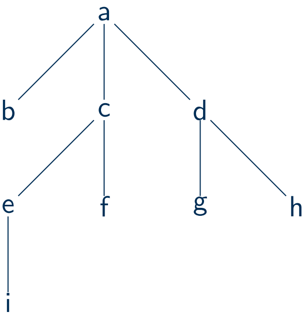
Trees

Dr. Chuck Rocca
`roccac@wcsu.edu`
`http://sites.wcsu.edu/roccac`

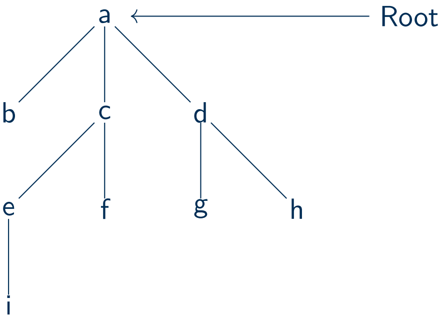
WESTERN
CONNECTICUT
STATE UNIVERSITY
MACRICOSTAS
SCHOOL OF ARTS
& SCIENCES

Table of Contents

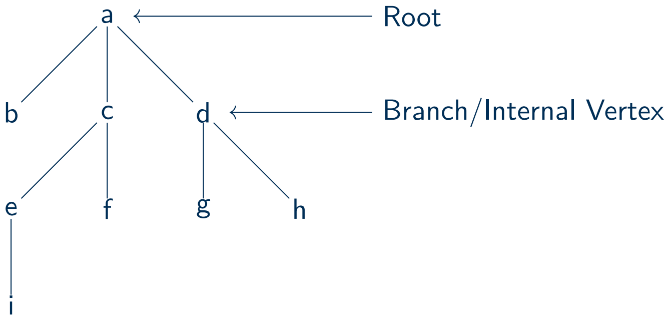
Tree Terminology



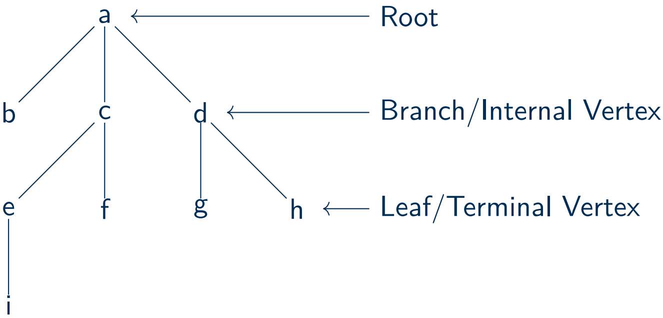
Tree Terminology



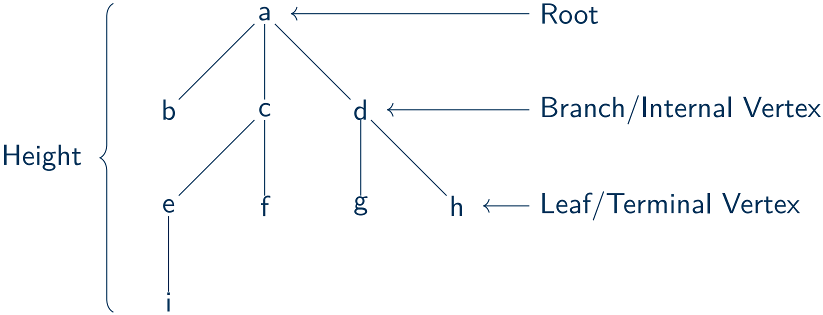
Tree Terminology



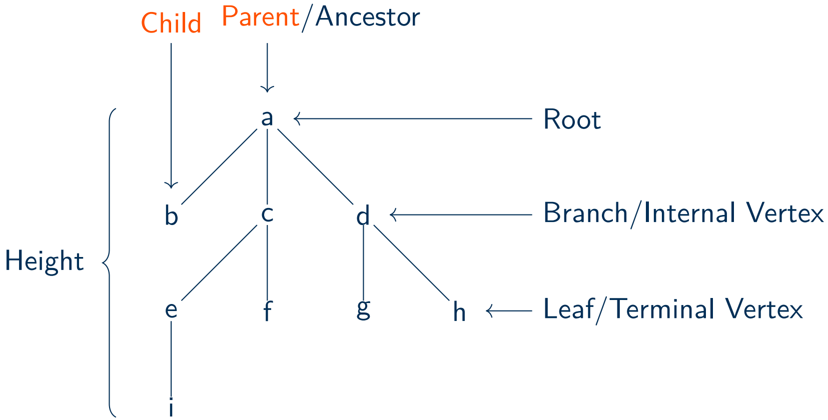
Tree Terminology



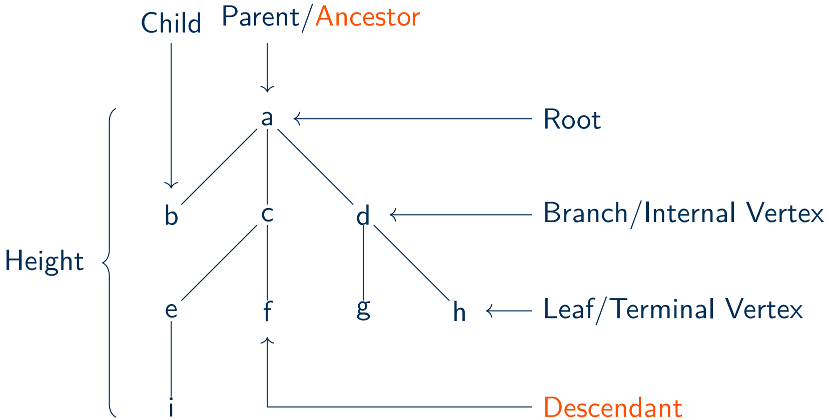
Tree Terminology



Tree Terminology



Tree Terminology



Tree Terminology

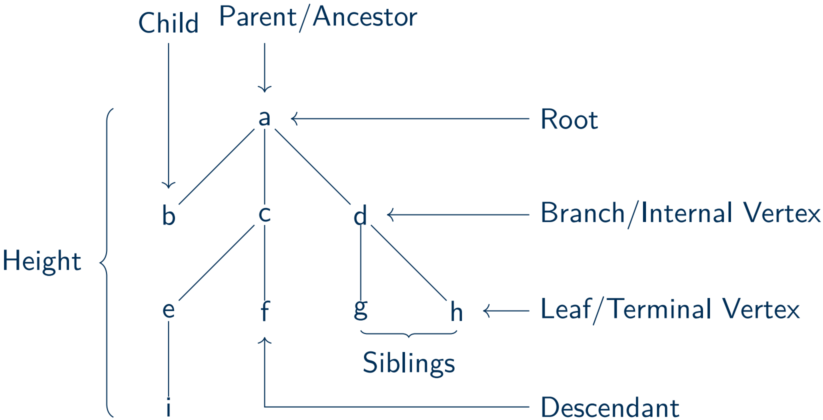


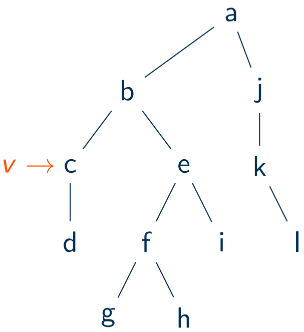
Table of Contents

Vertices and Edges

Lemma

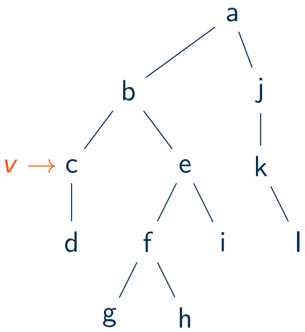
Trees with two or more vertices has a vertex of degree 1.

Degree 1 Vertex



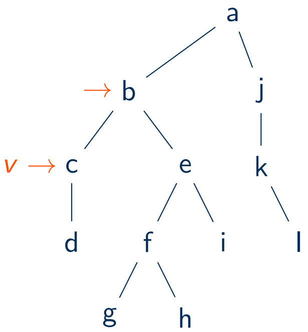
- pick a vertex v

Degree 1 Vertex



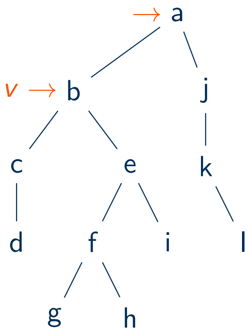
- pick a vertex v
- while $deg(v) > 1$:

Degree 1 Vertex



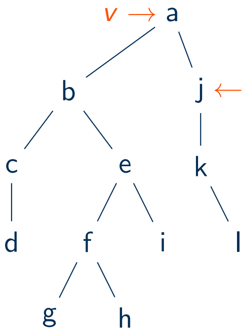
- pick a vertex v
- while $deg(v) > 1$:
    - move to an un-visited vertex

Degree 1 Vertex



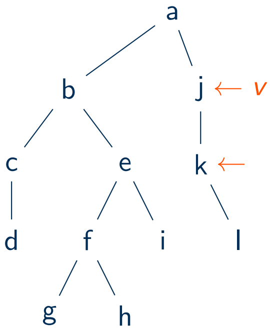
- pick a vertex v
- while $deg(v) > 1$:
  - move to an un-visited vertex
  - call it v

Degree 1 Vertex



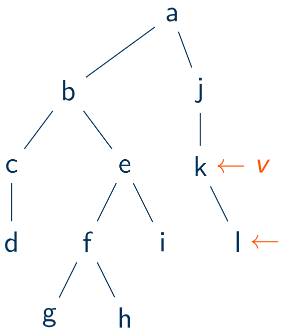
- pick a vertex v
- while $deg(v) > 1$:
  - move to an un-visited vertex
  - call it v

Degree 1 Vertex



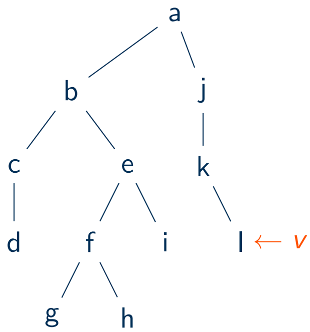
- pick a vertex *v*
- while $deg(v) > 1$:
  - move to an un-visited vertex
  - call it *v*

Degree 1 Vertex



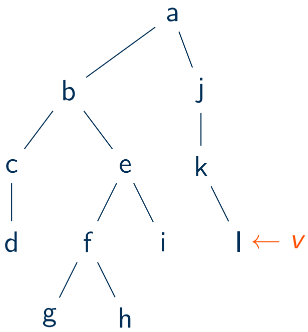
- pick a vertex v
- while $deg(v) > 1$:
    - move to an un-visited vertex
    - call it v

Degree 1 Vertex



- pick a vertex v
- while $deg(v) > 1$:
  - move to an un-visited vertex
  - call it v

Degree 1 Vertex



- pick a vertex v
- while $deg(v) > 1$:
    - move to an un-visited vertex
    - call it v
- Terminates as long as there are no circuits and the graph is finite.

Vertices and Edges

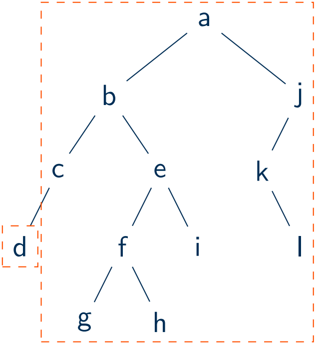
Lemma

Trees with two or more vertices has a vertex of degree 1.

Theorem

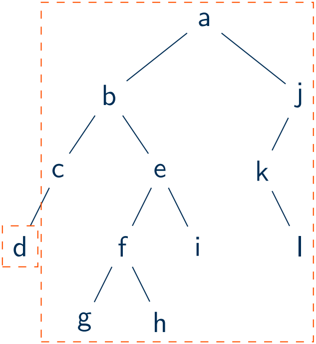
A tree with n vertices has exactly $n - 1$ edges.

$n - 1$ Edges in a Tree



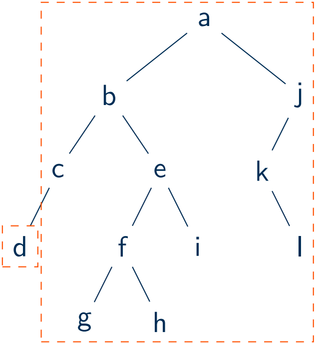
- "Obvious" if there are just two vertices.

$n - 1$ Edges in a Tree



- “Obvious” if there are just two vertices.
- By previous lemma there is at least 1 leaf

$n-1$ Edges in a Tree



- “Obvious” if there are just two vertices.
- By previous lemma there is at least 1 leaf
- Use induction with the subgraph not containing the leaf

Vertices and Edges

Lemma

Trees with two or more vertices has a vertex of degree 1.

Theorem

A tree with n vertices has exactly $n - 1$ edges.

Theorem

A connected graph with n vertices and $n - 1$ edges is a tree.

Vertices and Edges

Lemma

Trees with two or more vertices has a vertex of degree 1.

Theorem

A tree with n vertices has exactly $n - 1$ edges.

Theorem

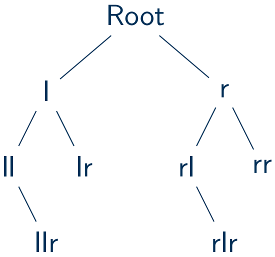
A connected graph with n vertices and $n - 1$ edges is a tree.

Theorem

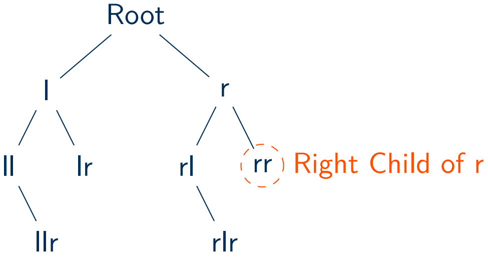
A graph with at least as many edges as vertices has a circuit.

Table of Contents

Binary Tree

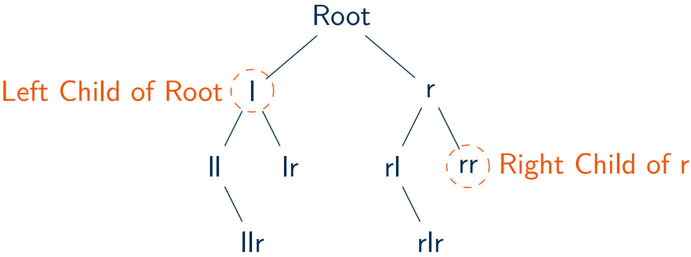


Binary Tree

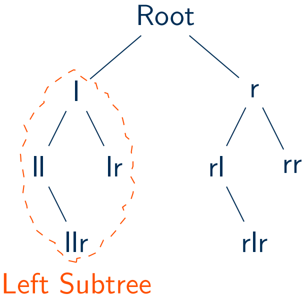


Right Child of r

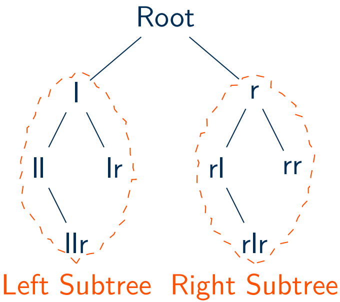
Binary Tree



Binary Tree

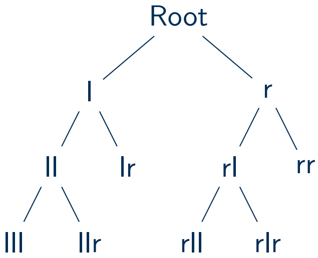


Binary Tree



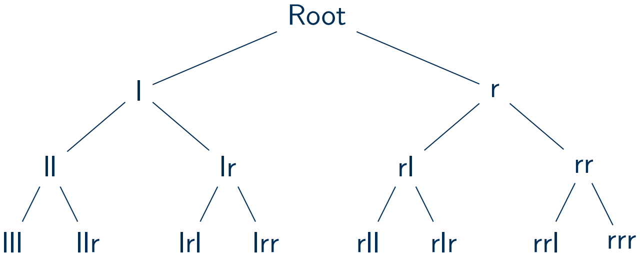
Full Binary Tree

Every *parent* has two *children*.



Complete Binary Tree

A *full binary tree* in which every *leaf* is at the same height.

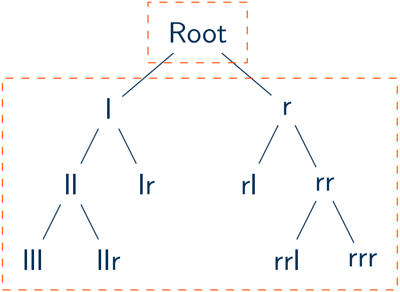


Theorems on Binary Trees

Theorem

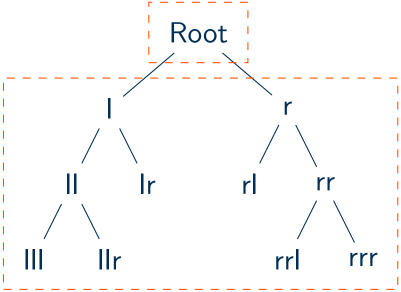
A full binary tree with k internal vertices has $2k+1$ total vertices and $k+1$ leaves.

Full Trees and Vertices



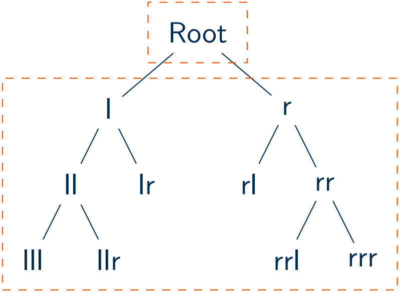
- All internal vertices have 2 children, $2k$ children.

Full Trees and Vertices



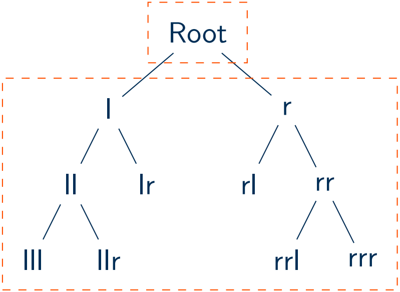
- All internal vertices have 2 children, $2k$ children.
- Only one vertex has no parent, *Root*.

Full Trees and Vertices



- All internal vertices have 2 children, $2k$ children.
- Only one vertex has no parent, *Root*.
- *Total Vertices* $= 2k + 1$.

Full Trees and Vertices



- All internal vertices have 2 children, $2k$ children.
- Only one vertex has no parent, *Root*.
- $\mathit{Total\ Vertices} = 2k + 1$.
- Leaves aren't internal, $\mathit{Leaves} = 2k + 1 - k = k + 1$.

Theorems on Binary Trees

Theorem

A full binary tree with k internal vertices has $2k+1$ total vertices and $k+1$ leaves.

Theorem

Given a binary tree with height h and t leaves,

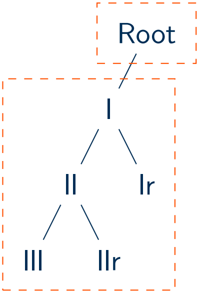
$$t \leq 2^h \text{ and } \log_2(t) \leq h.$$

Leaves and Height

Root

- $h = 0 \Rightarrow 2^h = 2^0 = 1 \geq t$

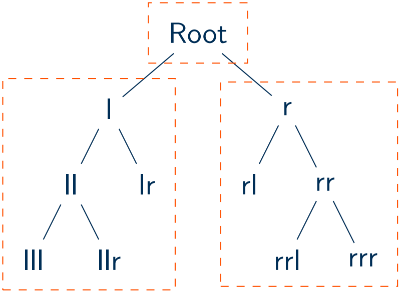
Leaves and Height



- $h = 0 \Rightarrow 2^h = 2^0 = 1 \geq t$
- $h = k + 1$ and *Root* has one child

$$\begin{aligned} 2^{k+1} &= 2 \cdot 2^k \\ &\geq 2^k + 1 \\ &\geq t_l + 1 \\ &= t \end{aligned}$$

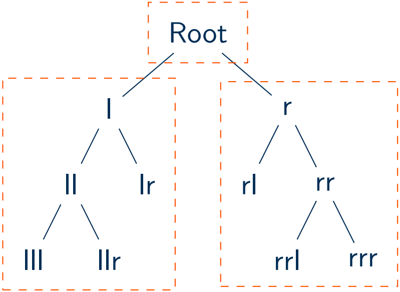
Leaves and Height



- $h = 0 \Rightarrow 2^h = 2^0 = 1 \geq t$
- $h = k + 1$ and *Root* has one child
- $h = k + 1$ and *Root* has two children

$$\begin{aligned} 2^{k+1} &= 2 \cdot 2^k \\ &\geq t_l + t_r \\ &= t \end{aligned}$$

Leaves and Height



- $h = 0 \Rightarrow 2^h = 2^0 = 1 \geq t$
- $h = k + 1$ and *Root* has one child
- $h = k + 1$ and *Root* has two children
- $t \leq 2^h$ implies $\log_2(t) \leq h$

Theorems on Binary Trees

Theorem

A full binary tree with k *internal vertices has* $2k+1$ *total vertices and* $k+1$ *leaves.*

Theorem

Given a binary tree with height h *and* t *leaves,*

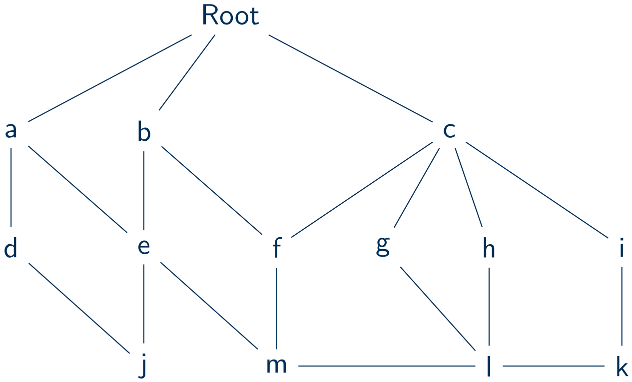
$$t \leq 2^h \text{ and } \log_2(t) \leq h.$$

Corollary

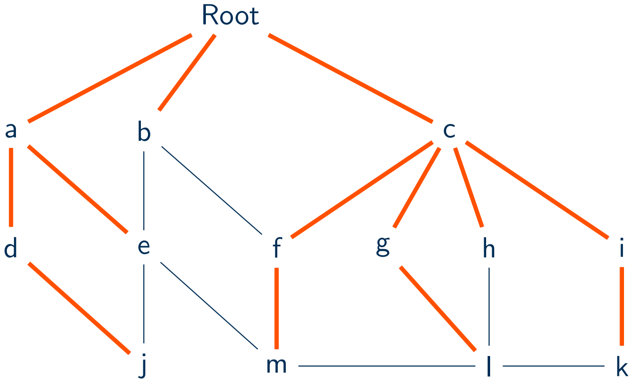
A complete binary tree of height h *has exactly* $t = 2^h$ *leaves.*

Table of Contents

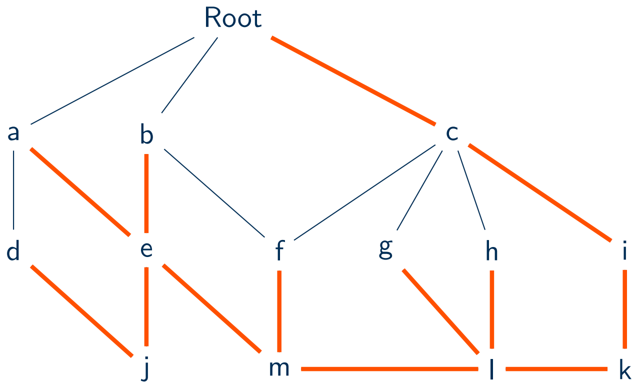
Spanning Tree Definition and Examples



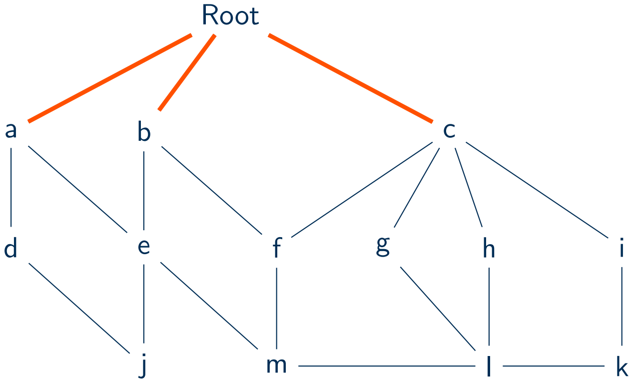
Spanning Tree Definition and Examples



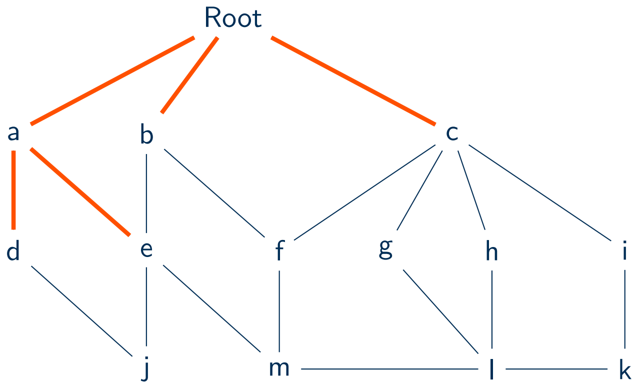
Spanning Tree Definition and Examples



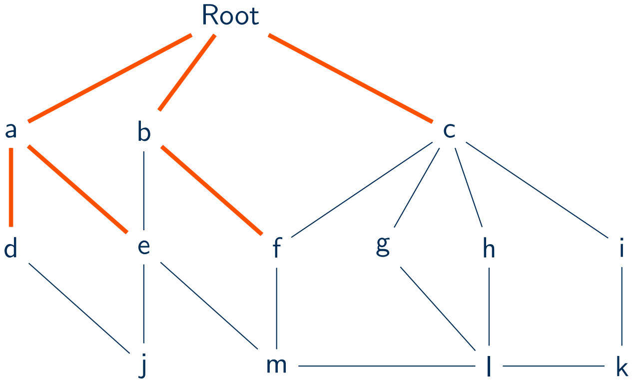
Spanning Tree Definition and Examples



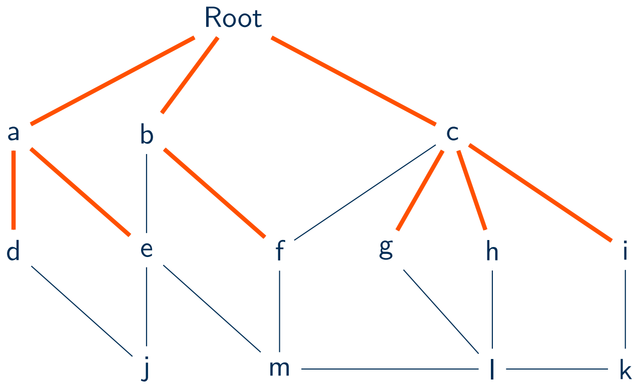
Spanning Tree Definition and Examples



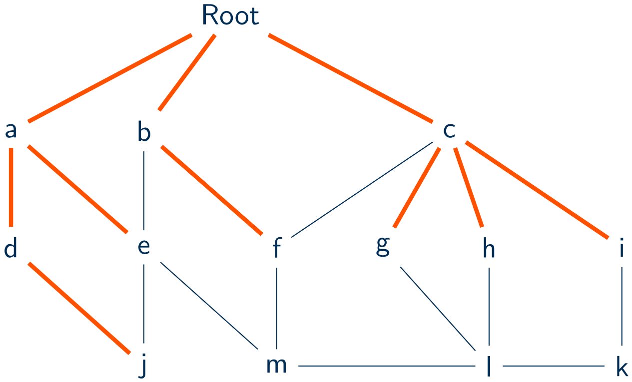
Spanning Tree Definition and Examples



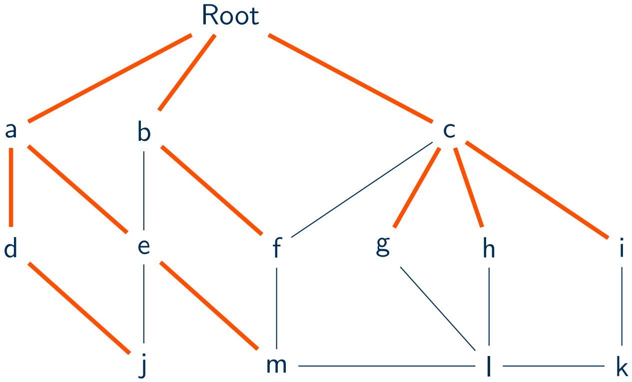
Spanning Tree Definition and Examples



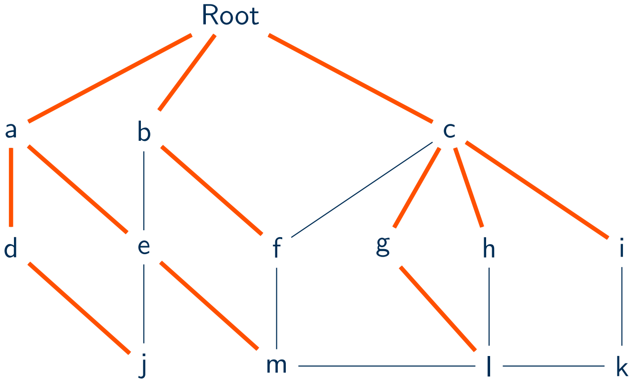
Spanning Tree Definition and Examples



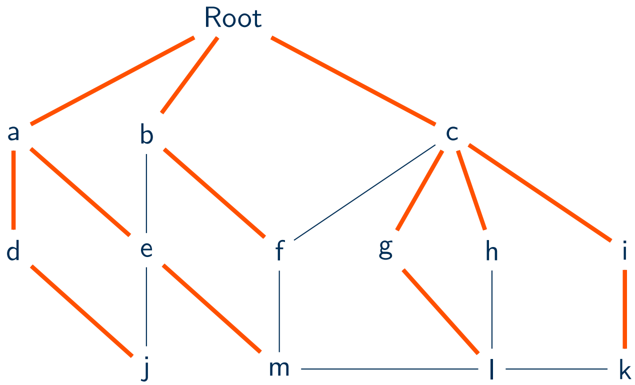
Spanning Tree Definition and Examples



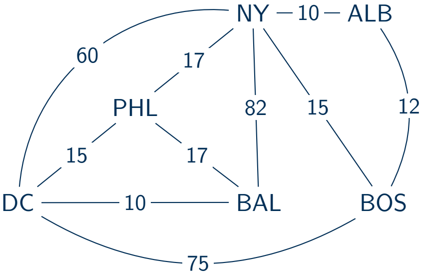
Spanning Tree Definition and Examples



Spanning Tree Definition and Examples

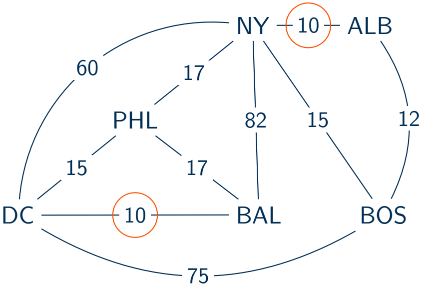


Kruskal's Minimal Tree Algorithm



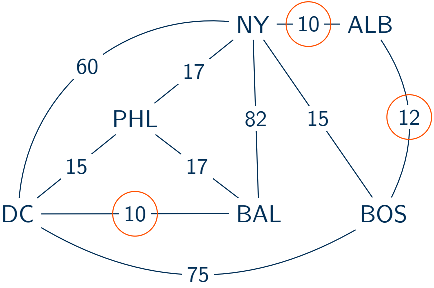
- Find the unused edge with the lowest value
- If it doesn't create a circuit add it to the tree
- Repeat until there are $n - 1$ edges

Kruskal's Minimal Tree Algorithm



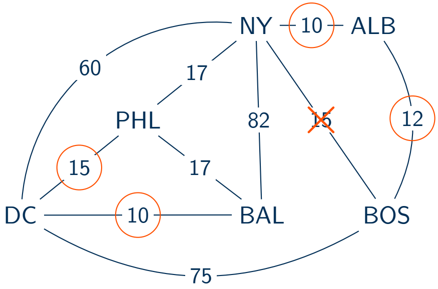
- Find the unused edge with the lowest value
- If it doesn't create a circuit add it to the tree
- Repeat until there are $n - 1$ edges

Kruskal's Minimal Tree Algorithm



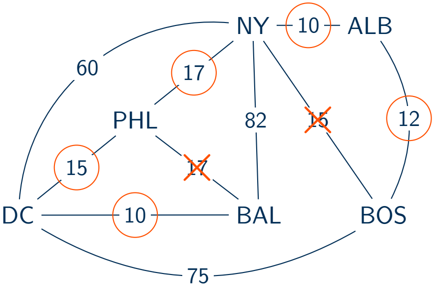
- Find the unused edge with the lowest value
- If it doesn't create a circuit add it to the tree
- Repeat until there are $n - 1$ edges

Kruskal's Minimal Tree Algorithm



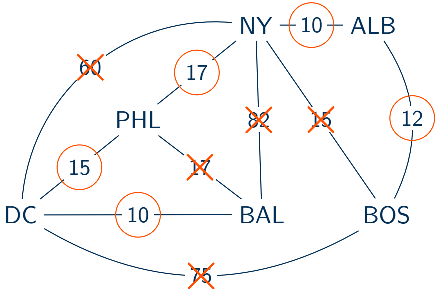
- Find the unused edge with the lowest value
- If it doesn't create a circuit add it to the tree
- Repeat until there are $n-1$ edges

Kruskal's Minimal Tree Algorithm



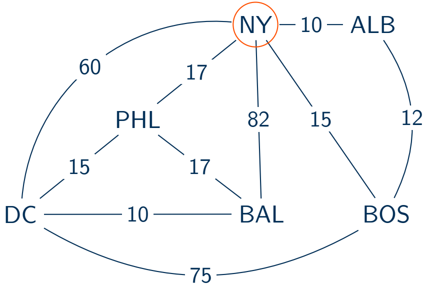
- Find the unused edge with the lowest value
- If it doesn't create a circuit add it to the tree
- Repeat until there are $n - 1$ edges

Kruskal's Minimal Tree Algorithm



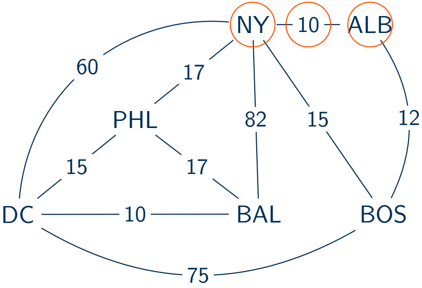
- Find the unused edge with the lowest value
- If it doesn't create a circuit add it to the tree
- Repeat until there are $n-1$ edges

Prim's Minimal Tree Algorithm



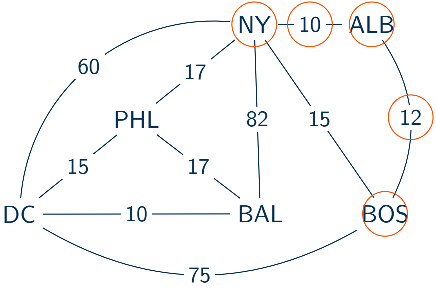
- Pick a starting vertex to add to the tree
- Add the edge that has least weight and connects to ***exactly*** one vertex already in the tree
- Add the vertex on the other end of the edge to the tree
- Repeat $n - 1$ times

Prim's Minimal Tree Algorithm



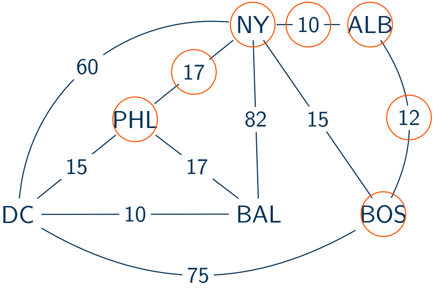
- Pick a starting vertex to add to the tree
- Add the edge that has least weight and connects to ***exactly*** one vertex already in the tree
- Add the vertex on the other end of the edge to the tree
- Repeat $n - 1$ times

Prim's Minimal Tree Algorithm



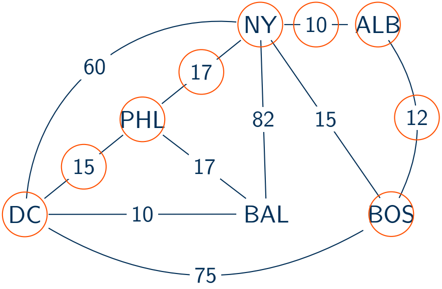
- Pick a starting vertex to add to the tree
- Add the edge that has least weight and connects to ***exactly*** one vertex already in the tree
- Add the vertex on the other end of the edge to the tree
- Repeat $n-1$ times

Prim's Minimal Tree Algorithm



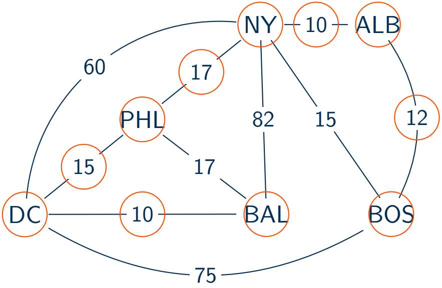
- Pick a starting vertex to add to the tree
- Add the edge that has least weight and connects to ***exactly*** one vertex already in the tree
- Add the vertex on the other end of the edge to the tree
- Repeat $n - 1$ times

Prim's Minimal Tree Algorithm



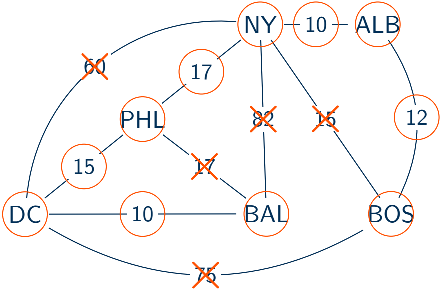
- Pick a starting vertex to add to the tree
- Add the edge that has least weight and connects to ***exactly*** one vertex already in the tree
- Add the vertex on the other end of the edge to the tree
- Repeat $n - 1$ times

Prim's Minimal Tree Algorithm



- Pick a starting vertex to add to the tree
- Add the edge that has least weight and connects to ***exactly*** one vertex already in the tree
- Add the vertex on the other end of the edge to the tree
- Repeat $n - 1$ times

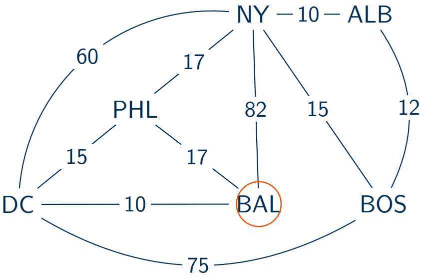
Prim's Minimal Tree Algorithm



- Pick a starting vertex to add to the tree
- Add the edge that has least weight and connects to ***exactly*** one vertex already in the tree
- Add the vertex on the other end of the edge to the tree
- Repeat $n - 1$ times

Dijkstra's "Shortest" Path Algorithm

Baltimore to Boston



Round 1

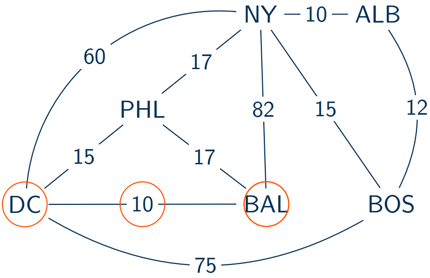
$v = BAL$

$F = \{DC, PHL, NY\}$

City	Old Label	New Label
ALB	$(\infty, -)$	$(\infty, -)$
BAL	$(0, -)$	$(0, -)$
BOS	$(\infty, -)$	$(\infty, -)$
DC	$(\infty, -)$	$(10, BAL)$
NY	$(\infty, -)$	$(82, BAL)$
PHL	$(\infty, -)$	$(17, BAL)$

Dijkstra's "Shortest" Path Algorithm

Baltimore to Boston



Round 2

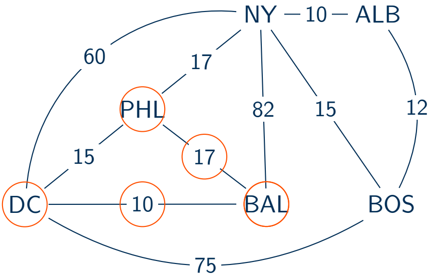
$v = DC$

$F = \{PHL, NY, BOS\}$

City	Old Label	New Label
ALB	$(\infty, -)$	$(\infty, -)$
BAL	$(0, -)$	$(0, -)$
BOS	$(\infty, -)$	$(85, DC)$
DC	$(10, BAL)$	$(10, BAL)$
NY	$(82, BAL)$	$(70, DC)$
PHL	$(17, BAL)$	$(17, BAL)$

Dijkstra's "Shortest" Path Algorithm

Baltimore to Boston



Round 3

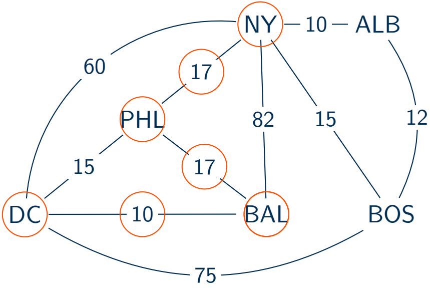
$v = PHL$

$F = \{NY, BOS\}$

City	Old Label	New Label
ALB	$(\infty, -)$	$(\infty, -)$
BAL	$(0, -)$	$(0, -)$
BOS	$(85, DC)$	$(85, DC)$
DC	$(10, BAL)$	$(10, BAL)$
NY	$(70, DC)$	$(34, PHL)$
PHL	$(17, BAL)$	$(17, BAL)$

Dijkstra's "Shortest" Path Algorithm

Baltimore to Boston



Round 4

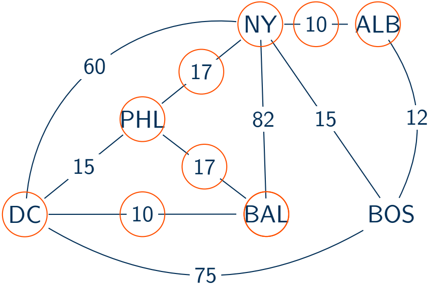
$v = NY$

$F = \{ALB, BOS\}$

City	Old Label	New Label
ALB	$(\infty, -)$	$(44, NY)$
BAL	$(0, -)$	$(0, -)$
BOS	$(85, DC)$	$(49, NY)$
DC	$(10, BAL)$	$(10, BAL)$
NY	$(34, PHL)$	$(34, PHL)$
PHL	$(17, BAL)$	$(17, BAL)$

Dijkstra's "Shortest" Path Algorithm

Baltimore to Boston



Round 5

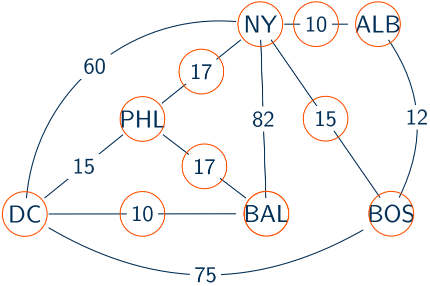
$v = ALB$

$F = \{BOS\}$

City	Old Label	New Label
ALB	$(44, NY)$	$(44, NY)$
BAL	$(0, -)$	$(0, -)$
BOS	$(49, NY)$	$(49, NY)$
DC	$(10, BAL)$	$(10, BAL)$
NY	$(34, PHL)$	$(34, PHL)$
PHL	$(17, BAL)$	$(17, BAL)$

Dijkstra's "Shortest" Path Algorithm

Baltimore to Boston



Round 6

$v = BOS$

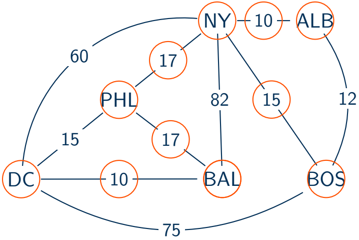
$F = \{\}$

City	Old Label	New Label
ALB	$(44, NY)$	$(44, NY)$
BAL	$(0, -)$	$(0, -)$
BOS	$(49, NY)$	$(49, NY)$
DC	$(10, BAL)$	$(10, BAL)$
NY	$(34, PHL)$	$(34, PHL)$
PHL	$(17, BAL)$	$(17, BAL)$

$$L(BOS) = (49, NY) \leftarrow L(NY) = (34, PHL) \leftarrow L(PHL) = (17, BAL)$$

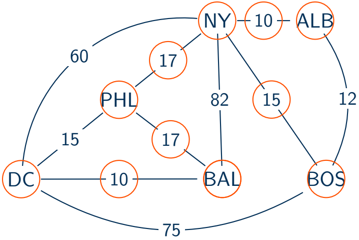
Dijkstra's "Shortest" Path Algorithm

- Start with vertex v at the start point



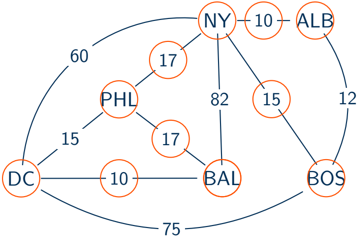
Dijkstra's "Shortest" Path Algorithm

- Start with vertex *v* at the start point
- Update the *label values* for vertices adjacent to *v*



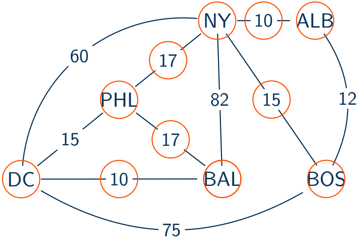
Dijkstra's "Shortest" Path Algorithm

- Start with vertex *v* at the start point
- Update the *label values* for vertices adjacent to *v*
- Update the *fringe F* by removing *v* and adding vertices adjacent to *v*



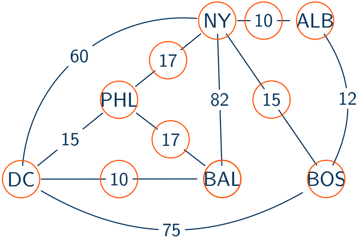
Dijkstra's "Shortest" Path Algorithm

- Start with vertex v at the start point
- Update the *label values* for vertices adjacent to v
- Update the *fringe* F by removing v and adding vertices adjacent to v
- Update v to be the vertex in F with the lowest label value



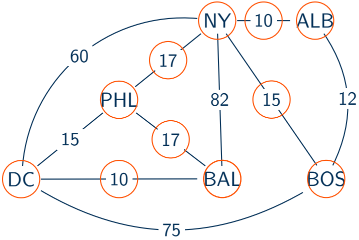
Dijkstra's "Shortest" Path Algorithm

- Start with vertex v at the start point
- Update the *label values* for vertices adjacent to v
- Update the *fringe* F by removing v and adding vertices adjacent to v
- Update v to be the vertex in F with the lowest label value
- Add the new v to the tree *along with the edge that let it achieve its minimal label value*

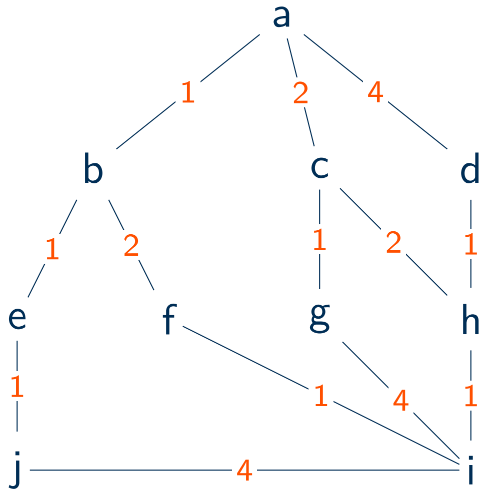


Dijkstra's "Shortest" Path Algorithm

- Start with vertex v at the start point
- Update the *label values* for vertices adjacent to v
- Update the *fringe* F by removing v and adding vertices adjacent to v
- Update v to be the vertex in F with the lowest label value
- Add the new v to the tree *along with the edge that let it achieve its minimal label value*
- Repeat until you reach the destination



One More Example



Trees

Dr. Chuck Rocca
roccac@wcsu.edu
http://sites.wcsu.edu/roccac

WESTERN CONNECTICUT
STATE UNIVERSITY
MACRICOSTAS
SCHOOL OF ARTS
& SCIENCES