Summary last week

- lexicographic product of partial orders
- inductively defined structures as least set satisfying clauses
- structural induction as induction principle w.r.t. sub-structure relation
- proof by counterexample minimal w.r.t. some well-founded order

Summary last week

- lexicographic product of partial orders
- inductively defined structures as least set satisfying clauses
- structural induction as induction principle w.r.t. sub-structure relation
- proof by counterexample minimal w.r.t. some well-founded order
- relation operations: identity, converse, intersection, union, composition, product
- preservation of property P by n-ary operation f: $P(f(x1, \ldots, x_n))$, if $P(x_1), \ldots, P(x_n)$
- being a function preserved: identity, composition, product
- being a partial order preserved: identity, converse, intersection, (lex) product
- being well-founded preserved: intersection, (lex) product, comp. extension

Summary last week

- lexicographic product of partial orders
- inductively defined structures as least set satisfying clauses
- structural induction as induction principle w.r.t. sub-structure relation
- proof by counterexample minimal w.r.t. some well-founded order

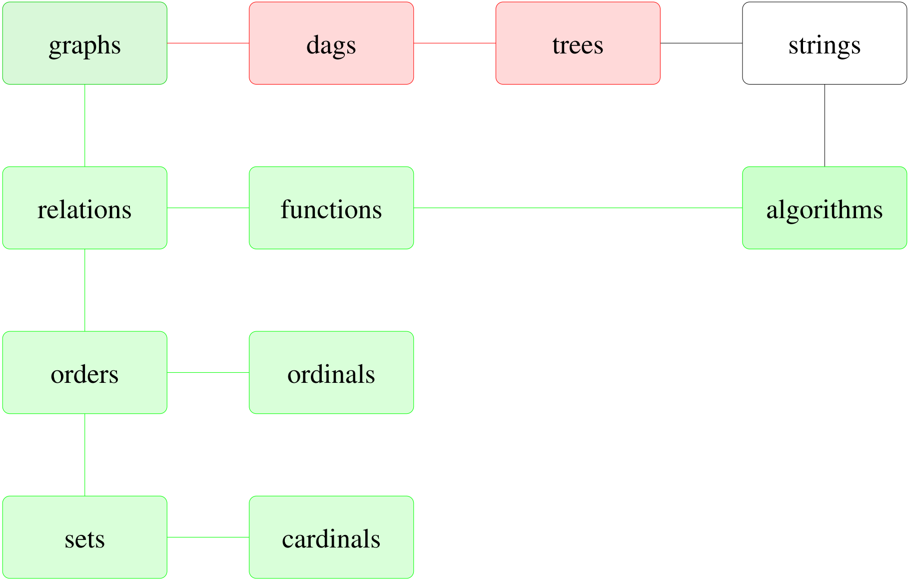
- relation operations: identity, converse, intersection, union, composition, product
- preservation of property P by n-ary operation f: $P(f(x1,\ldots,x_n))$, if $P(x_1),\ldots,P(x_n)$
- being a function preserved: identity, composition, product
- being a partial order preserved: identity, converse, intersection, (lex) product
- being well-founded preserved: intersection, (lex) product, comp. extension

- well-founded /well-orders as well-founded partial/total orders
- counting by cardinals, sets w.r.t. bijection; equinumerous
- counting by ordinals, well-orders w.r.t. isomorphism; order-preserving bijection

Course themes

- directed and undirected graphs
- relations and functions
- orders and induction
- trees and dags
- finite and infinite counting
- elementary number theory
- Turing machines, algorithms, and complexity
- decidable and undecidable problem

Discrete structures



Dags and trees motivation

Example (Dags)

- resource dependencies (build, citation)
- statement dependencies (out-of-order execution)
- sub-expression sharing (call-by-need)
- binary decision diagrams
- ...

Dags and trees motivation

Example (Dags)

- resource dependencies (build, citation)
- statement dependencies (out-of-order execution)
- sub-expression sharing (call-by-need)
- binary decision diagrams
- . . .

Example (Trees)

- data structures (searching, sorting, XML)
- parse tree (of text)/abstract syntax tree (of program)
- spanning tree (of graph)
- computation tree (of non-deterministic machines)
- . . .

Dags and trees

Definition (Cycle)

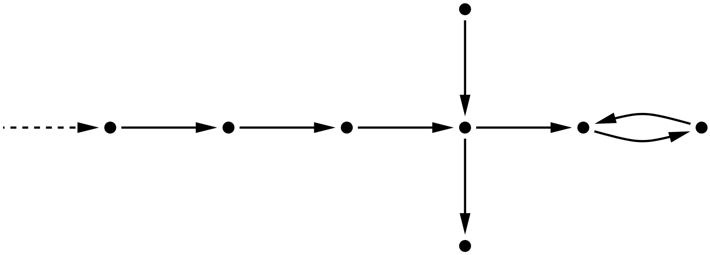
Let (V, E, src, tgt) be a directed multigraph

- a path is closed if its source is its target
- a non-empty closed path without repeated edges is a cycle
- directed multigraphs without cycles are cycle-free

Definition (Dags, forests and (rooted) trees)

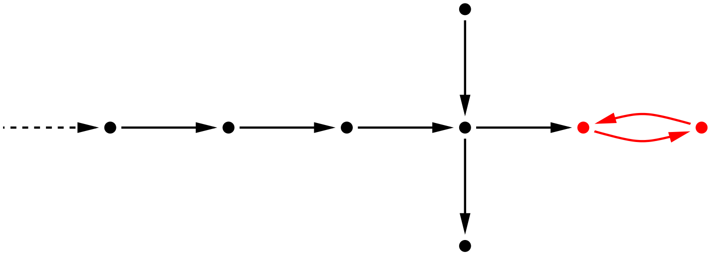
- a dag is a directed acyclic graph
- a forest is a dag with nodes of in-degree ≤ 1
- in a forest, nodes with out-degree 0 are called leaves
- a tree is a forest where all v_1, v_2 have a common ancestor v having paths to both
- a rooted tree is a tree with a node, the root, having a path to all nodes

Dags and trees example



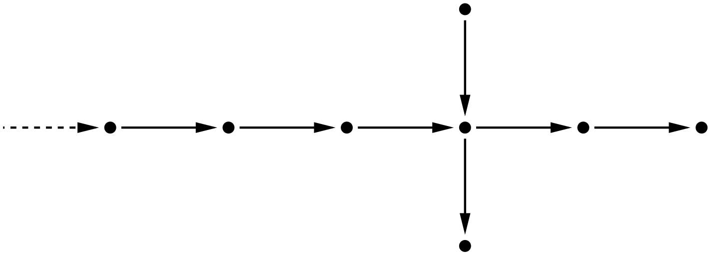
graph

Dags and trees example



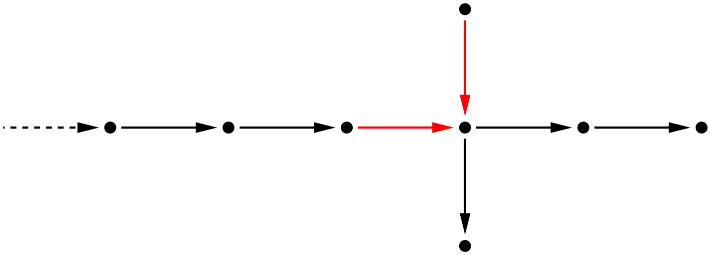
graph but not a dag (cycle)

Dags and trees example



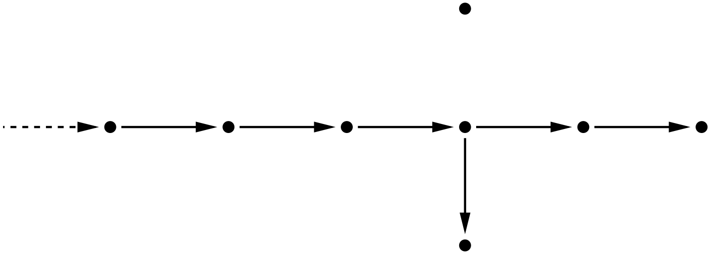
dag

Dags and trees example



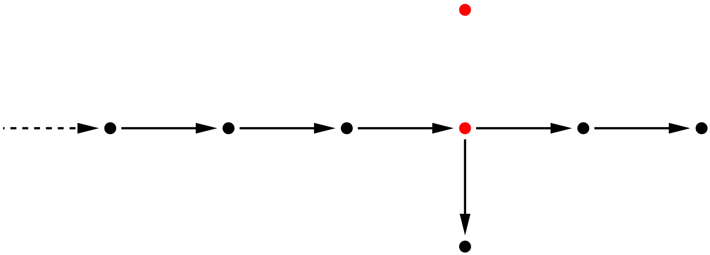
dag but not a forest (indegree 2)

Dags and trees example



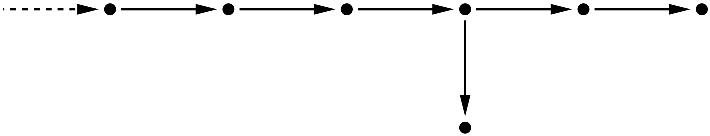
forest

Dags and trees example



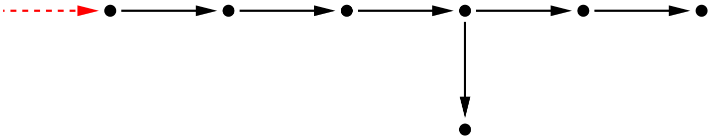
forest but not a tree (no common ancestor)

Dags and trees example



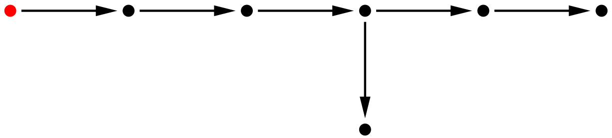
tree

Dags and trees example



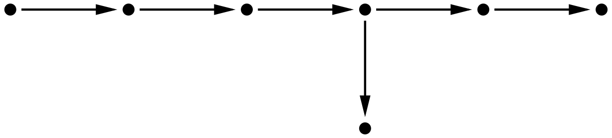
tree but not a rooted tree (no root)

Dags and trees example



rooted tree (root)

Dags and trees example



rooted tree

Simplicity in cycles

Lemma

simple paths do not have repeated edges.

Proof.

Let p be a simple path. if some edge e were to occur twice in it, the source node v of both occurrences of e would occur twice as well. ∎

Simplicity in cycles

Lemma

simple paths do not have repeated edges.

Proof.

Let p be a simple path. if some edge e were to occur twice in it, the source node v of both occurrences of e would occur twice as well. ∎

Corollary

every simple closed path in a multigraph is a cycle

Simplicity in cycles

Lemma

simple paths do not have repeated edges.

Proof.

Let p be a simple path. if some edge e were to occur twice in it, the source node v of both occurrences of e would occur twice as well. ■

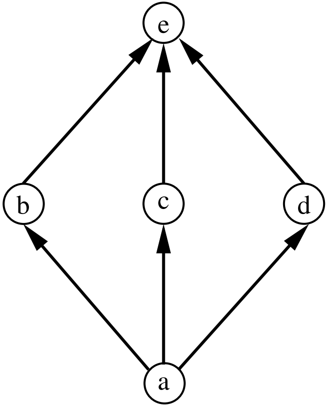
Corollary

every simple closed path in a multigraph is a cycle

Remark

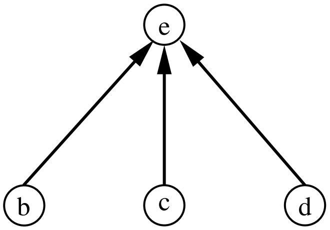
Since paths may be shortened to simple paths, cycles represent closed paths.

Topological sorting example



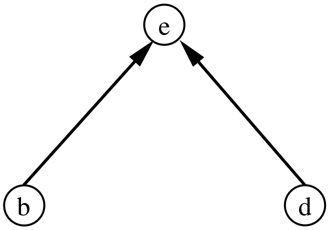
topological sorting: ()

Topological sorting example



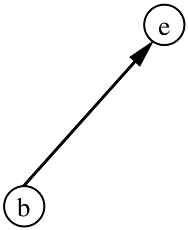
topological sorting: (a)

Topological sorting example



topological sorting: (a, c)

Topological sorting example



topological sorting: (a, c, d)

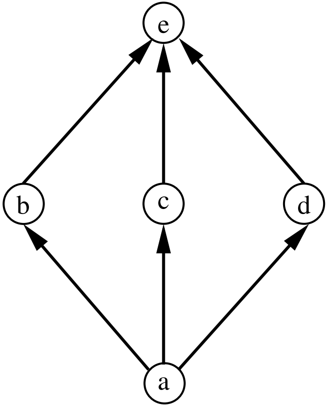
Topological sorting example

topological sorting: (a, c, d, b)

Topological sorting example

topological sorting: (a, c, d, b, e)

Topological sorting example



topological sorting: (a, c, d, b, e)

others: (a, c, b, d, e), (a, b, c, d, e), (a, b, d, c, e), (a, d, b, c, e), (a, d, c, b, e)

Topological sorting

Definition

a list $(a_0, \ldots, a_{n-1})$ is topologically $\leq$-sorted for partial order $\leq$, if $a_i < a_j$ implies $i < j$

Remark

if $\leq$ is a total order, then topologically $\leq$-sorted iff globally $\leq$-sorted

Topological sorting

Definition

a list $(a_0, \ldots, a_{n-1})$ is topologically $\leq$-sorted for partial order $\leq$, if $a_i < a_j$ implies $i < j$

Lemma

a finite set A can be topologically $\leq$-sorted by repeatedly removing minimal elements

Topological sorting

Definition

a list $(a_0, \ldots, a_{n-1})$ is topologically $\leq$-sorted for partial order $\leq$, if $a_i < a_j$ implies $i < j$

Lemma

a finite set A can be topologically $\leq$-sorted by repeatedly removing minimal elements

Proof.

if $l = (a_0, \ldots, a_{n-1})$ is topologically sorted and no element of A is smaller than any element of l, then so are $l' = (a_0, \ldots, a_{n-1}, a_n)$ and $A - \{a_n\}$ for a_n minimal in A: l' is topologically sorted since l is, and $a_n \not< a_j$ since no element of A is smaller than any element of l, and if $a_i < a_n$ then $i < n$ because $0 \leq i < n$ is an index in l. ■

Topological sorting

Definition

a list $(a_0, \ldots, a_{n-1})$ is topologically $\leq$-sorted for partial order $\leq$, if $a_i < a_j$ implies $i < j$

Lemma

a finite set A can be topologically $\leq$-sorted by repeatedly removing minimal elements

Lemma

if G is a dag, then $\leq_G$ defined by $v \leq_G v'$ if there is a path from v to v', is a partial order.

Topological sorting

Definition

a list $(a_0, \ldots, a_{n-1})$ is topologically $\leq$-sorted for partial order $\leq$, if $a_i < a_j$ implies $i < j$

Lemma

a finite set A can be topologically $\leq$-sorted by repeatedly removing minimal elements

Lemma

if G is a dag, then $\leq_G$ defined by $v \leq_G v'$ if there is a path from v to v', is a partial order.

Proof.

reflexivity and transitivity hold by empty paths and composing paths. To see anti-symmetry consider paths from v to v' and from v' to v. Both must be empty as otherwise their composition would yield a cycle in G, hence $v = v'$. ∎

Topological sorting

Definition

a list $(a_0, \ldots, a_{n-1})$ is topologically $\leq$-sorted for partial order $\leq$, if $a_i < a_j$ implies $i < j$

Lemma

a finite set A can be topologically $\leq$-sorted by repeatedly removing minimal elements

Lemma

if G is a dag, then $\leq_G$ defined by $v \leq_G v'$ if there is a path from v to v', is a partial order.

Corollary

every finite dag G can be topologically $\leq_G$-sorted.

Shortest paths in dags

Lemma

in a finite dag, shortest and longest paths can be computed in $O(n)$

Shortest paths in dags

Lemma

in a finite dag, shortest and longest paths can be computed in $O(n)$

Proof.

shortest path adapting topological sorting: let G be weighted graph with nodes v, v'.

1. initialise v with distance 0
2. while G is non-empty
   a) set w to a minimal node having some distance (no edges from other such to w), say d
   b) if $w = v'$ return d
   c) for each edge $e : w \to_k w'$ set the distance d' of w' to $\min(d', d + k)$.
   d) remove w and all edges from it, from G
3. return ∞

∎

Facts on trees

Lemma

every finite tree is a rooted tree.

Facts on trees

Lemma

every finite tree is a rooted tree.

Proof.

let G be a finite tree having, say, n nodes $\{v_1, \ldots, v_n\}$. Setting $v'_1 = v_1$ and v'_{i+1} to be a common ancestor of v'_i and v_{i+1}, we obtain that v'_n is a common ancestor of all nodes. Therefore, v'_n is the root. ■ ■

Lemma (Characterising forests and rooted trees)

1. *in a forest there is at most one path from a node v to a node v'*

Proof.

Lemma (Characterising forests and rooted trees)

1. *in a forest there is at most one path from a node v to a node v'*
2. *a multigraph is a rooted tree iff there is a node v with a unique path to each node*

Proof.

Lemma (Characterising forests and rooted trees)

1. *in a forest there is at most one path from a node v to a node v'*
2. *a multigraph is a rooted tree iff there is a node v with a unique path to each node*

Proof.

1. For a proof by contradiction, suppose there were two paths from v to v'. If $v = v'$, then one of them would be a cycle, contradicting acyclicity. If $v \neq v'$ let $e \neq f$ be the last edges where the paths differ, starting comparing from v'. By being the last such, e and f must have the same target, contradicting in-degree ≤ 1.

Lemma (Characterising forests and rooted trees)

1. *in a forest there is at most one path from a node v to a node v'*
2. *a multigraph is a rooted tree iff there is a node v with a unique path to each node*

Proof.

1. For a proof by contradiction, suppose there were two paths from v to v'. If $v = v'$, then one of them would be a cycle, contradicting acyclicity. If $v \neq v'$ let $e \neq f$ be the last edges where the paths differ, starting comparing from v'. By being the last such, e and f must have the same target, contradicting in-degree ≤ 1.
2. (only–if) By the definition of rooted tree and the previous item.

   (if) Uniqueness of paths entails the multigraph can have neither parallel edges nor cycles, so is a dag. If there were edges $e \neq f$ with the same target v', then there would be distinct paths from v to v' via the respective sources of e and f, which cannot be, so in-degree ≤ 1 and we have a forest. Taking v as root shows the forest is a rooted tree. ∎

The number of edges and vertices in a tree

Lemma

The number of vertices in a finite tree is the number of edges $+1$

The number of edges and vertices in a tree

Lemma

The number of vertices in a finite tree is the number of edges $+1$

Proof.

Since the tree is finite, it has some root v. Consider the relation R relating every vertex v' to the last edge on a path from v to v'.

The number of edges and vertices in a tree

Lemma

The number of vertices in a finite tree is the number of edges $+1$

Proof.

Since the tree is finite, it has some root v. Consider the relation R relating every vertex v' to the last edge on a path from v to v'.

- R is a function from $V - \{v\}$ to E, since for each node $v' \neq v$ there is a unique non-empty path from the root v to v'.

The number of edges and vertices in a tree

Lemma

The number of vertices in a finite tree is the number of edges $+1$

Proof.

Since the tree is finite, it has some root v. Consider the relation R relating every vertex v' to the last edge on a path from v to v'.

- R is a function from $V - \{v\}$ to E, since for each node $v' \neq v$ there is a unique non-empty path from the root v to v'.
- R^{-1} relates edges to their targets. It is a function from E to $V - \{v\}$, since any edge, say from v' to v'' is the last edge of the unique path from v to v' to v'', and its target v'' is distinct from the root (otherwise there would be a cycle).

The number of edges and vertices in a tree

Lemma

The number of vertices in a finite tree is the number of edges $+1$

Proof.

Since the tree is finite, it has some root v. Consider the relation R relating every vertex v' to the last edge on a path from v to v'.

- R is a function from $V - \{v\}$ to E, since for each node $v' \neq v$ there is a unique non-empty path from the root v to v'.
- R^{-1} relates edges to their targets. It is a function from E to $V - \{v\}$, since any edge, say from v' to v'' is the last edge of the unique path from v to v' to v'', and its target v'' is distinct from the root (otherwise there would be a cycle).

We obtain R is bijection, hence $V - \{v\}$ and E are equinumerous. ∎

Definition (undirected multigraph)

An undirected multigraph is given by

- a set of nodes or vertices V
- a set of edges E
- a map $r\colon E \to \{\{c,d\} \mid c,d \in V\}$ with $e \mapsto r(e)$, that maps every edge e to a set $r(e)$ having one or two elements, its endpoints.
- e is an edge between, joining or incident on its endpoints

Definition (undirected multigraph)

An undirected multigraph is given by

- a set of nodes or vertices V
- a set of edges E
- a map $r\colon E \to \{\{c,d\} \mid c,d \in V\}$ with $e \mapsto r(e)$, that maps every edge e to a set $r(e)$ having one or two elements, its endpoints.
- e is an edge between, joining or incident on its endpoints

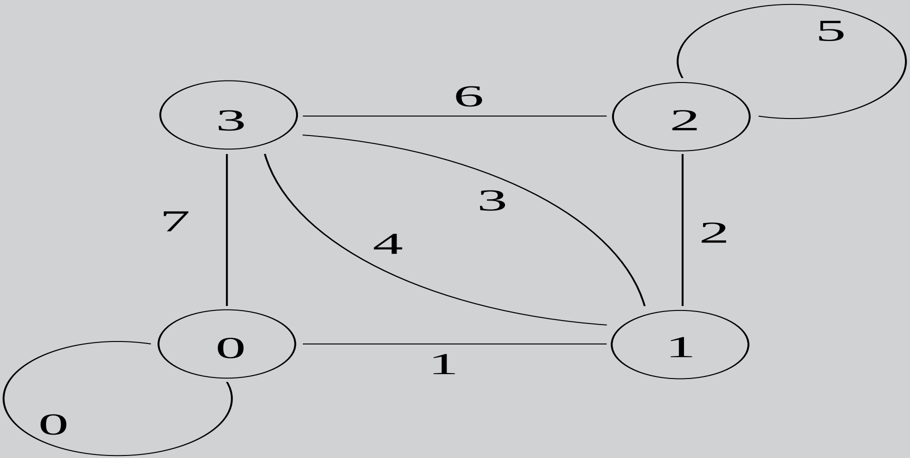
Example

Let $V = \{0,1,2,3\}$, $E = \{0,1,2,\dots,7\}$ and the function r be defined by

e	$r(e)$
0	$\{0\}$
1	$\{0,1\}$
2	$\{1,2\}$
3	$\{1,3\}$

e	$r(e)$
4	$\{1,3\}$
5	$\{2\}$
6	$\{2,3\}$
7	$\{0,3\}$

Example (Continued)



From directed to undirected multigraphs, and back

Definition

To a directed multigraph an undirected multigraph can be associated by forgetting the directions of edges, defining the set of end-points of an edge e to comprise its source and target: $r(e) = \{src(e), tgt(e)\}$.

From directed to undirected multigraphs, and back

Definition

To a directed multigraph an undirected multigraph can be associated by forgetting the directions of edges, defining the set of end-points of an edge e to comprise its source and target: $r(e) = \{src(e), tgt(e)\}$.

Definition

To an undirected multigraph a directed multigraph can be associated by duplicating each edge e into e_l and e_r directed to the left resp. right, i.e. if $r(e) = \{c, d\}$ then e_l is from d to c, and e_r from c to d.

From directed to undirected multigraphs, and back

Definition

To a directed multigraph an undirected multigraph can be associated by forgetting the directions of edges, defining the set of end-points of an edge e to comprise its source and target: $r(e) = \{src(e), tgt(e)\}$.

Definition

To an undirected multigraph a directed multigraph can be associated by duplicating each edge e into e_l and e_r directed to the left resp. right, i.e. if $r(e) = \{c, d\}$ then e_l is from d to c, and e_r from c to d.

Remark

Not inverse to each other, but often preserve properties. For instance, there being a path between two nodes.

Definition

- vertex c is a neighbour of the vertex d, if there exists an edge joining both
- an edge having only one endpoint is a loop
- two edges having the same endpoints are parallel
- the degree of a vertex v is the number of edges having v as endpoint
- a multigraph is vertex- resp. edge-labelled, if there is a function from V resp. E to a set of labels.
- if the labels are numbers, we speak of weights and weighted multigraphs

Definition

- vertex c is a neighbour of the vertex d, if there exists an edge joining both
- an edge having only one endpoint is a loop
- two edges having the same endpoints are parallel
- the degree of a vertex v is the number of edges having v as endpoint
- a multigraph is vertex- resp. edge-labelled, if there is a function from V resp. E to a set of labels.
- if the labels are numbers, we speak of weights and weighted multigraphs

Definition (undirected graph)

An undirected graph is an undirected multigraph without parallel edges: then there is for every set of nodes $\{c, d\}$ at most one edge joining c and d.

Definition

- Let $G = (V, E, r)$ be an undirected multigraph
- $G' = (V', E', r')$ is sub-multigraph of G, if $V' \subseteq V,\ E' \subseteq E$ and $r'(e) = r(e)$ for $e \in E'$
- A sub-graph is a sub-multigraph that itself is a graph

Definition

- Let $G = (V, E, r)$ be an undirected multigraph
- $G' = (V', E', r')$ is sub-multigraph of G, if $V' \subseteq V$, $E' \subseteq E$ and $r'(e) = r(e)$ for $e \in E'$
- A sub-graph is a sub-multigraph that itself is a graph

Definition

Let (V, E, r) be an undirected multigraph, and let c, d be vertices

- A tuple $(e_0, e_1, \ldots, e_{\ell-1}) \in E^\ell$ is a path from c to d of length ℓ, if there are vertices $v_0, v_1, \ldots, v_\ell$ with $v_0 = c$, $v_\ell = d$, and $r(e_i) = \{v_i, v_{i+1}\}$ for $i = 0, 1, \ldots, \ell - 1$
- v_0 is the initial or starting node; v_ℓ it its end-node
- $v_1, v_2, \ldots, v_{\ell-1}$ are the intermediate nodes
- For every node $v \in V$, the empty tuple $() \in E^0$ is the empty path with starting node v and end-node v

Definition (Continued)

- A sub-multigraph is connected, if there are paths between all its nodes

Definition (Continued)

- A sub-multigraph is connected, if there are paths between all its nodes
- A connected component is a maximal connected sub-multigraph

Definition (Continued)

- A sub-multigraph is connected, if there are paths between all its nodes
- A connected component is a maximal connected sub-multigraph
- Path is simple if non-empty and no repeated nodes (exception $v_0 = v_\ell$), edges.

Definition (Continued)

- A sub-multigraph is connected, if there are paths between all its nodes
- A connected component is a maximal connected sub-multigraph
- Path is simple if non-empty and no repeated nodes (exception $v_0 = v_\ell$), edges.
- For every path $(e_0, e_1, \ldots, e_{\ell-2}, e_{\ell-1})$ from c to d there is the inverse path $(e_{\ell-1}, e_{\ell-2}, \ldots, e_1, e_0)$ from d to c

Definition (Continued)

- A sub-multigraph is connected, if there are paths between all its nodes
- A connected component is a maximal connected sub-multigraph
- Path is simple if non-empty and no repeated nodes (exception $v_0 = v_\ell$), edges.
- For every path $(e_0, e_1, \ldots, e_{\ell-2}, e_{\ell-1})$ from c to d there is the inverse path $(e_{\ell-1}, e_{\ell-2}, \ldots, e_1, e_0)$ from d to c
- The concatenation or composition of the paths $(e_0, e_1, \ldots, e_{\ell-1})$ from c to d and $(f_0, f_1, \ldots, f_{m-1})$ from d to e is the path

$$(e_0, e_1, \ldots, e_{\ell-1}, f_0, f_1, \ldots, f_{m-1})$$

  from c to e

Definition (Continued)

- A sub-multigraph is connected, if there are paths between all its nodes
- A connected component is a maximal connected sub-multigraph
- Path is simple if non-empty and no repeated nodes (exception $v_0 = v_\ell$), edges.
- For every path $(e_0, e_1, \ldots, e_{\ell-2}, e_{\ell-1})$ from c to d there is the inverse path $(e_{\ell-1}, e_{\ell-2}, \ldots, e_1, e_0)$ from d to c
- The concatenation or composition of the paths $(e_0, e_1, \ldots, e_{\ell-1})$ from c to d and $(f_0, f_1, \ldots, f_{m-1})$ from d to e is the path

$$(e_0, e_1, \ldots, e_{\ell-1}, f_0, f_1, \ldots, f_{m-1})$$

  from c to e
- A path is closed, if its starting and end-nodes are the same

Definition (Continued)

- A sub-multigraph is connected, if there are paths between all its nodes
- A connected component is a maximal connected sub-multigraph
- Path is simple if non-empty and no repeated nodes (exception $v_0 = v_\ell$), edges.
- For every path $(e_0, e_1, \ldots, e_{\ell-2}, e_{\ell-1})$ from c to d there is the inverse path $(e_{\ell-1}, e_{\ell-2}, \ldots, e_1, e_0)$ from d to c
- The concatenation or composition of the paths $(e_0, e_1, \ldots, e_{\ell-1})$ from c to d and $(f_0, f_1, \ldots, f_{m-1})$ from d to e is the path

  $$(e_0, e_1, \ldots, e_{\ell-1}, f_0, f_1, \ldots, f_{m-1})$$

  from c to e
- A path is closed, if its starting and end-nodes are the same
- A cycle is a non-empty closed simple path

Definition (Continued)

- A sub-multigraph is connected, if there are paths between all its nodes
- A connected component is a maximal connected sub-multigraph
- Path is simple if non-empty and no repeated nodes (exception $v_0 = v_\ell$), edges.
- For every path $(e_0, e_1, \ldots, e_{\ell-2}, e_{\ell-1})$ from c to d there is the inverse path $(e_{\ell-1}, e_{\ell-2}, \ldots, e_1, e_0)$ from d to c
- The concatenation or composition of the paths $(e_0, e_1, \ldots, e_{\ell-1})$ from c to d and $(f_0, f_1, \ldots, f_{m-1})$ from d to e is the path

  $$(e_0, e_1, \ldots, e_{\ell-1}, f_0, f_1, \ldots, f_{m-1})$$

  from c to e
- A path is closed, if its starting and end-nodes are the same
- A cycle is a non-empty closed simple path
- undirected multigraphs without cycles are called cycle-free

Undirected forests and trees

Definition

- A forest is a cycle-free undirected multigraph
- A tree is a connected forest
- leaves are nodes with degree ≤ 1 in a forest

Undirected forests and trees

Definition

- A forest is a cycle-free undirected multigraph
- A tree is a connected forest
- leaves are nodes with degree ≤ 1 in a forest

Example

Undirected forests and trees

Definition

- A forest is a cycle-free undirected multigraph
- A tree is a connected forest
- leaves are nodes with degree ≤ 1 in a forest

Example

- In the multigraph in the first example there are the following paths from node 0 to node 3

$$(1,2,6),(1,2,5,6),(1,3),(1,4),(1,3,7,1,3),(1,4,7,1,3),(7)$$

Undirected forests and trees

Definition

- A forest is a cycle-free undirected multigraph
- A tree is a connected forest
- leaves are nodes with degree ≤ 1 in a forest

Example

- In the multigraph in the first example there are the following paths from node 0 to node 3

$$(1, 2, 6), (1, 2, 5, 6), (1, 3), (1, 4), (1, 3, 7, 1, 3), (1, 4, 7, 1, 3), (7)$$

- The multigraph is connected

Undirected forests and trees

Definition

- A forest is a cycle-free undirected multigraph
- A tree is a connected forest
- leaves are nodes with degree ≤ 1 in a forest

Example

- In the multigraph in the first example there are the following paths from node 0 to node 3

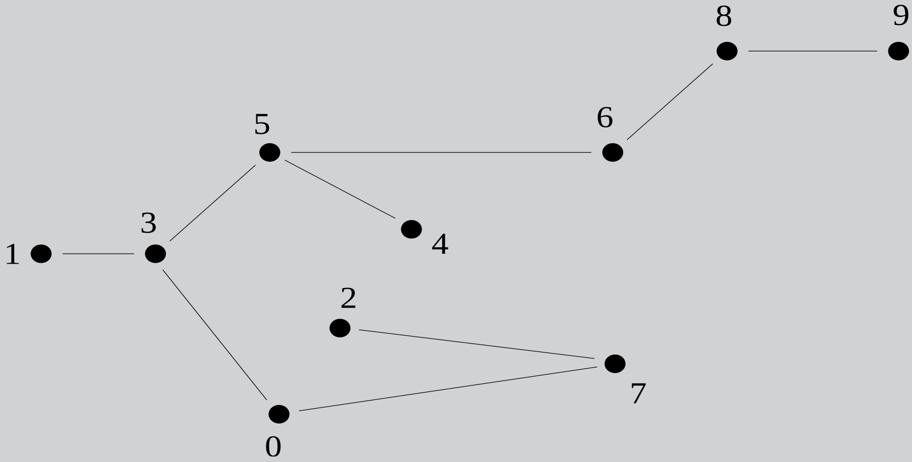
$$(1, 2, 6), (1, 2, 5, 6), (1, 3), (1, 4), (1, 3, 7, 1, 3), (1, 4, 7, 1, 3), (7)$$

- The multigraph is connected
- There are simple cycles with starting-node 0

$$(0), (1, 2, 6, 7, (1, 3, 7), (1, 4, 7), (7, 3, 1), (7, 4, 1), (7, 6, 2, 1)$$

Example

The following graph is a connected forest, and therefore a tree; its leaves are $1, 2, 4, 9$



Characterising undirected trees

Lemma

For an undirected multigraph G, the following are equivalent.

0) *G is connected but removing any edge makes the graph disconnected*

1) *in G there is a unique simple path between any two nodes*

2) *G is connected and acyclic but adding any edge makes the graph contain a cycle*

Proof.

0)⇒1) Suppose there were two paths between two nodes. W.l.o.g. we may assume these are of minimal (total) length. Then they do not have edges in common, so removing any edge on them the graph would remain connected. Contradiction ■

Characterising undirected trees

Lemma

For an undirected multigraph G, the following are equivalent.

0) *G is connected but removing any edge makes the graph disconnected*

1) *in G there is a unique simple path between any two nodes*

2) *G is connected and acyclic but adding any edge makes the graph contain a cycle*

Proof.

1)⇒2) By assumption, there is a unique path p between v and v'. Adding a fresh edge e between them, makes the concatenation of p and e into a cycle. ■

Characterising undirected trees

Lemma

For an undirected multigraph G, the following are equivalent.

0) *G is connected but removing any edge makes the graph disconnected*

1) *in G there is a unique simple path between any two nodes*

2) *G is connected and acyclic but adding any edge makes the graph contain a cycle*

Proof.

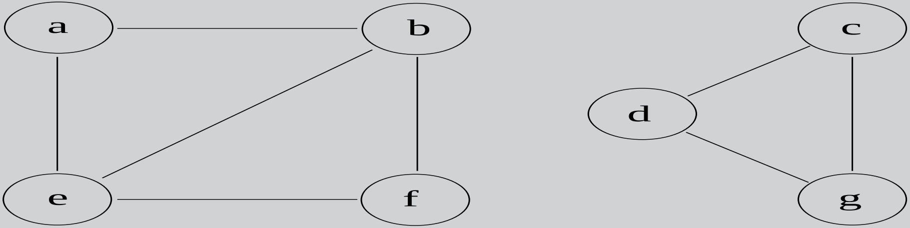
2)$\Rightarrow$0) If removing an edge e between v and v' from G would not affect being connected, there would be a path between v and v' in which e does not occur. But then the concatenation of e and p would be a cycle in G already. Contradiction. ∎

Definition

- Let G be an undirected multigraph
- A sub-graph G' of G is a spanning forest of G, if
  1. G is a forest, and
  2. the partitionings of G resp. G' into connected components are the same.
- Then $V' = V$

Example

The following graph has $8 \cdot 3 = 24$ spanning forests



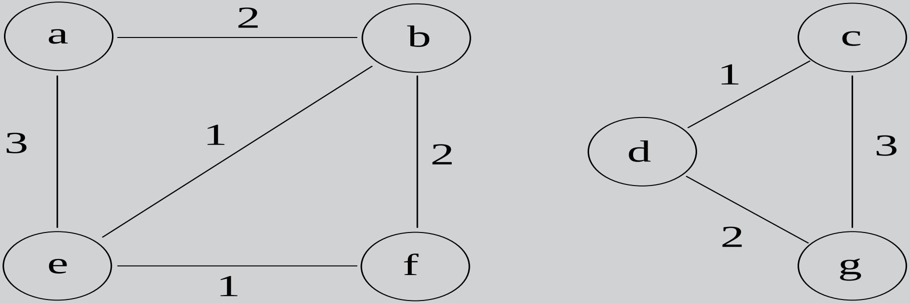
Theorem (Kruskal's algorithm)

1. *Let* $G = (V, E, r)$ *be an undirected multigraph with weights* b
2. *We want to construct a partitioning of* V *into connected components, and a set of edges* F *that constitutes a spanning forest of* G *having minimal weight* $\sum_{e \in F} b(e)$
3. *We preprocess* G *by removing all loops and all parallel edges except for a single one of least weight*
4. *The algorithm then proceeds as follows, with complexity* $\mathrm{O}(\#(V) \cdot \#(E))$

> *Set* $F = \varnothing$ *and* $P = \{\{v\} \mid v \in V\}$
> *For* i *from* 0 *to* $m - 1$ *repeat:*
>   *if the nodes* v *and* u *of* e_i *are in distinct blocks of* P*,*
>   *combine both blocks of* P *and adjoin* e_i *to* F

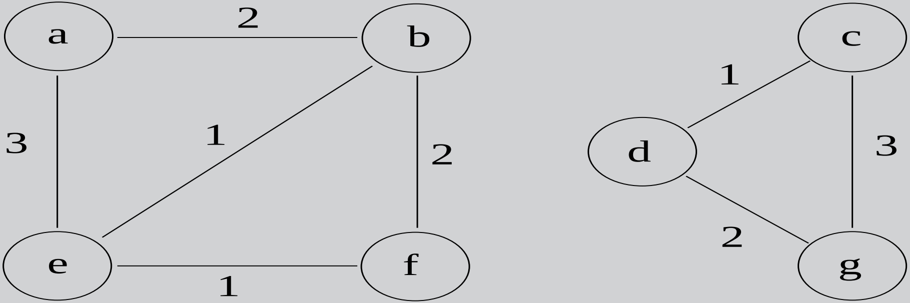
Example

For the weighted graph



Example

For the weighted graph



Kruskal's algorithm starts with $F = \varnothing$; $P = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}\}$ and terminates with

$$F = \{\{a, b\}, \{b, e\}, \{c, d\}, \{d, g\}, \{e, f\}\}$$
$$P = \{\{a, b, e, f\}, \{c, d, g\}\}$$

Proof.

- Let G_i be the sub-graph of G with V as nodes and edges $\{e_0, e_1, , \ldots, e_i\}$

Proof.

- Let G_i be the sub-graph of G with V as nodes and edges $\{e_0, e_1, , \dots, e_i\}$
- The algorithm starts with partitioning into singletons of nodes and then proceeds by combining blocks by edges connecting them

Proof.

- Let G_i be the sub-graph of G with V as nodes and edges $\{e_0, e_1, , \ldots, e_i\}$
- The algorithm starts with partitioning into singletons of nodes and then proceeds by combining blocks by edges connecting them
- After step i, P is a partitioning of G_i into connected components

Proof.

- Let G_i be the sub-graph of G with V as nodes and edges $\{e_0, e_1, , \ldots, e_i\}$
- The algorithm starts with partitioning into singletons of nodes and then proceeds by combining blocks by edges connecting them
- After step i, P is a partitioning of G_i into connected components
- Initially, the set F is empty, and in step i it is extended by a connecting edge whose endpoints are in the combined block.

Proof.

- Let G_i be the sub-graph of G with V as nodes and edges $\{e_0, e_1, , \ldots, e_i\}$
- The algorithm starts with partitioning into singletons of nodes and then proceeds by combining blocks by edges connecting them
- After step i, P is a partitioning of G_i into connected components
- Initially, the set F is empty, and in step i it is extended by a connecting edge whose endpoints are in the combined block.
- For every block B, the sub-graph restricted to nodes B and the corresponding edges in F, is a tree

Proof.

- Let G_i be the sub-graph of G with V as nodes and edges $\{e_0, e_1, , \ldots, e_i\}$
- The algorithm starts with partitioning into singletons of nodes and then proceeds by combining blocks by edges connecting them
- After step i, P is a partitioning of G_i into connected components
- Initially, the set F is empty, and in step i it is extended by a connecting edge whose endpoints are in the combined block.
- For every block B, the sub-graph restricted to nodes B and the corresponding edges in F, is a tree
- Therefore, after step i, the sub-graph having nodes V and edges F is a spanning forest of G_i

Proof.

- Let G_i be the sub-graph of G with V as nodes and edges $\{e_0, e_1, , \ldots, e_i\}$
- The algorithm starts with partitioning into singletons of nodes and then proceeds by combining blocks by edges connecting them
- After step i, P is a partitioning of G_i into connected components
- Initially, the set F is empty, and in step i it is extended by a connecting edge whose endpoints are in the combined block.
- For every block B, the sub-graph restricted to nodes B and the corresponding edges in F, is a tree
- Therefore, after step i, the sub-graph having nodes V and edges F is a spanning forest of G_i
- We show that the greedy strategy employed, yields a spanning forest of minimal weight

Proof (continued).

- Let F' be the set of edges of a spanning forest of minimal weight, and suppose $F' \neq F$; then there exists an edge e_i in F not in F'.

Proof (continued).

- Let F' be the set of edges of a spanning forest of minimal weight, and suppose $F' \neq F$; then there exists an edge e_i in F not in F'.
- Let v_1, v_2 be the endpoints of e_i and V_1, V_2 the corresponding blocks in the algorithm

Proof (continued).

- Let F' be the set of edges of a spanning forest of minimal weight, and suppose $F' \neq F$; then there exists an edge e_i in F not in F'.
- Let v_1, v_2 be the endpoints of e_i and V_1, V_2 the corresponding blocks in the algorithm
- Since there is a path p from v_1 to v_2 with edges in F', there exists an edge e_j in the path p having one endpoint in V_1 and the other endpoint not in it. Hence, $j > i$ and $b(e_j) \geq b(e_i)$.

Proof (continued).

- Let F' be the set of edges of a spanning forest of minimal weight, and suppose $F' \neq F$; then there exists an edge e_i in F not in F'.
- Let v_1, v_2 be the endpoints of e_i and V_1, V_2 the corresponding blocks in the algorithm
- Since there is a path p from v_1 to v_2 with edges in F', there exists an edge e_j in the path p having one endpoint in V_1 and the other endpoint not in it. Hence, $j > i$ and $b(e_j) \geq b(e_i)$.
- The sub-graph with nodes V and edges defined by $E' := (F' \setminus \{e_j\}) \cup \{e_i\}$ then is a spanning tree, because every path via e_j can be transformed into one via e_i and the other edges of p and vice versa; moreover that sub-graph has minimal weight.

Proof (continued).

- Let F' be the set of edges of a spanning forest of minimal weight, and suppose $F' \neq F$; then there exists an edge e_i in F not in F'.
- Let v_1, v_2 be the endpoints of e_i and V_1, V_2 the corresponding blocks in the algorithm
- Since there is a path p from v_1 to v_2 with edges in F', there exists an edge e_j in the path p having one endpoint in V_1 and the other endpoint not in it. Hence, $j > i$ and $b(e_j) \geq b(e_i)$.
- The sub-graph with nodes V and edges defined by $E' := (F' \setminus \{e_j\}) \cup \{e_i\}$ then is a spanning tree, because every path via e_j can be transformed into one via e_i and the other edges of p and vice versa; moreover that sub-graph has minimal weight.
- by finitely many such exchanges we obtain F from F'

Proof (continued).

- Let F' be the set of edges of a spanning forest of minimal weight, and suppose $F' \neq F$; then there exists an edge e_i in F not in F'.
- Let v_1, v_2 be the endpoints of e_i and V_1, V_2 the corresponding blocks in the algorithm
- Since there is a path p from v_1 to v_2 with edges in F', there exists an edge e_j in the path p having one endpoint in V_1 and the other endpoint not in it. Hence, $j > i$ and $b(e_j) \geq b(e_i)$.
- The sub-graph with nodes V and edges defined by $E' := (F' \setminus \{e_j\}) \cup \{e_i\}$ then is a spanning tree, because every path via e_j can be transformed into one via e_i and the other edges of p and vice versa; moreover that sub-graph has minimal weight.
- by finitely many such exchanges we obtain F from F'
- Since F' has a minimal weight, so has F

Proof (continued).

- Let F' be the set of edges of a spanning forest of minimal weight, and suppose $F' \neq F$; then there exists an edge e_i in F not in F'.
- Let v_1, v_2 be the endpoints of e_i and V_1, V_2 the corresponding blocks in the algorithm
- Since there is a path p from v_1 to v_2 with edges in F', there exists an edge e_j in the path p having one endpoint in V_1 and the other endpoint not in it. Hence, $j > i$ and $b(e_j) \geq b(e_i)$.
- The sub-graph with nodes V and edges defined by $E' := (F' \setminus \{e_j\}) \cup \{e_i\}$ then is a spanning tree, because every path via e_j can be transformed into one via e_i and the other edges of p and vice versa; moreover that sub-graph has minimal weight.
- by finitely many such exchanges we obtain F from F'
- Since F' has a minimal weight, so has F ∎