Teoria dos Grafos e Computabilidade

— Trees and spanning trees —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2025

Teoria dos Grafos e Computabilidade

— Trees —

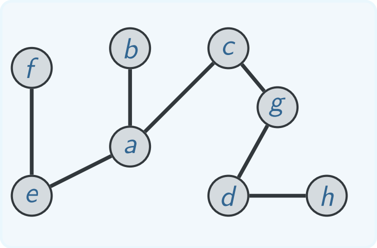
Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

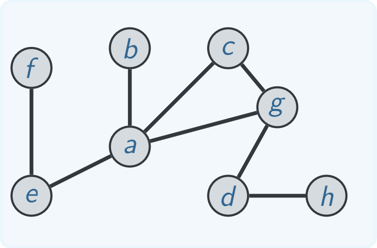
Feb 2025

Trees

- A tree is an undirected connected graph with no cycles.
- Genealogical trees, evolutionary trees, decision trees, various data structures in Computer Science



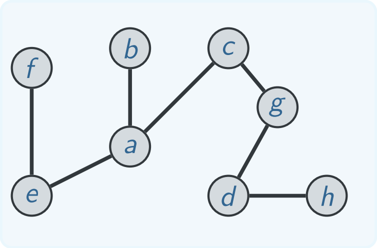
Tree



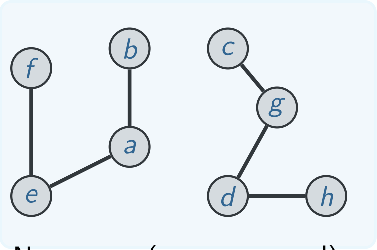
Not a tree (has cycle)

Trees

- A tree is an undirected connected graph with no cycles.
- Genealogical trees, evolutionary trees, decision trees, various data structures in Computer Science



Tree



Not a tree (not connected) – this is a forest

Trees

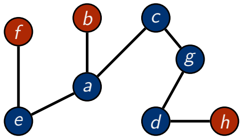
Theorem A tree has exactly one path between any pair of vertices

Theorem A tree has exactly one path between any pair of vertices

- A vertex of degree 1 is called a leaf.
- Sometimes, vertices of degree 0 are also counted as leaves
- A vertex with degree greater than 1 is an internal vertex.

Theorem A tree has exactly one path between any pair of vertices

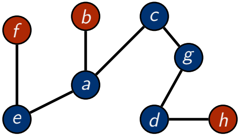
- A vertex of degree 1 is called a leaf.
- Sometimes, vertices of degree 0 are also counted as leaves
- A vertex with degree greater than 1 is an internal vertex.



Trees

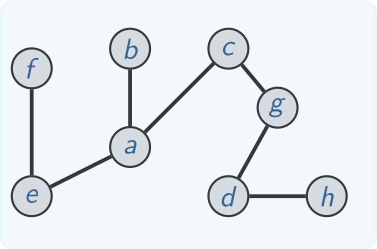
Theorem Every tree, with at least two vertices, has at least two leaves.

- A vertex of degree 1 is called a leaf.
- Sometimes, vertices of degree 0 are also counted as leaves
- A vertex with degree greater than 1 is an internal vertex.



Trees

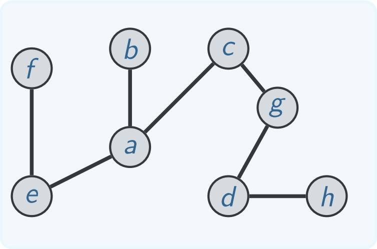
Theorem All trees on $n \geq 1$ vertices have exactly $n - 1$ edges



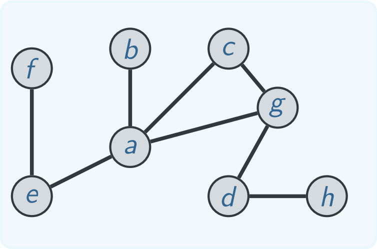
Tree

Trees

Theorem All trees on $n \geq 1$ vertices have exactly $n - 1$ edges



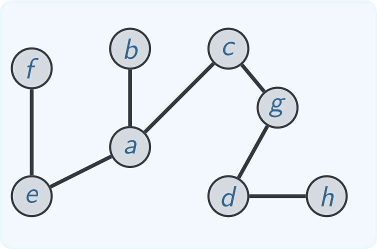
Tree



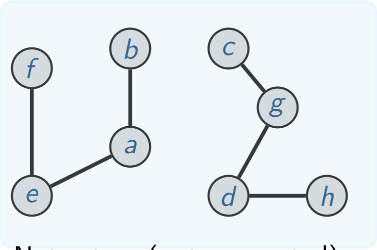
Not a tree (has cycle)

Trees

Theorem All trees on $n \geq 1$ vertices have exactly $n - 1$ edges

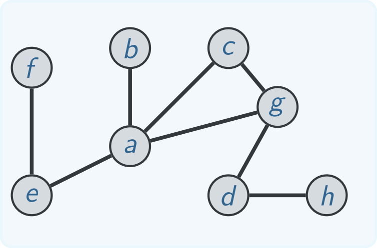


Tree

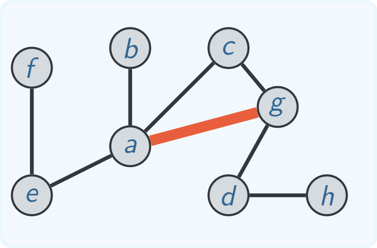


Not a tree (not connected) – this is a forest

Lemma Removing an edge from a cycle keeps connectivity



Not a tree (has cycle)



Still connected after removal

Spanning trees

A spanning tree of an undirected graph is a subgraph that is a tree and includes all vertices.

Spanning trees

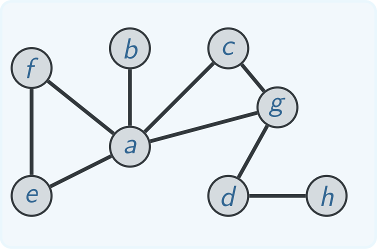
A spanning tree of an undirected graph is a subgraph that is a tree and includes all vertices.

A graph G has a spanning tree iff it is connected:

- If G has a spanning tree, it is connected: any two vertices have a path between them in the spanning tree and hence in G.
- If G is connected, we will construct a spanning tree

Spanning trees

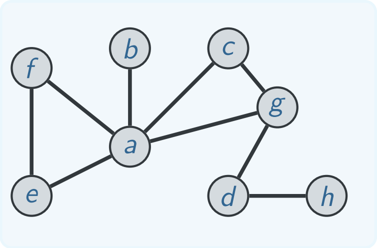
1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.



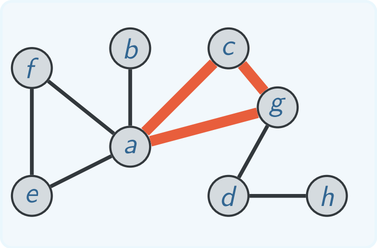
There exists a cycle?

Spanning trees

1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.



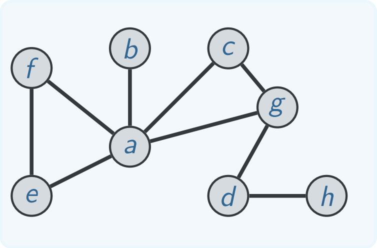
There exists a cycle?



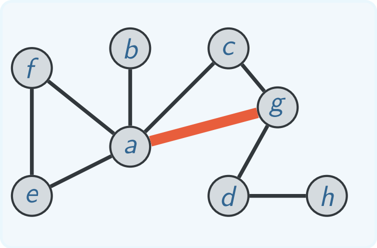
Cycle: a-c-g-a

Spanning trees

1. Let G be a connected graph on *n* vertices.
2. If there are any cycles, pick one and remove any edge.

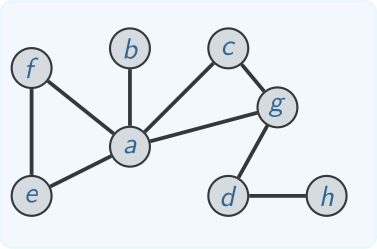


There exists a cycle?

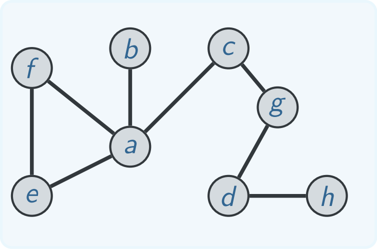


Remove the edge a-g

1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.
3. Repeat the item 2 until we arrive at a subgraph T with no cycles.



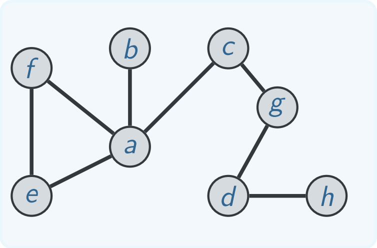
There exists a cycle?



Remove the edge a-g

Spanning trees

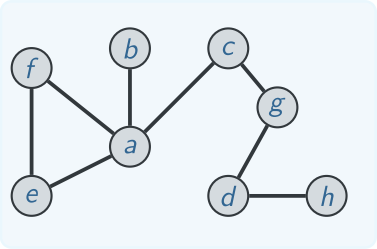
1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.



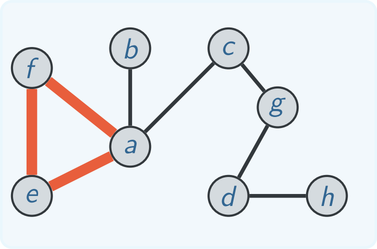
There exists a cycle?

Spanning trees

1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.

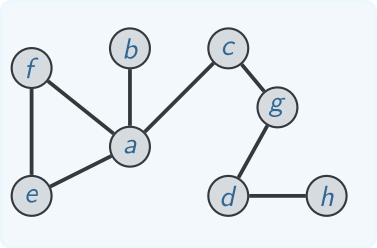


There exists a cycle?

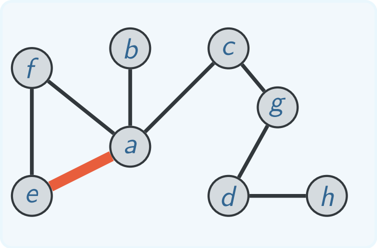


Cycle: a-f-e-a

1. Let G be a connected graph on *n* vertices.
2. If there are any cycles, pick one and remove any edge.
3. Repeat the item 2 until we arrive at a subgraph T with no cycles.



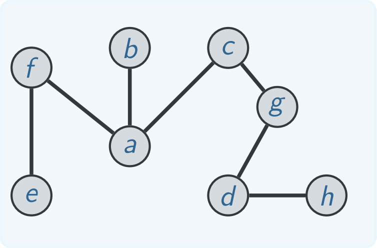
There exists a cycle?



Remove the edge a-e

Spanning trees

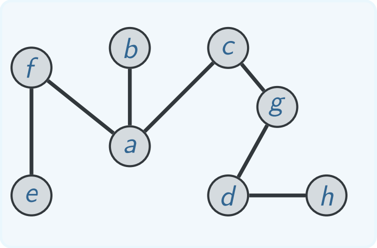
1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.
3. Repeat the item 2 until we arrive at a subgraph T with no cycles.



T is still connected, and has no cycles, so it' is a tree!

Spanning trees

1. Let G be a connected graph on n vertices.
2. If there are any cycles, pick one and remove any edge.
3. Repeat the item 2 until we arrive at a subgraph T with no cycles.



T is still connected, and has no cycles, so it' is a tree!

It reaches all vertices, so it is a spanning tree

Spanning trees

Converse theorem

If a connected graph on n vertices has $n - 1$ edges, it is a tree

Spanning trees

Converse theorem

If a connected graph on n vertices has $n-1$ edges, it is a tree

A forest is an undirected graph with no cycles and each connected component is a tree.

Spanning trees

Converse theorem

If a connected graph on n vertices has $n-1$ edges, it is a tree

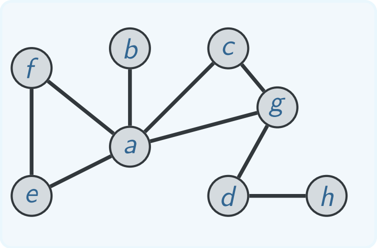
A forest is an undirected graph with no cycles and each connected component is a tree.

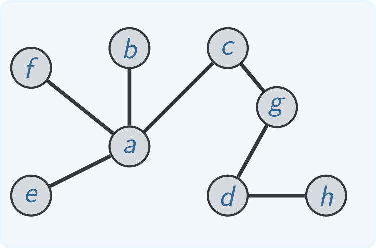
Theorem

A forest with n vertices and k trees has $n-k$ edges.

Spanning trees

Let G be a connected graph on n vertices and T be a spanning tree computed from G

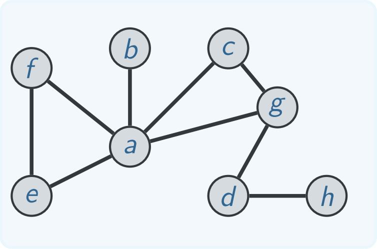


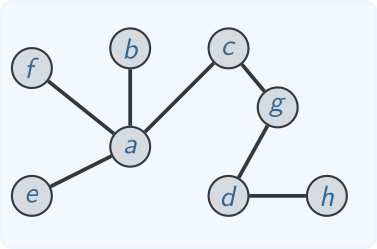


Spanning trees

Let G be a connected graph on n vertices and T be a spanning tree computed from G

1. A branch is an edge in a spanning tree T.

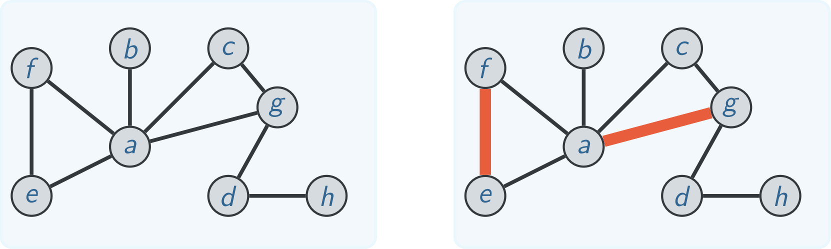




Spanning trees

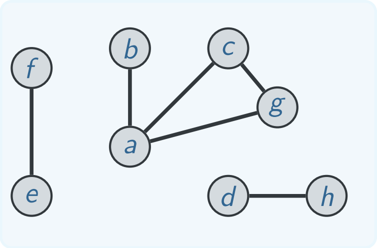
Let G be a connected graph on n vertices and T be a spanning tree computed from G

1. A branch is an edge in a spanning tree T.
2. A chord is an edge of the connected graph G that is not a branch of a spanning tree T.



Spanning forest

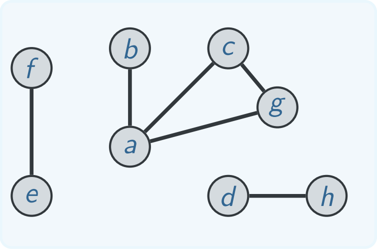
Let G be a graph on n vertices.

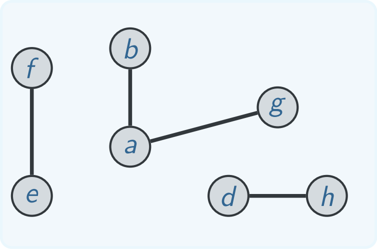


Spanning forest

Let G be a graph on n vertices.

1. A forest is a collection of trees in the graph

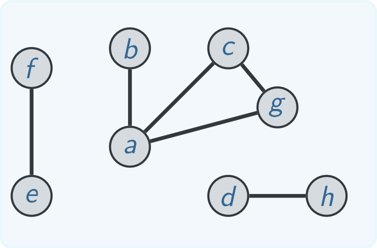


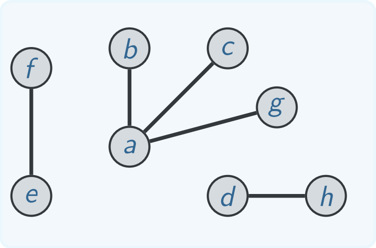


Spanning forest

Let G be a graph on n vertices.

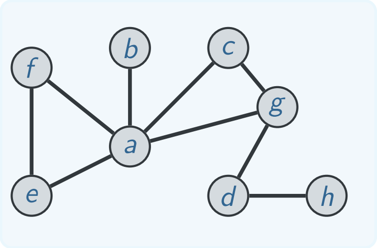
1. A forest is a collection of trees in the graph
2. A spanning forest is a collection of spanning trees.





Spanning trees

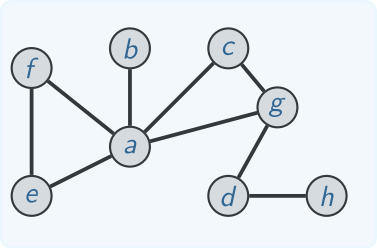
Let G be a connected graph on n vertices and T be a spanning tree computed from G



Spanning trees

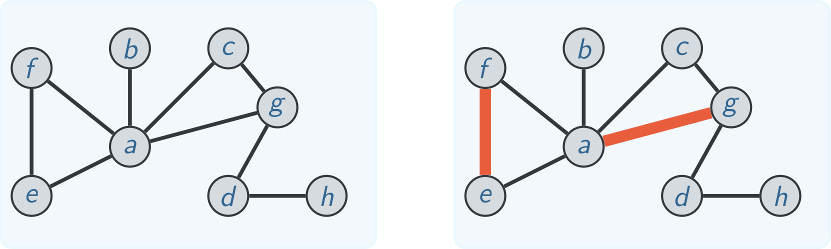
Let G be a connected graph on n vertices and T be a spanning tree computed from G

1. A rank of G is the number of branches of the spanning trees (or spanning forest) which is given by $r = n - k$



Let G be a connected graph on n vertices and T be a spanning tree computed from G

1. A **rank** of G is the number of branches of the spanning trees (or spanning forest) which is given by $r = n - k$
2. A **nullity** is the number of chords related to the spanning trees (or spanning forest) which is given by $\mu = e - n - k$



Spanning trees

Let G be a connected graph on n vertices and T be a spanning tree computed from G

1. A rank of G is the number of branches of the spanning trees (or spanning forest) which is given by $r = n - k$
2. A nullity is the number of chords related to the spanning trees (or spanning forest) which is given by $\mu = e - n - k$

Remember that k is the number of connected component. For a spanning tree, $k = 1$, but for a spanning forest, k is the number of spanning trees which are in the spanning forest.

Teoria dos Grafos e Computabilidade

— Minimum Spanning Trees —

Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2025

Minimum Spanning Tree (MST)

- Given an undirected graph $G = (V, E)$ with a cost $c_e > 0$ associated with each edge $e \in E$.
- Find a subset T of edges such that the graph (V, T) is connected and the cost $\sum_{e \in T} c_e$ is as small as possible.

Minimum Spanning Tree (MST)

- Given an undirected graph $G = (V, E)$ with a cost $c_e > 0$ associated with each edge $e \in E$.
- Find a subset T of edges such that the graph (V, T) is connected and the cost $\sum_{e \in T} c_e$ is as small as possible.

MINIMUM SPANNING TREE

INSTANCE An undirected graph $G = (V, E)$ and a function $c : E \rightarrow \mathbb{R}^+$

SOLUTION A set $T \subseteq E$ of edges such that (V, T) is connected and the $\sum_{e \in T} c_e$ is as small as possible.

Minimum Spanning Tree (MST)

- Given an undirected graph $G = (V, E)$ with a cost $c_e > 0$ associated with each edge $e \in E$.
- Find a subset T of edges such that the graph (V, T) is connected and the cost $\sum_{e \in T} c_e$ is as small as possible.

MINIMUM SPANNING TREE

INSTANCE An undirected graph $G = (V, E)$ and a function $c : E \to \mathbb{R}^+$

SOLUTION A set $T \subseteq E$ of edges such that (V, T) is connected and the $\sum_{e \in T} c_e$ is as small as possible.

- Claim: If T is a minimum-cost solution to this network design problem then (V, T) is a tree.
- A subset T of E is a spanning tree of G if (V, T) is a tree.

Greedy Algorithm for the MST Problem

- Template: process edges in some order. Add an edge to T if tree property is not violated.

Greedy Algorithm for the MST Problem

- Template: process edges in some order. Add an edge to T if tree property is not violated.

  Increasing cost order *Process edges in increasing order of cost. Discard an edge if it creates a cycle.*

  Dijkstra-like *Start from a node s and grow T outward from s: add the node that can be attached most cheaply to current tree.*

  Decreasing cost order *Delete edges in order of decreasing cost as long as graph remains connected.*

Greedy Algorithm for the MST Problem

- Template: process edges in some order. Add an edge to T if tree property is not violated.

  Increasing cost order *Process edges in increasing order of cost. Discard an edge if it creates a cycle.*

  Dijkstra-like *Start from a node s and grow T outward from s: add the node that can be attached most cheaply to current tree.*

  Decreasing cost order *Delete edges in order of decreasing cost as long as graph remains connected.*

- Which of these algorithms works?

Greedy Algorithm for the MST Problem

- Template: process edges in some order. Add an edge to T if tree property is not violated.

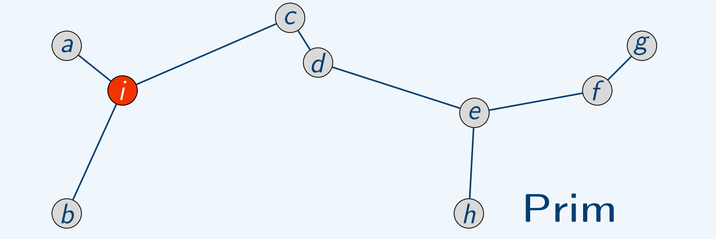
  Increasing cost order *Process edges in increasing order of cost. Discard an edge if it creates a cycle.* *Kruskal's algorithm*

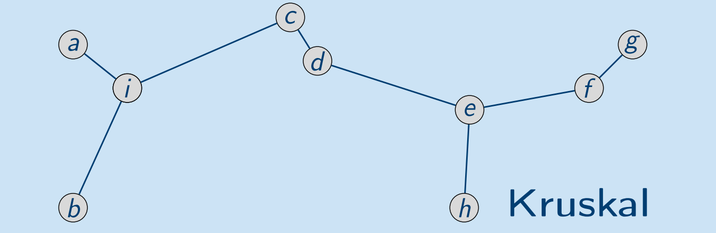
  Dijkstra-like *Start from a node s and grow T outward from s: add the node that can be attached most cheaply to current tree.* *Prim's algorithm*

  Decreasing cost order *Delete edges in order of decreasing cost as long as graph remains connected.* *Reverse-Delete algorithm*

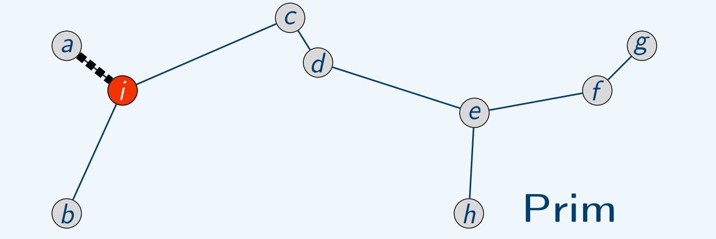
- Which of these algorithms works? All of them!

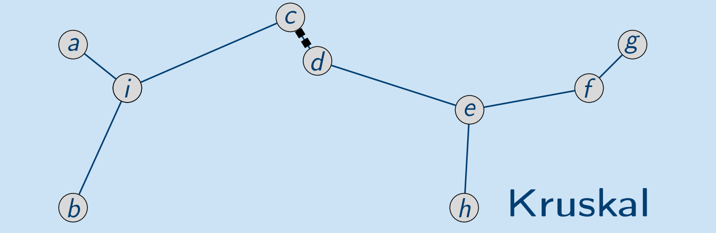
Example of Prim's and Kruskal's Algorithms



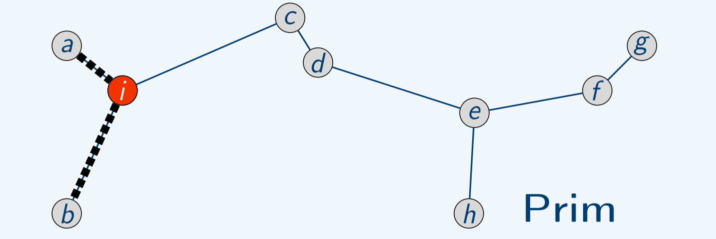


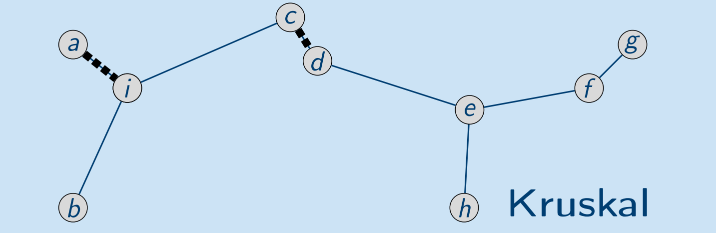
Example of Prim's and Kruskal's Algorithms



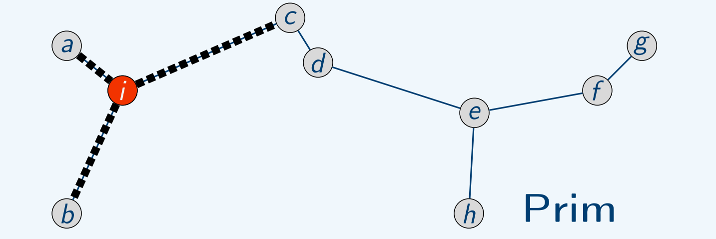


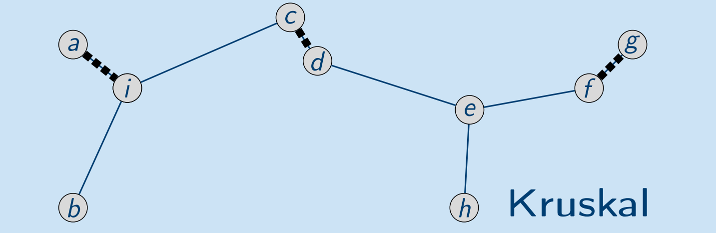
Example of Prim's and Kruskal's Algorithms

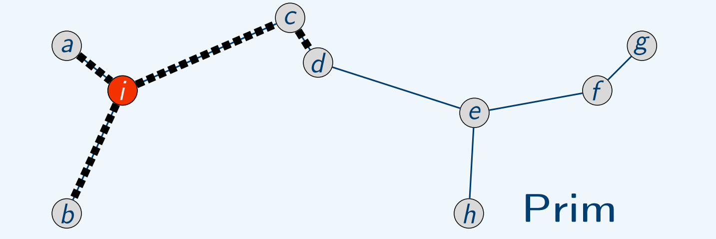


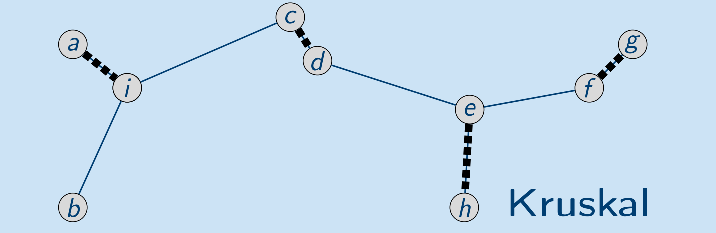


Example of Prim's and Kruskal's Algorithms

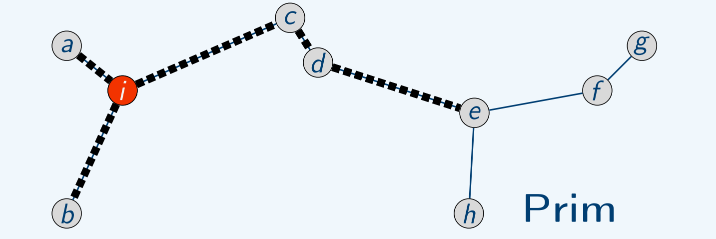


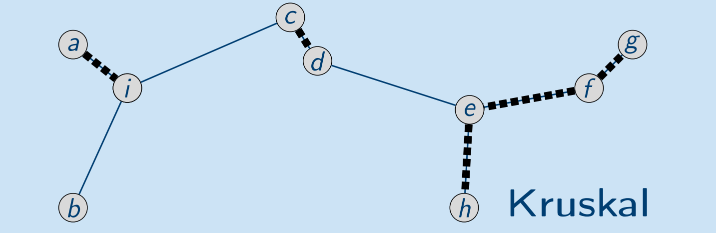



a
b
c
d
e
f
g
h
i
Prim


a
b
c
d
e
f
g
h
i
Kruskal

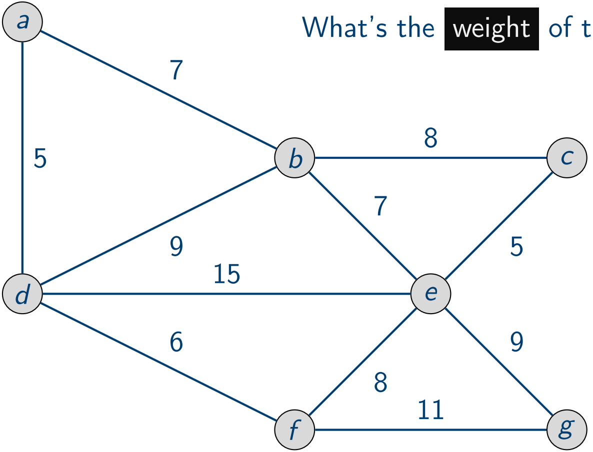
Example of Prim's and Kruskal's Algorithms





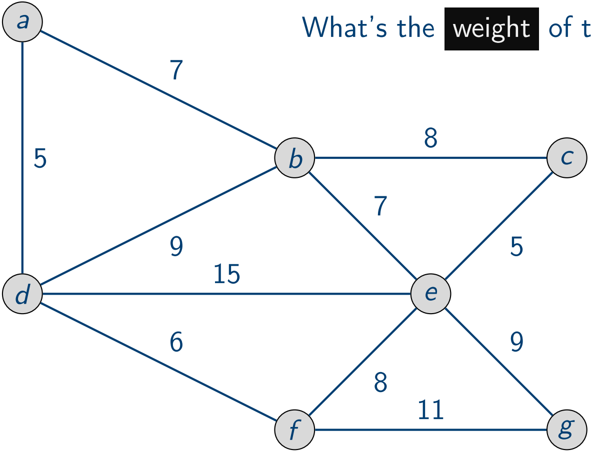
Example of Prim's Algorithm

What's the weight of the MST?



Example of Kruskal's Algorithm

What's the weight of the MST?

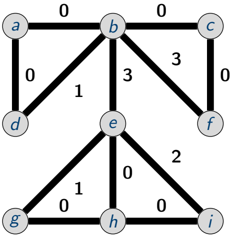


Graph Cuts

- A cut in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).

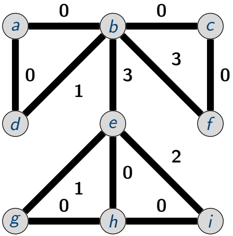
Graph Cuts

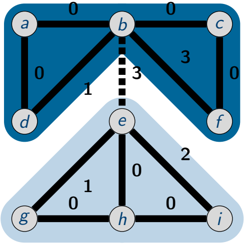
- A cut in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).



Graph Cuts

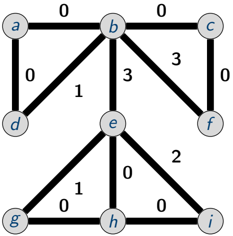
- A cut in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).

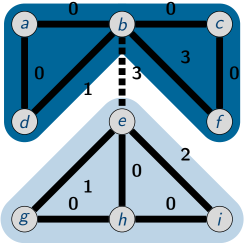




Graph Cuts

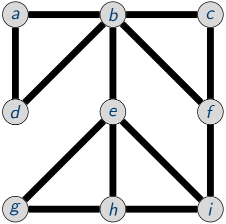
- A cut in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: cut(S) is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.





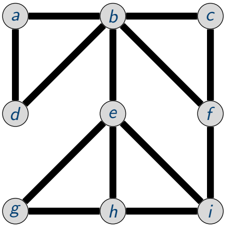
Graph Cuts

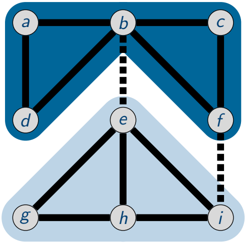
- A cut in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



Graph Cuts

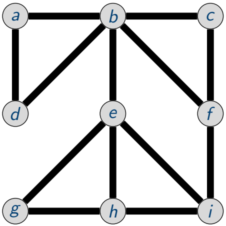
- A cut in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.

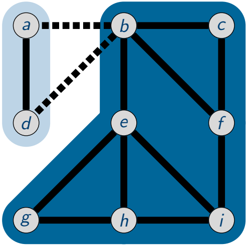




Graph Cuts

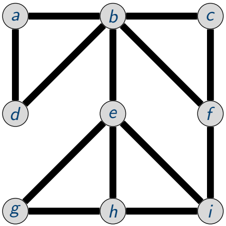
- A cut in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.

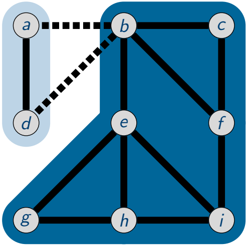




Graph Cuts

- A cut in a graph $G = (V, E)$ is a set of edges whose removal **disconnects** the graph (into two or more connected components).
- Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: cut(S) is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.
- cut(S) is a cut because deleting the edges in cut(S) disconnects S from $V - S$.



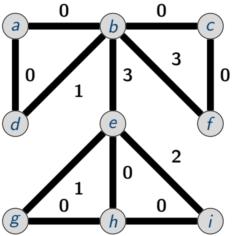


Graph Cuts

- A cut in a graph $G = (V, E)$ is a set of edges whose removal disconnects the graph (into two or more connected components).

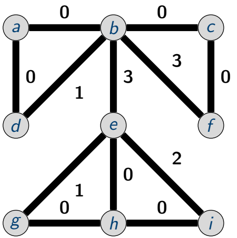
Graph Cuts

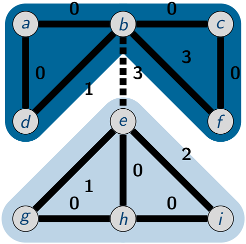
- A cut in a graph $G = (V, E)$ is a set of edges whose removal disconnects the graph (into two or more connected components).



Graph Cuts

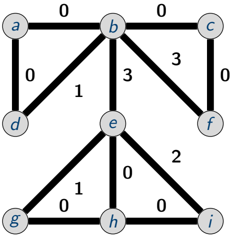
- A cut in a graph $G = (V, E)$ is a set of edges whose removal disconnects the graph (into two or more connected components).

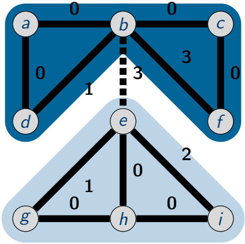




Graph Cuts

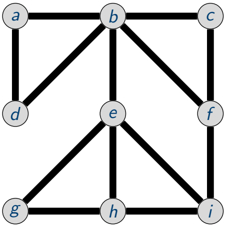
- A cut in a graph $G = (V, E)$ is a set of edges whose removal disconnects the graph (into two or more connected components).
- Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\mathrm{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.





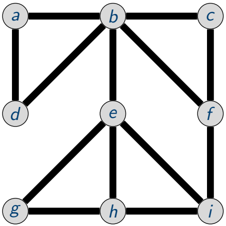
Graph Cuts

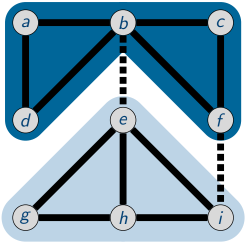
- A cut in a graph $G = (V, E)$ is a set of edges whose removal disconnects the graph (into two or more connected components).
- Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: cut(S) is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.



Graph Cuts

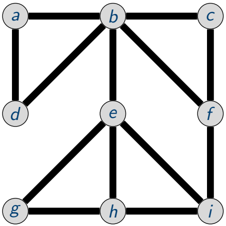
- A cut in a graph $G = (V, E)$ is a set of edges whose removal disconnects the graph (into two or more connected components).
- Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: cut(S) is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.

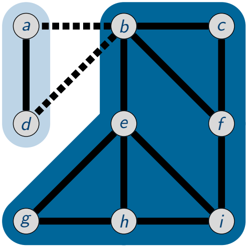




Graph Cuts

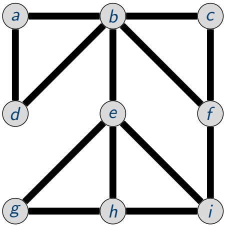
- A cut in a graph $G = (V, E)$ is a set of edges whose removal disconnects the graph (into two or more connected components).
- Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.

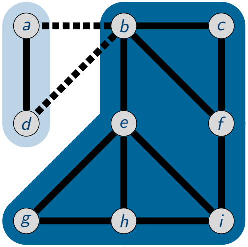




Graph Cuts

- A cut in a graph $G = (V, E)$ is a set of edges whose removal disconnects the graph (into two or more connected components).
- Every set $S \subset V$ (S cannot be empty or the entire set V) has a corresponding cut: $\text{cut}(S)$ is the set of edges (v, w) such that $v \in S$ and $w \in V - S$.
- $\text{cut}(S)$ is a cut because deleting the edges in $\text{cut}(S)$ disconnects S from $V - S$.





Cut Property

- When is it safe to include an edge in an MST?

Cut Property

- When is it safe to include an edge in an MST?
- Assume all edge costs are distinct.
- Let $S \subset V$, S is not empty or equal to V.
- Let e be the cheapest edge in cut(S).
- Claim: every MST contains e.

Cut Property

- When is it safe to include an edge in an MST?
- Assume all edge costs are distinct.
- Let $S \subset V$, S is not empty or equal to V.
- Let e be the cheapest edge in cut(S).
- Claim: every MST contains e.
- Proof: exchange argument. If a supposed MST T does not contain e, show that there is a tree with smaller cost than T that contains e.

Using the Cut Property

- Let F be the set of all edges that satisfy the cut property.
- Is the graph induced by F connected?
- Can the graph induced by F contain a cycle?
- How many edges can F contain?

Using the Cut Property

- Let F be the set of all edges that satisfy the cut property.
- Is the graph induced by F connected? **Yes.**
- Can the graph induced by F contain a cycle? **No.**
- How many edges can F contain? $n - 1$

Using the Cut Property

- Let F be the set of all edges that satisfy the cut property.
- Is the graph induced by F connected? **Yes.**
- Can the graph induced by F contain a cycle? **No.**
- How many edges can F contain? $n - 1$
- F is the unique MST.
- Kruskal's and Prim's algorithms compute F **efficiently**.

Optimality of Kruskal's Algorithm

- Kruskal's algorithm:
  - Start with an empty set T of edges.
  - Process edges in E in non decreasing order of cost.
  - Add the next edge e to T only if adding e does not create a cycle. Discard e if it creates a cycle.
- Claim: Kruskal's algorithm outputs an MST.

Optimality of Kruskal's Algorithm

- Kruskal's algorithm:
  - Start with an empty set T of edges.
  - Process edges in E in non decreasing order of cost.
  - Add the next edge e to T only if adding e does not create a cycle. Discard e if it creates a cycle.
- Claim: Kruskal's algorithm outputs an MST.
  1. For every edge e added, demonstrate the existence of S and $V - S$ such that e and S satisfy the cut property.
  2. Prove that the algorithm computes a spanning tree.

Optimality of Prim's Algorithm

- Prim's algorithm: Maintain a tree (S, U)
  - Start with an arbitrary node $s \in S$ and $U = \emptyset$.
  - Add the node v to S and the edge e to U that minimize

$$\min_{e=(u,v),u \in S, v \notin S} c_e \equiv \min_{e \in \mathrm{cut}(S)} c_e.$$

  - Stop when $S = V$.
- Claim: Prim's algorithm outputs an MST.

Optimality of Prim's Algorithm

- Prim's algorithm: Maintain a tree (S, U)
  - Start with an arbitrary node $s \in S$ and $U = \emptyset$.
  - Add the node v to S and the edge e to U that minimize

$$\min_{e=(u,v), u \in S, v \notin S} c_e \equiv \min_{e \in \text{cut}(S)} c_e.$$

  - Stop when $S = V$.
- Claim: Prim's algorithm outputs an MST.
  1. Prove that every edge inserted satisfies the cut property.
  2. Prove that the graph constructed is a spanning tree.

Cycle Property

- When can we be sure that an edge cannot be in any MST?

Cycle Property

- When can we be sure that an edge cannot be in any MST?
- Let C be any cycle in G and let $e = (v, w)$ be the most expensive edge in C.
- Claim: e does not belong to any MST of G.

Cycle Property

- When can we be sure that an edge cannot be in any MST?
- Let C be any cycle in G and let $e = (v, w)$ be the most expensive edge in C.
- Claim: e does not belong to any MST of G.
- Proof: exchange argument. If a supposed MST T contains e, show that there is a tree with smaller cost than T that does not contain e.

Optimality of the Reverse-Delete Algorithm

- Reverse-Delete algorithm: Maintain a set E' of edges.
  - Start with $E' = E$.
  - Process edges in non increasing order of cost.
  - Delete the next edge e from E' only if (V, E') is connected after removal.
  - Stop after processing all the edges.
- Claim: the Reverse-Delete algorithm outputs an MST.

Optimality of the Reverse-Delete Algorithm

- Reverse-Delete algorithm: Maintain a set E' of edges.
  - Start with $E' = E$.
  - Process edges in non increasing order of cost.
  - Delete the next edge e from E' only if (V, E') is connected after removal.
  - Stop after processing all the edges.
- Claim: the Reverse-Delete algorithm outputs an MST.
  1. Show that every edge deleted belongs to no MST.
  2. Prove that the graph remaining at the end is a spanning tree.

Comments on MST Algorithms

- To handle multiple edges with the same weight, perturb each length by a random infinitesimal amount.
- Any algorithm that constructs a spanning tree by including edges that satisfy the cut property and deleting edges that satisfy the cycle property will yield an **MST**!

Teoria dos Grafos e Computabilidade

— Steiner Trees —

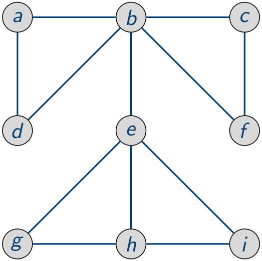
Silvio Jamil F. Guimarães

Graduate Program in Informatics – PPGINF
Laboratory of Image and Multimedia Data Science – IMScience
Pontifical Catholic University of Minas Gerais – PUC Minas

Feb 2025

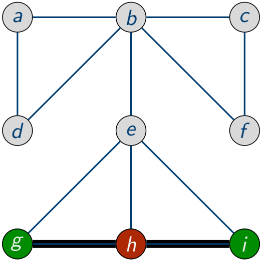
Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimun size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree .



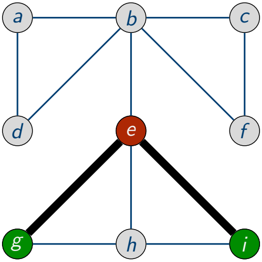
Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimun size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree .



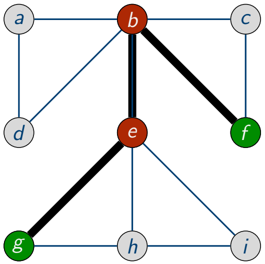
Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimun size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree .



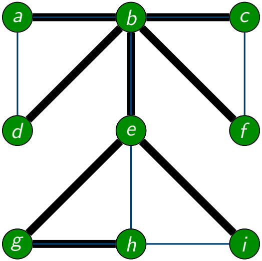
Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimun size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree .



Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimun size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree .



Is a spanning tree T' of G a Steiner tree?

Steiner Tree

Given a connected undirected graph $G = (V, E)$ and a set of $T \subseteq V$. A minimun size tree $H = (V', E')$ subgraph of G such that $T \subseteq V'$ is called as Steiner tree .

- The vertices in T are called terminals
- The vertices in $V \ \ T$ are called Steiner points
- Denote $n = |V|$, $m = |E|$ and $t = |T|$
- A minimum size:
  - Vertex cardinality: $|V'|$ or rather $|S| = |V' \ \ T|$ (default)
  - Edge cardinality: $|E'| = |V'| - 1$
  - Node weighted: Given $w : V \to \mathbb{N}$ minimize w(S)
  - Edge weighted: Given $w : E \to \mathbb{N}$ minimize w(E')