism on rooted trees

Example:



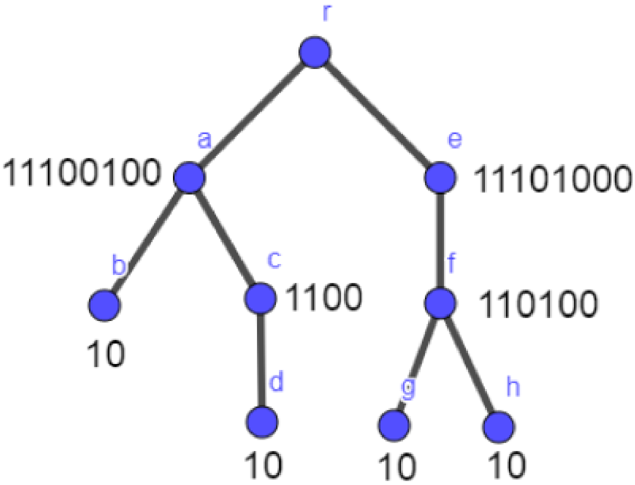
Graph isomorphism on rooted trees

Example:



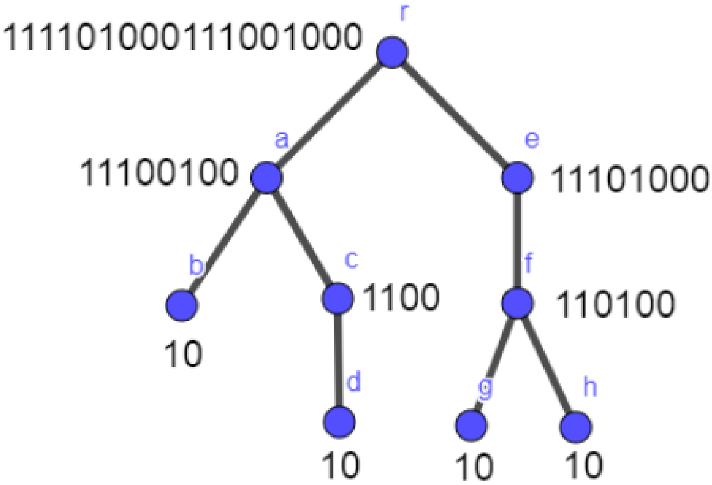
Graph isomorphism on rooted trees

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 $r_2 = f(r_1)$ 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
- $a_2 = f(a_1)$

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 $\ell(w_1) \geq \ldots \geq \ell(w_k)$,
and then it sets $label(r_1) =$ "1" $\ell(w_1)\ell(w_2)\ldots\ell(w_p)$ "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 $\ell(w_1) \geq \ldots \geq \ell(w_k)$,
and then it sets $label(r_1) =$ "1" $\ell(w_1)\ell(w_2)\ldots\ell(w_p)$ "0".

Exactly the same happens for $label(r_2)$,
i.e. $label(r_2) =$ "1" $\ell'(w_1)\ell'(w_2)\ldots\ell'(w_k)$ "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 $\ell(w_1) \geq \ldots \geq \ell(w_k)$,
and then it sets $label(r_1) =$ "1" $\ell(w_1)\ell(w_2)\ldots\ell(w_p)$ "0".

Exactly the same happens for $label(r_2)$,
i.e. $label(r_2) =$ "1" $\ell'(w_1)\ell'(w_2)\ldots\ell'(w_k)$ "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 $\ell(w_1) \geq \ldots \geq \ell(w_k)$,
and then it sets $label(r_1) =$ "1" $\ell(w_1)\ell(w_2)\ldots\ell(w_p)$ "0".

Exactly the same happens for $label(r_2)$,
i.e. $label(r_2) =$ "1" $\ell'(w_1)\ell'(w_2)\ldots\ell'(w_k)$ "0".

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

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 $\ell(w_1) \geq \ldots \geq \ell(w_k)$,
and then it sets $label(r_1) =$ "1" $\ell(w_1)\ell(w_2)\ldots\ell(w_p)$ "0".

Exactly the same happens for $label(r_2)$,
i.e. $label(r_2) =$ "1" $\ell'(w_1)\ell'(w_2)\ldots\ell'(w_k)$ "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)