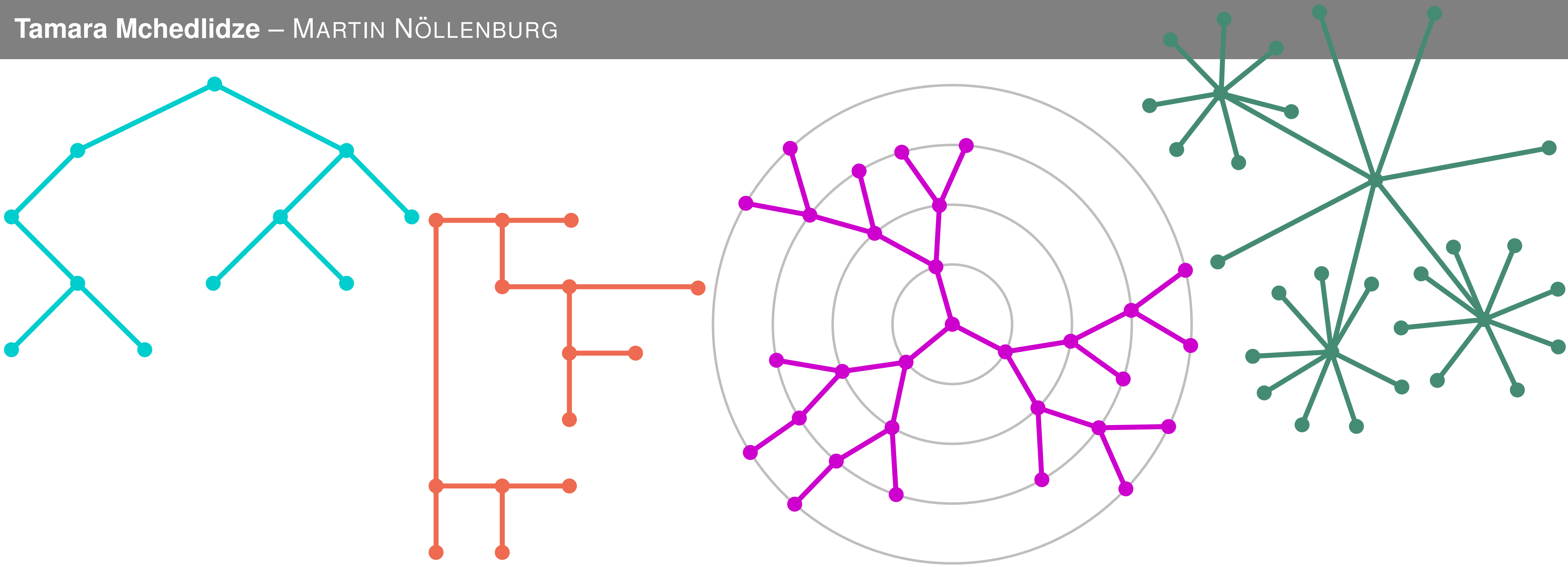
Algorithms for graph visualization

Divide and Conquer - Tree Layouts

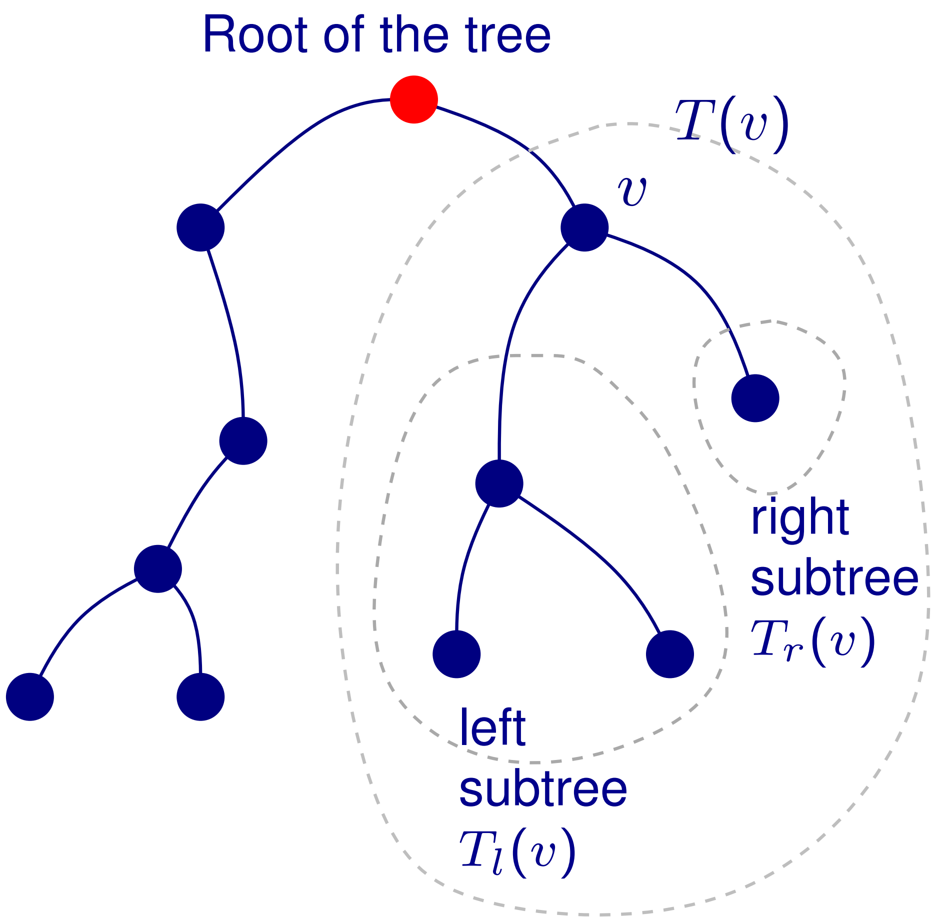
WINTER SEMESTER 2013/2014

Tamara Mchedlidze – MARTIN NÖLLENBURG



Basic Definitions

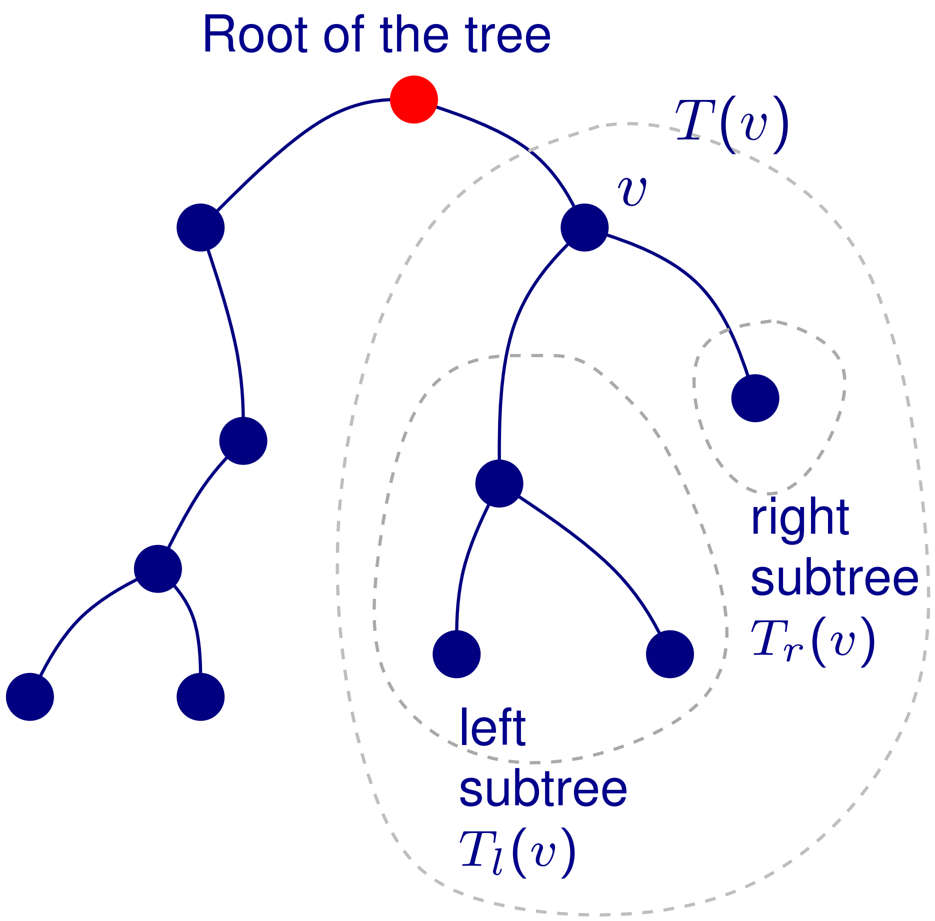
- Tree, Binary Tree



Basic Definitions

- Tree, Binary Tree

Tree traversals

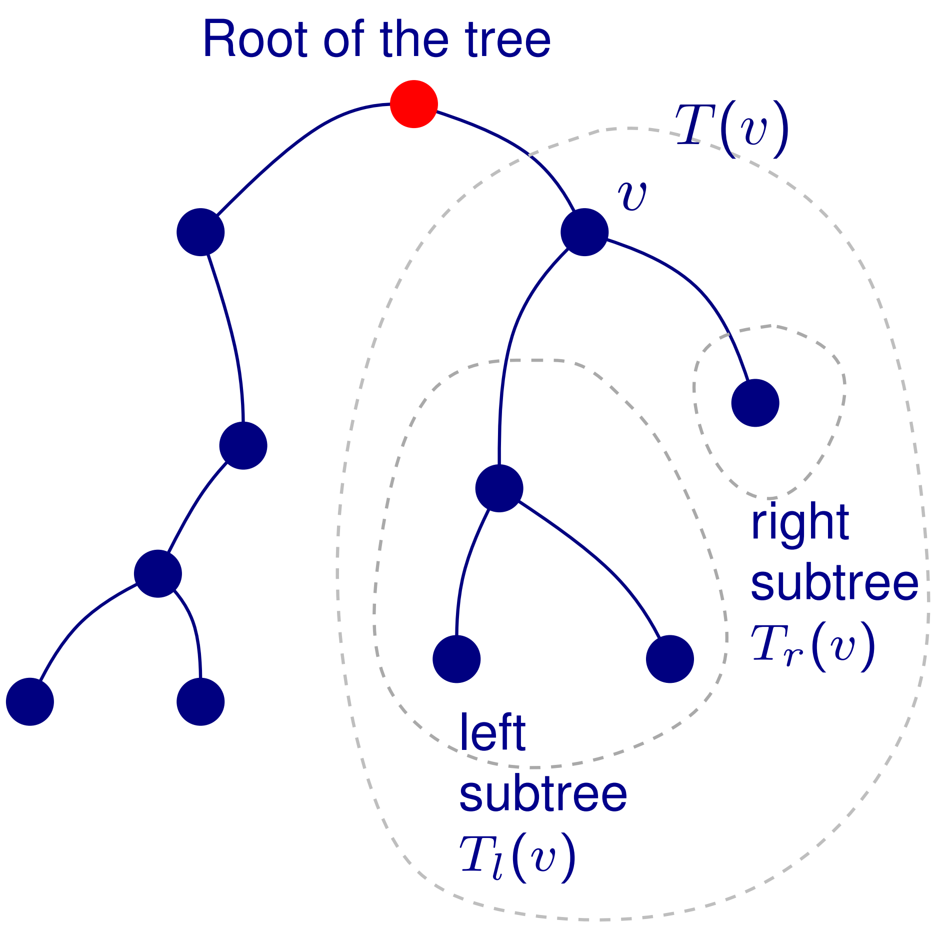


Basic Definitions

- Tree, Binary Tree

Tree traversals

Depth-first search



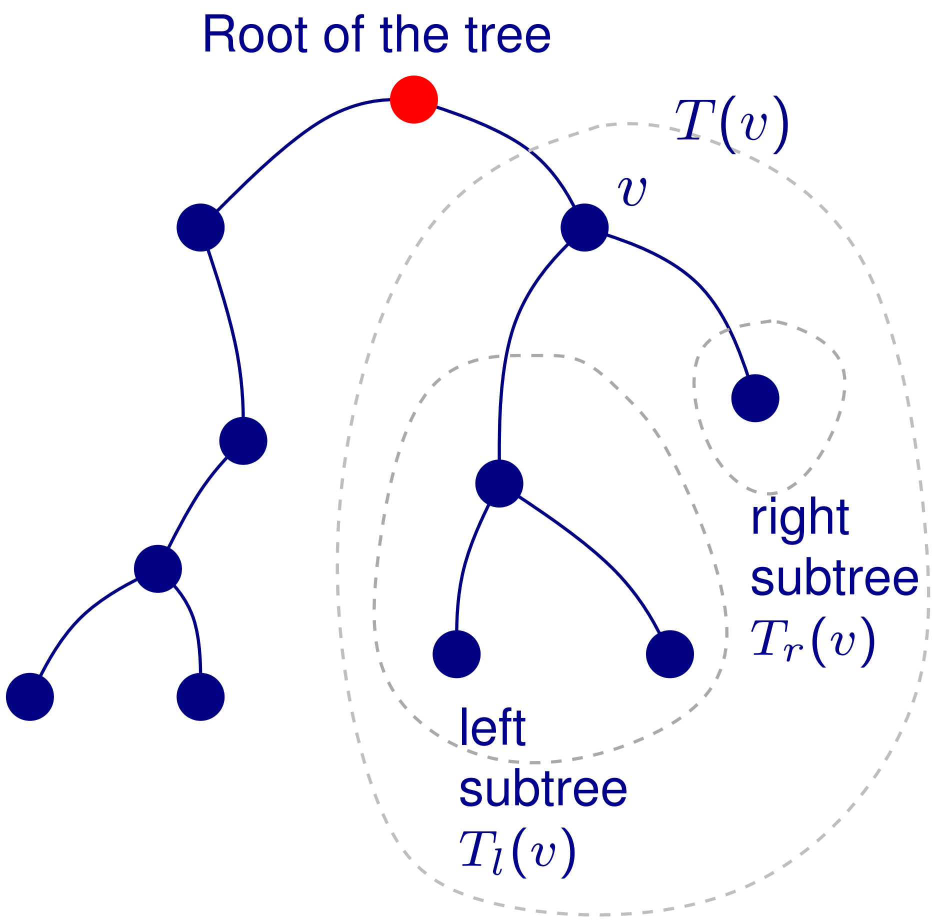
Basic Definitions

- Tree, Binary Tree

Tree traversals

Depth-first search

- Pre-order (First parent, then subtrees)
- In-order (Left child, parent, right child)
- Post-order (First subtrees, then parent)



Basic Definitions

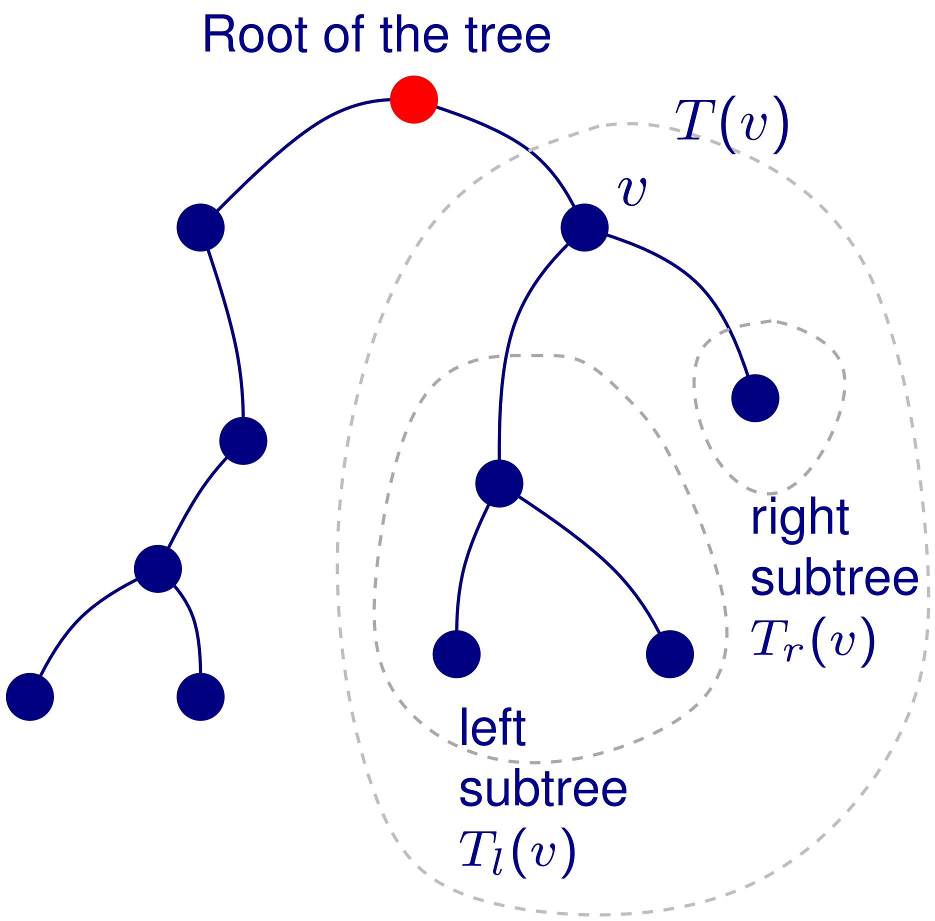
- Tree, Binary Tree

Tree traversals

Depth-first search

- Pre-order (First parent, then subtrees)
- In-order (Left child, parent, right child)
- Post-order (First subtrees, then parent)

Breadth-first search



Basic Definitions

- Tree, Binary Tree

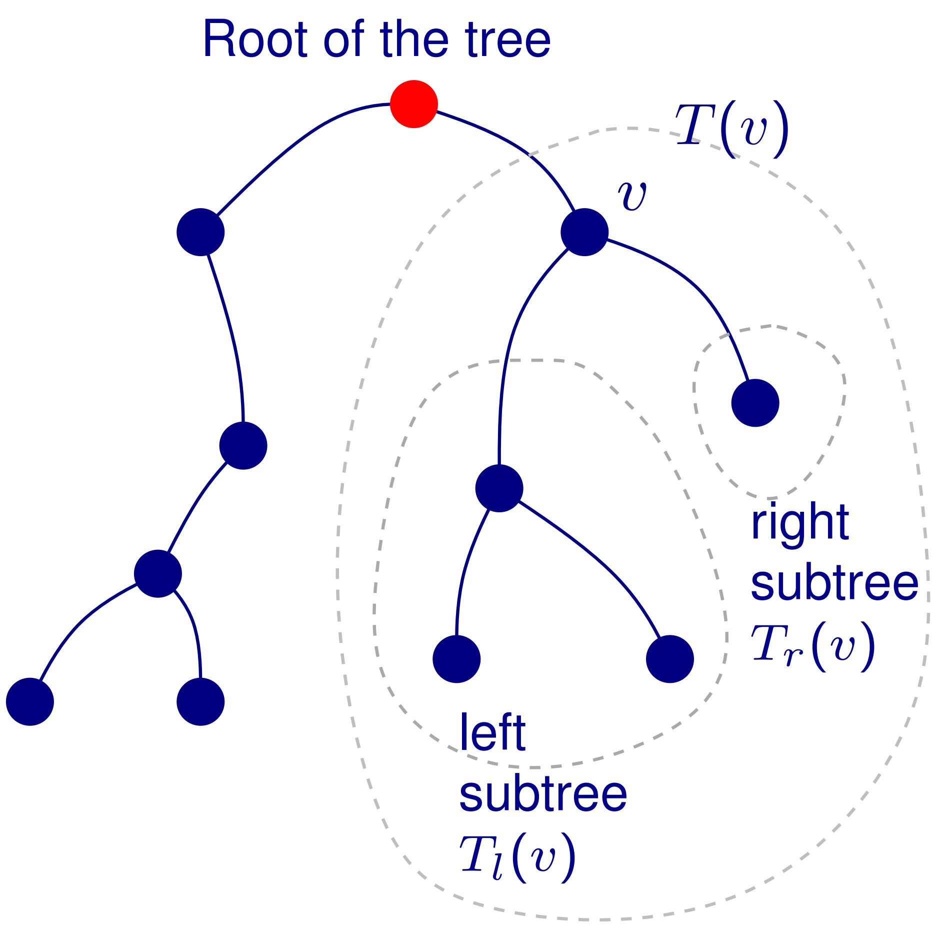
Tree traversals

Depth-first search

- Pre-order (First parent, then subtrees)
- In-order (Left child, parent, right child)
- Post-order (First subtrees, then parent)

Breadth-first search

- Assignes vertices to layers



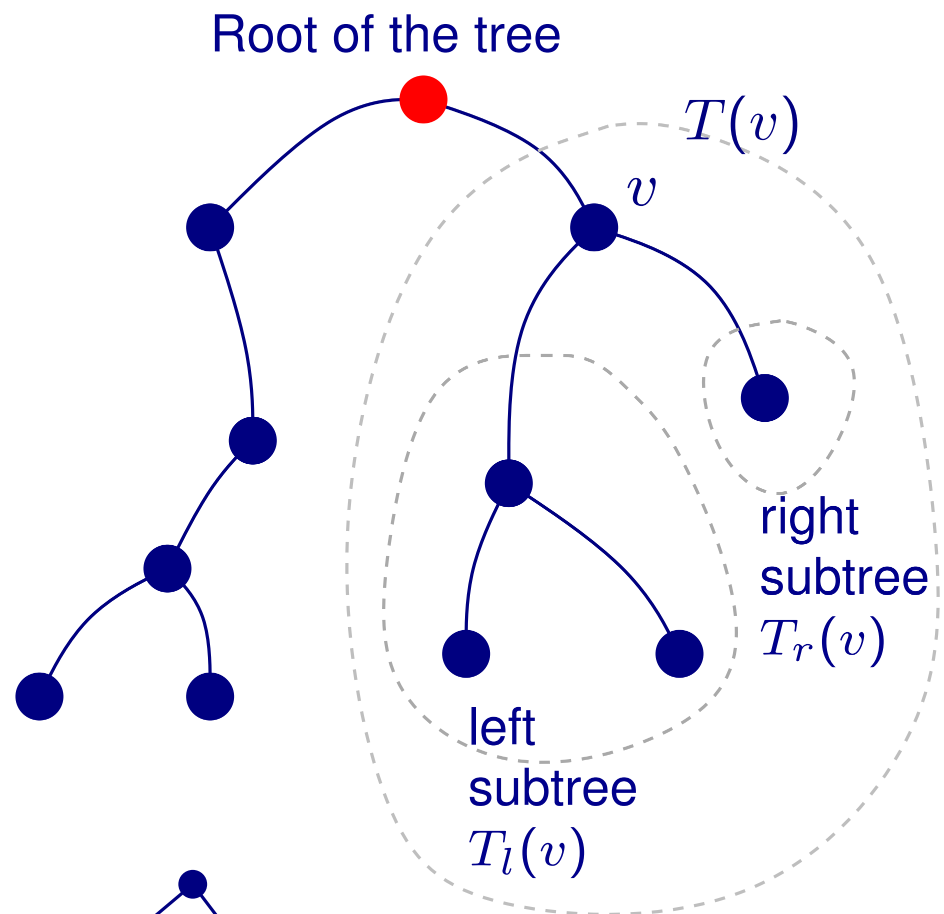
Basic Definitions

- Tree, Binary Tree

Tree traversals

Depth-first search

- Pre-order (First parent, then subtrees)
- In-order (Left child, parent, right child)
- Post-order (First subtrees, then parent)



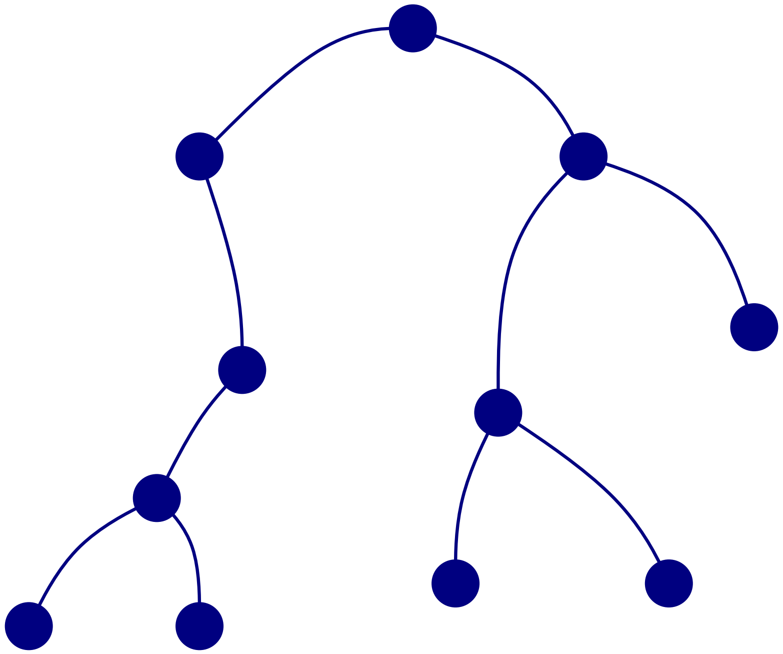
Breadth-first search

- Assignes vertices to layers

Simply and Axially isomorpthic trees

Drawing of a Tree

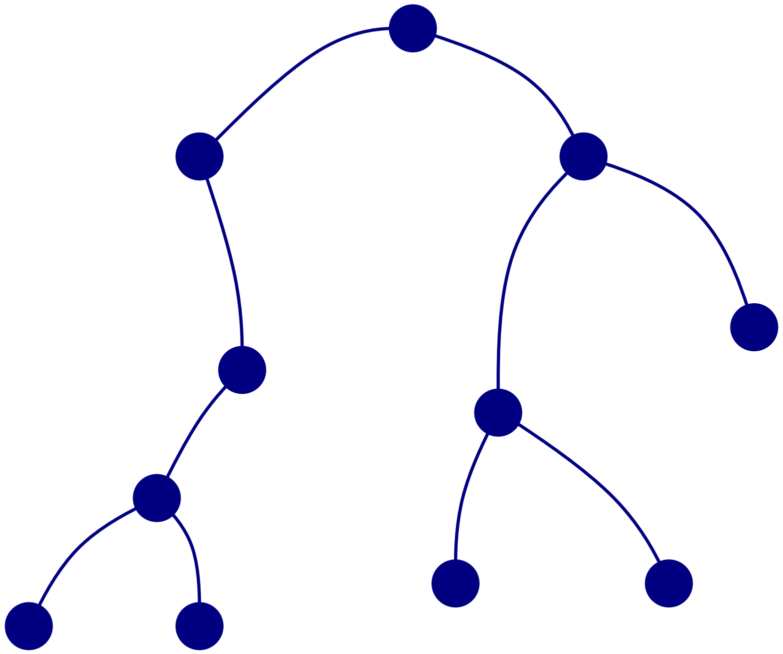
Given: A rooted binary tree



Drawing of a Tree

Given: A rooted binary tree

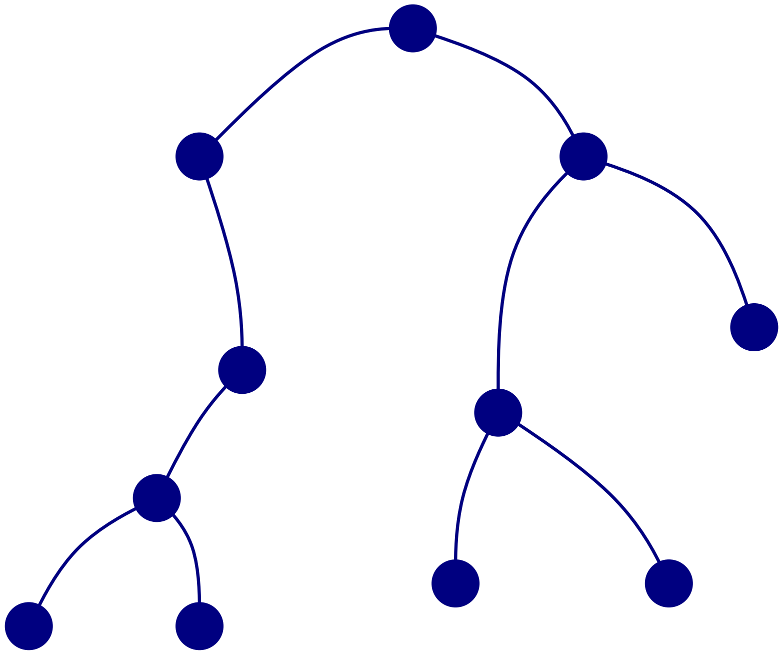
Question: How would we draw it?



Drawing of a Tree

Given: A rooted binary tree

Question: How would we draw it?

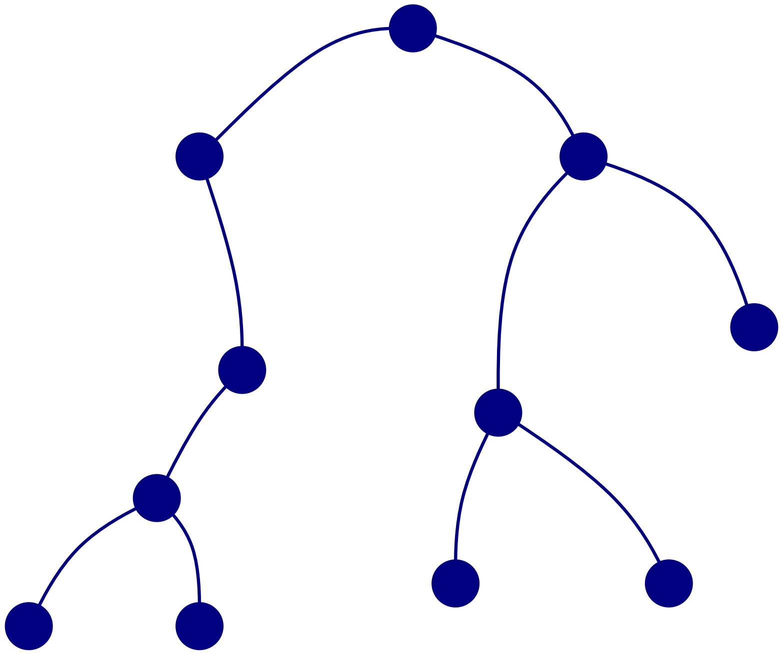


Drawing of a Tree

Given: A rooted binary tree

Question: How would we draw it?

How would look like an algorithms that draws it?

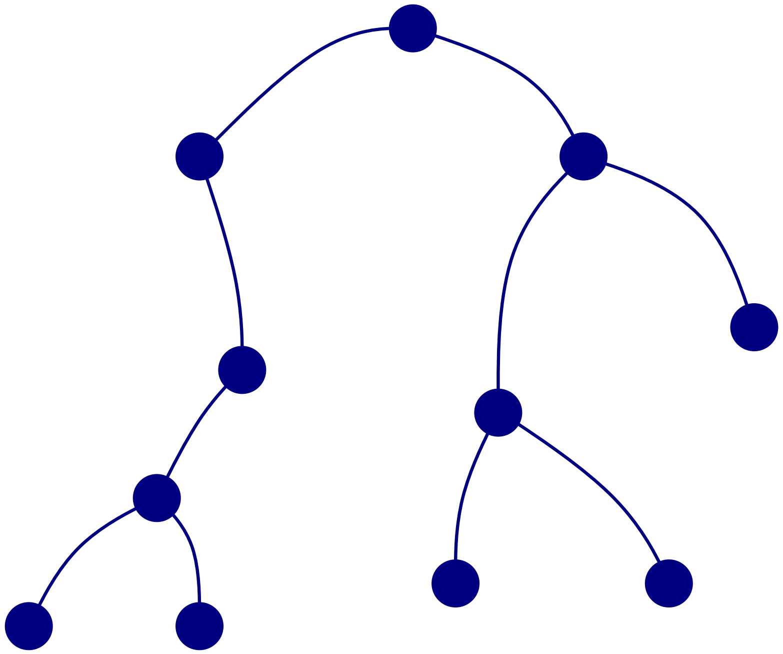


Drawing of a Tree

Given: A rooted binary tree

Question: How would we draw it?

How would look like an algorithms that draws it?

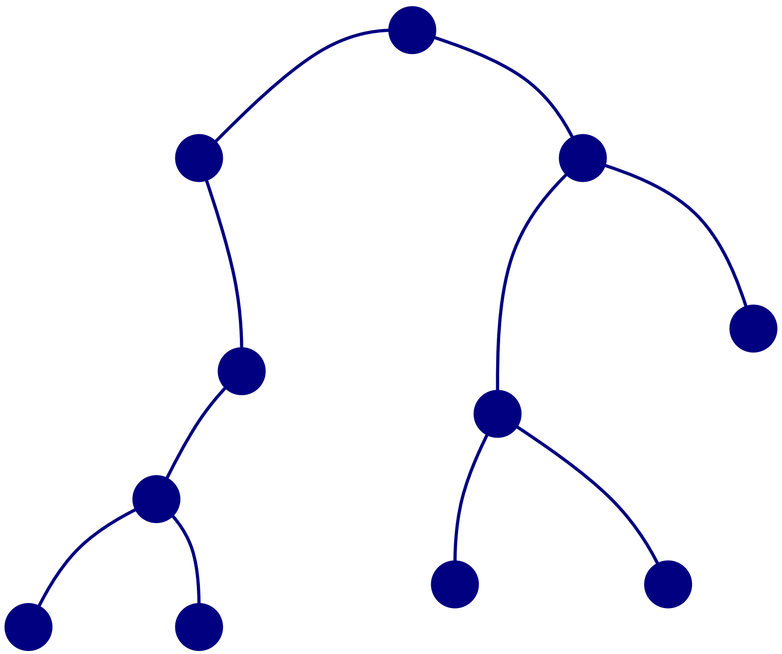


How to draw a tree

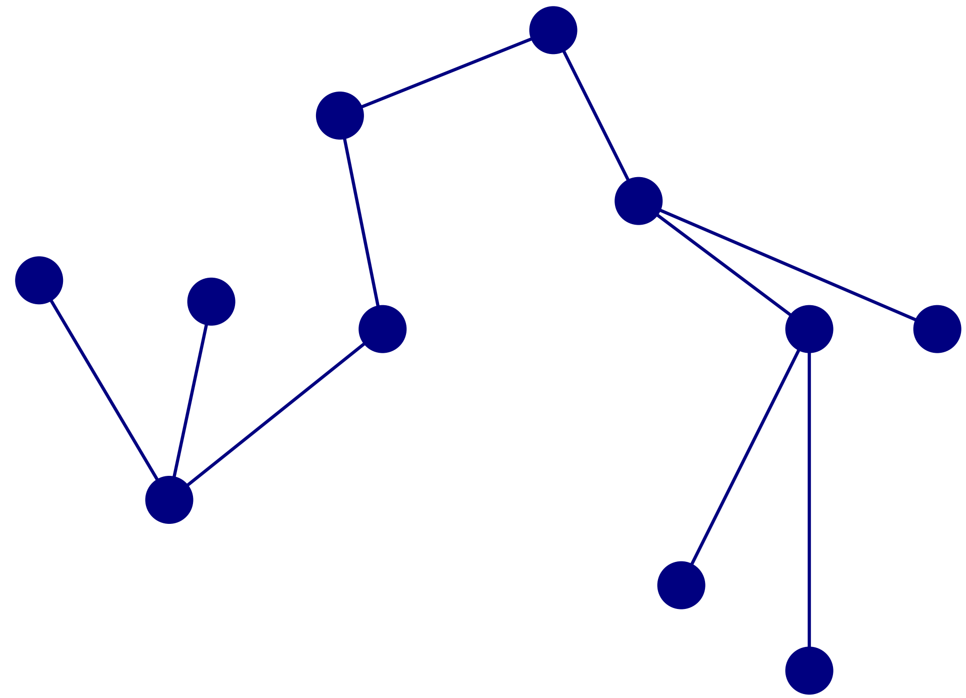
- Draw vertices as circles
  - Assign to the vertices x and y-coordinates
- Connect them by straight-line segments

A Nice Drawing of a Tree

Input (rooted binary tree)

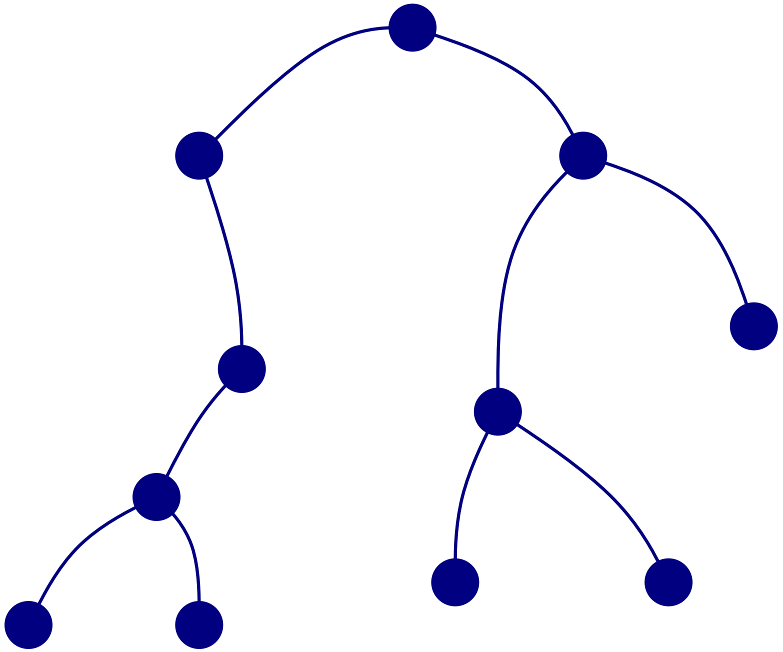


Output

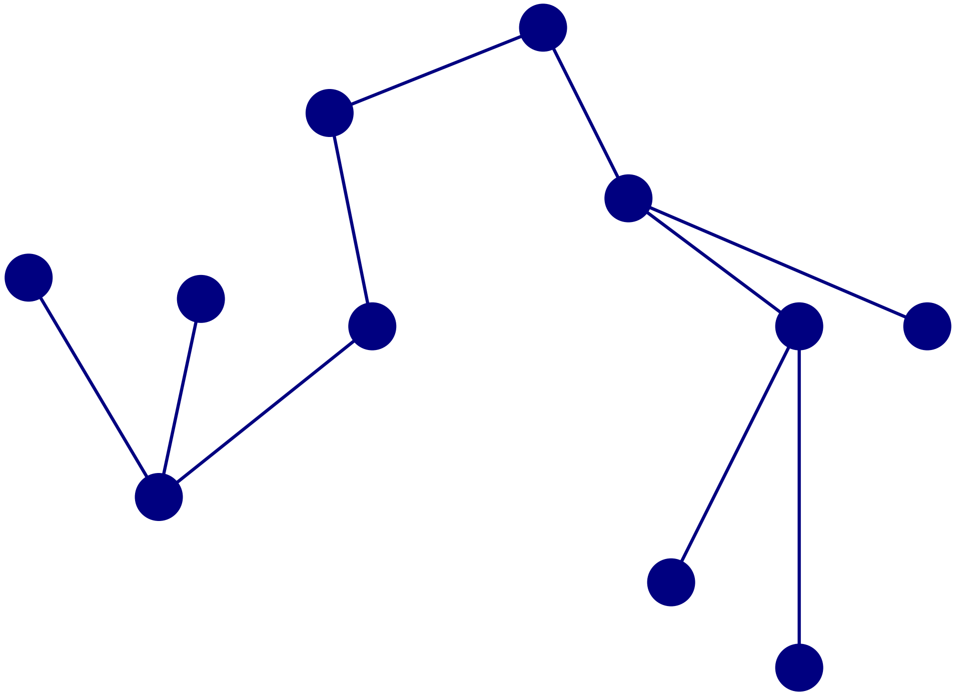


A Nice Drawing of a Tree

Input (rooted binary tree)



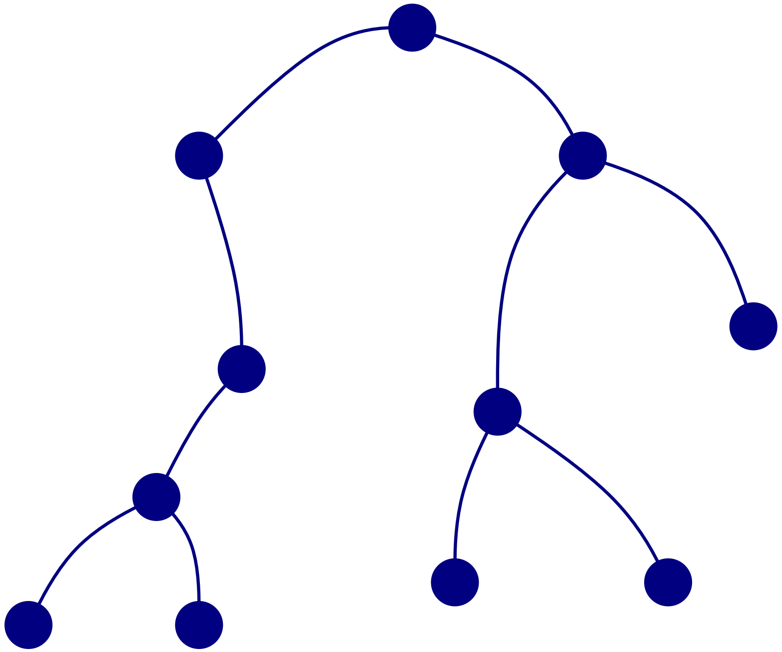
Output



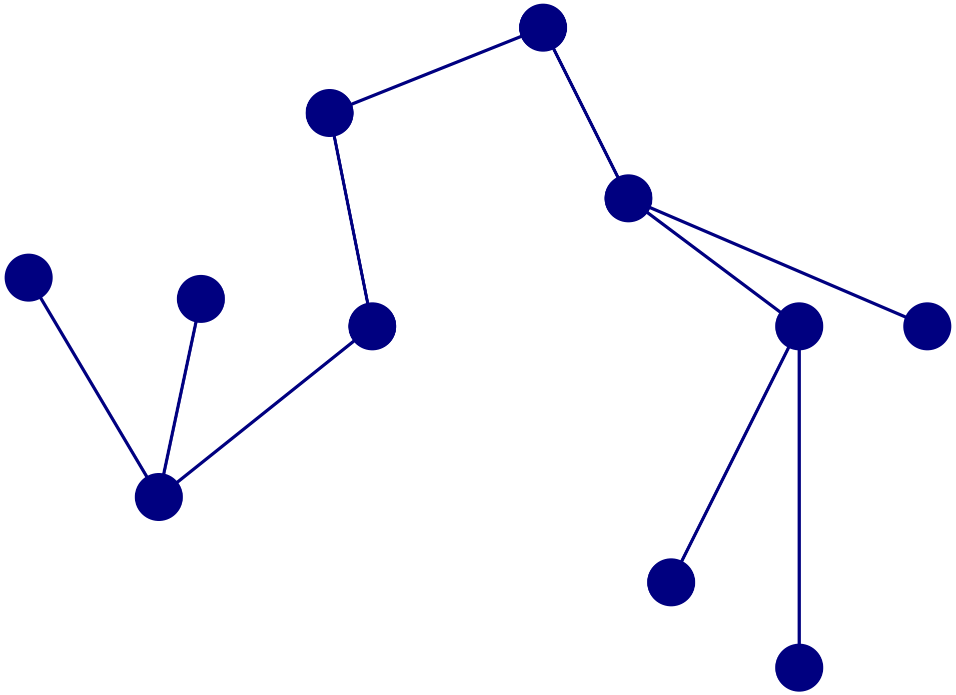
- Are we happy with such a drawing?

A Nice Drawing of a Tree

Input (rooted binary tree)



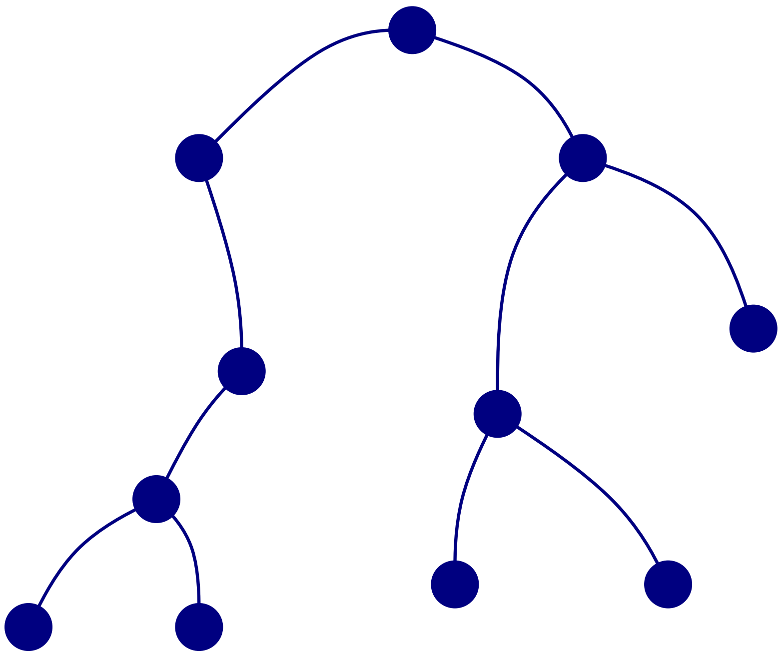
Output



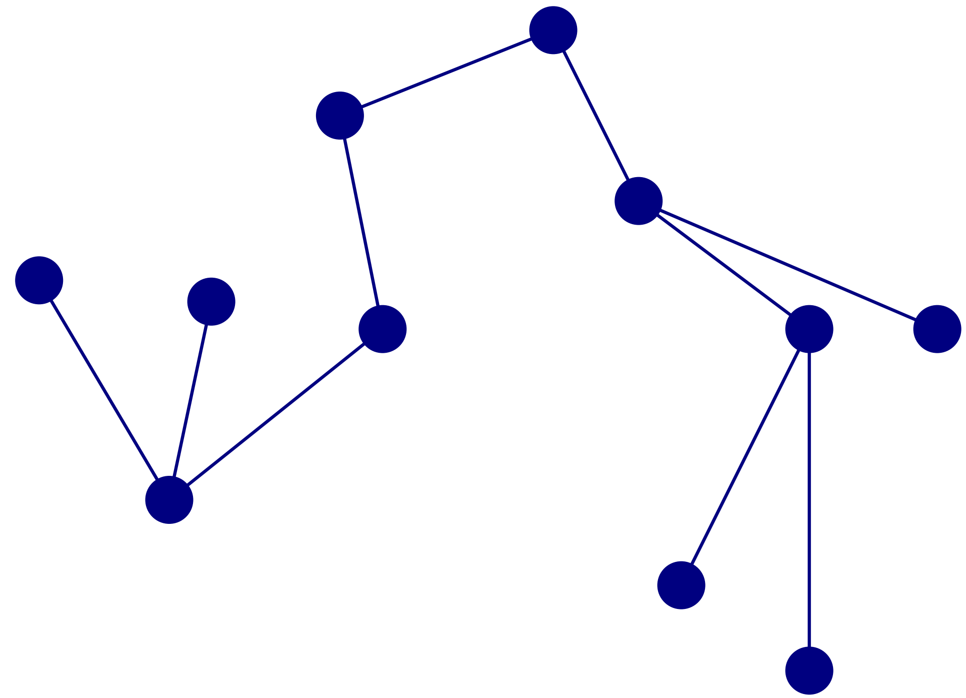
- Are we happy with such a drawing? Probably not...

A Nice Drawing of a Tree

Input (rooted binary tree)



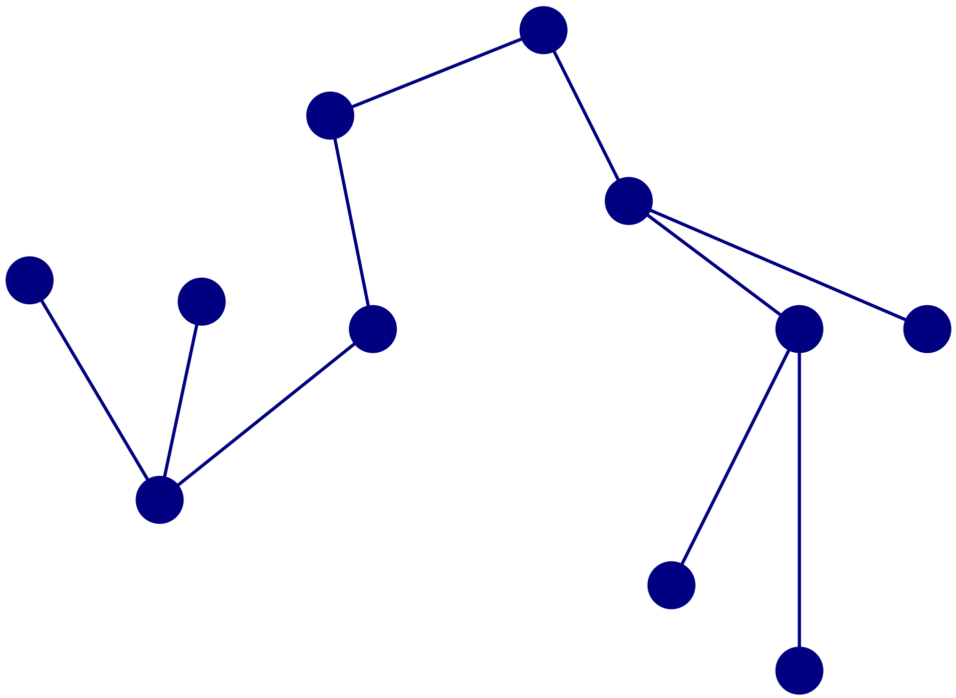
Output



- Are we happy with such a drawing? Probably not...
- We need **rules** which capture the notion of an admissible drawing of a binary tree... (Drawing conventions)

A Nice Drawing of a Tree

Output

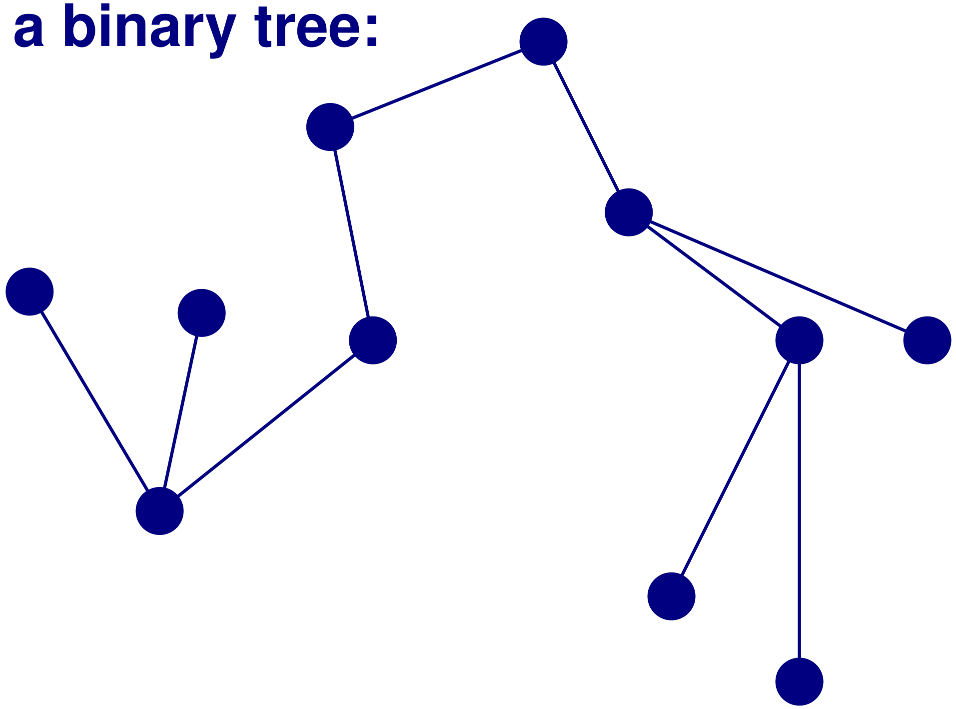


- Are we happy with such a drawing? Probably not...
- We need **rules** which capture the notion of an admissible drawing of a binary tree... (Drawing conventions)

A Nice Drawing of a Tree

Output

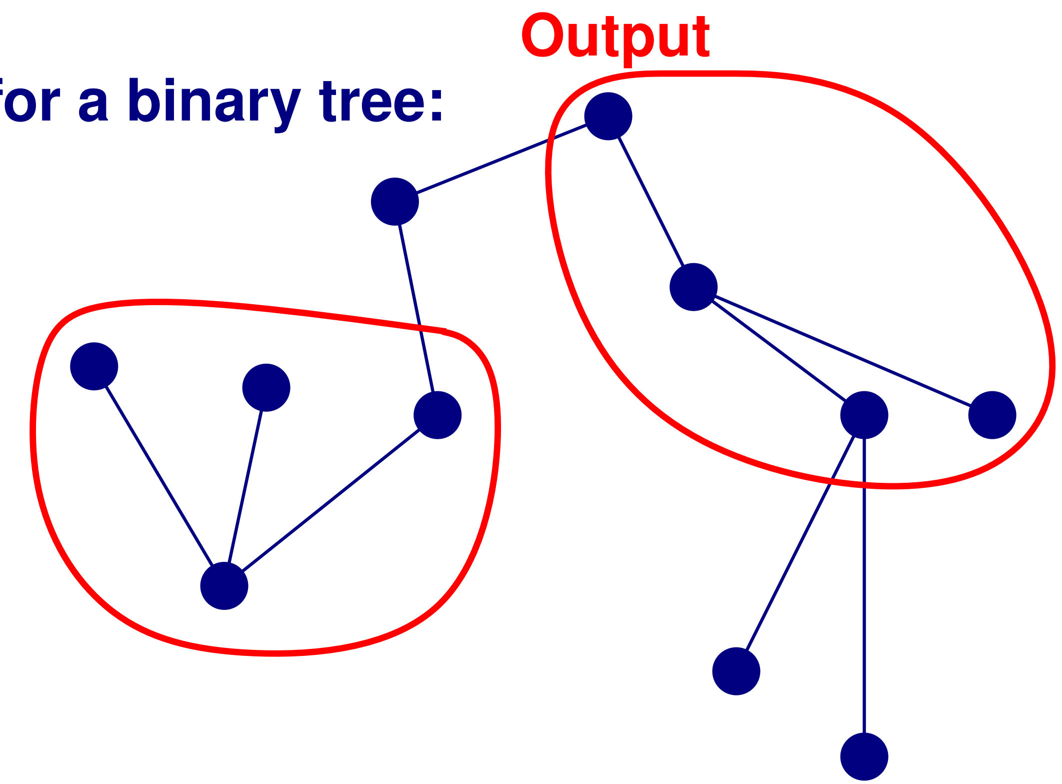
Possible drawing conventions for a binary tree:



- Are we happy with such a drawing? Probably not...
- We need **rules** which capture the notion of an admissible drawing of a binary tree... (Drawing conventions)

A Nice Drawing of a Tree

Possible drawing conventions for a binary tree:

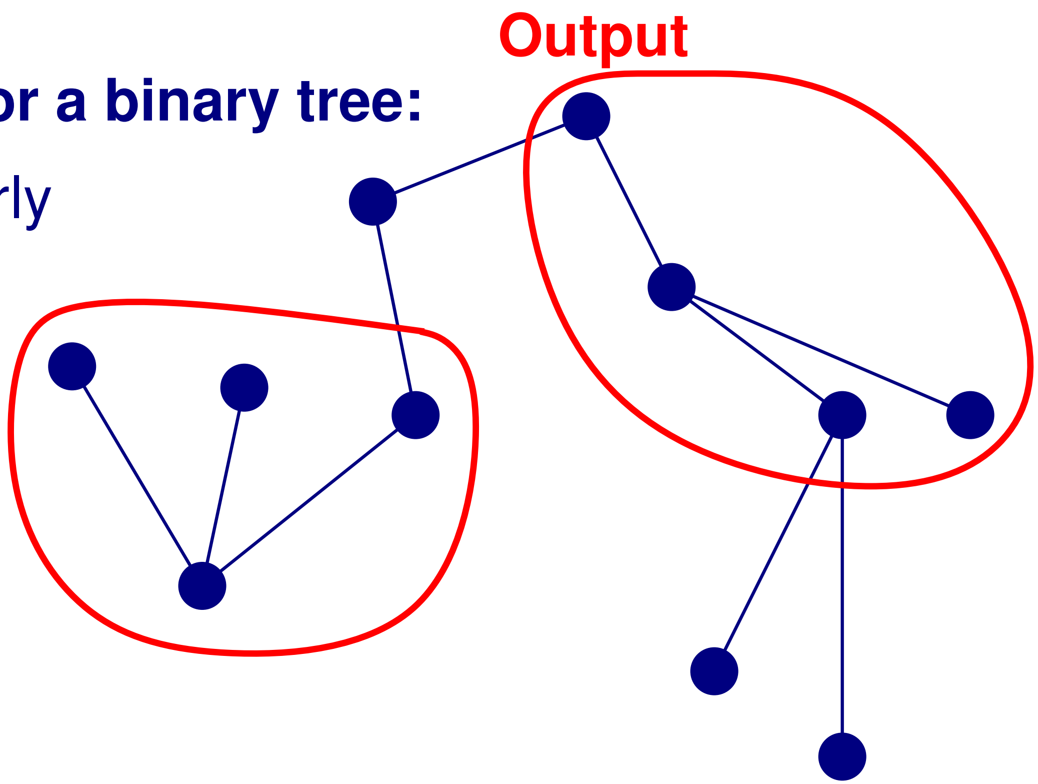


- Are we happy with such a drawing? Probably not...
- We need **rules** which capture the notion of an admissible drawing of a binary tree... (Drawing conventions)

A Nice Drawing of a Tree

Possible drawing conventions for a binary tree:

- Similar trees are drawn similarly

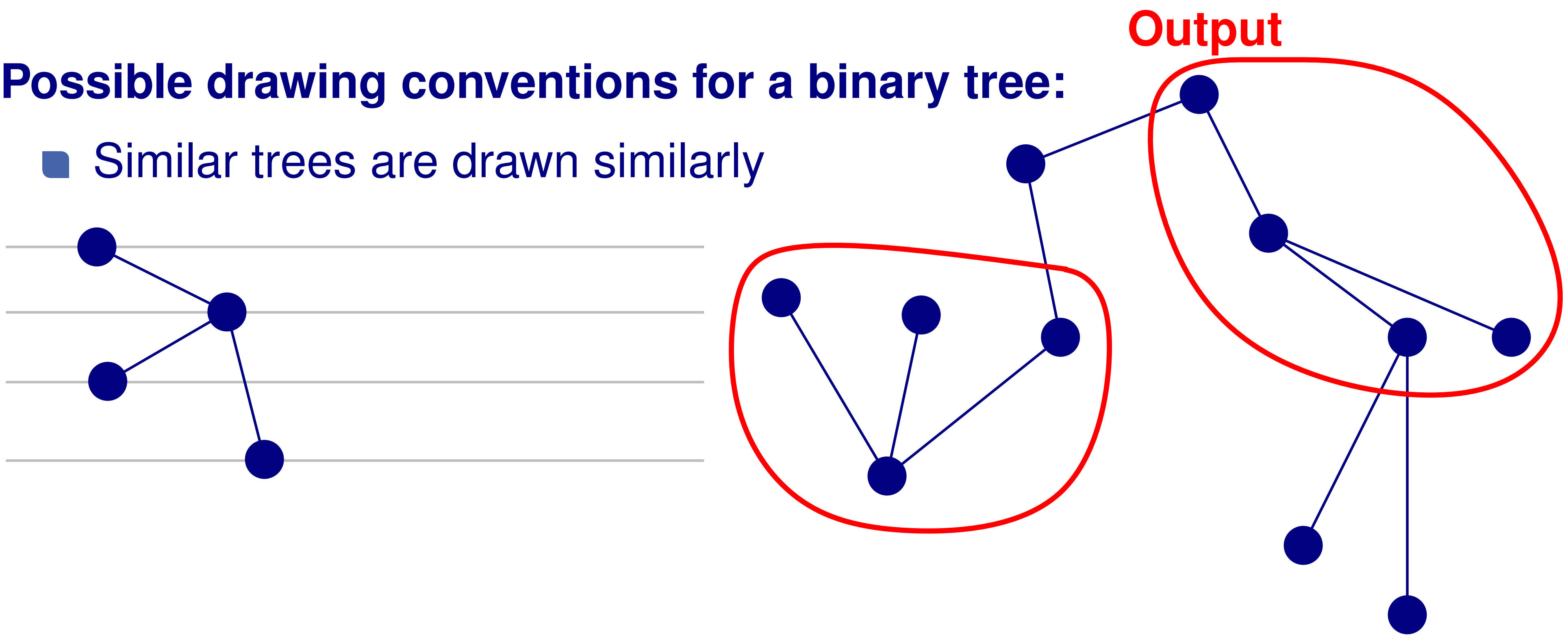


- Are we happy with such a drawing? Probably not...
- We need **rules** which capture the notion of an admissible drawing of a binary tree... (Drawing conventions)

A Nice Drawing of a Tree

Possible drawing conventions for a binary tree:

- Similar trees are drawn similarly

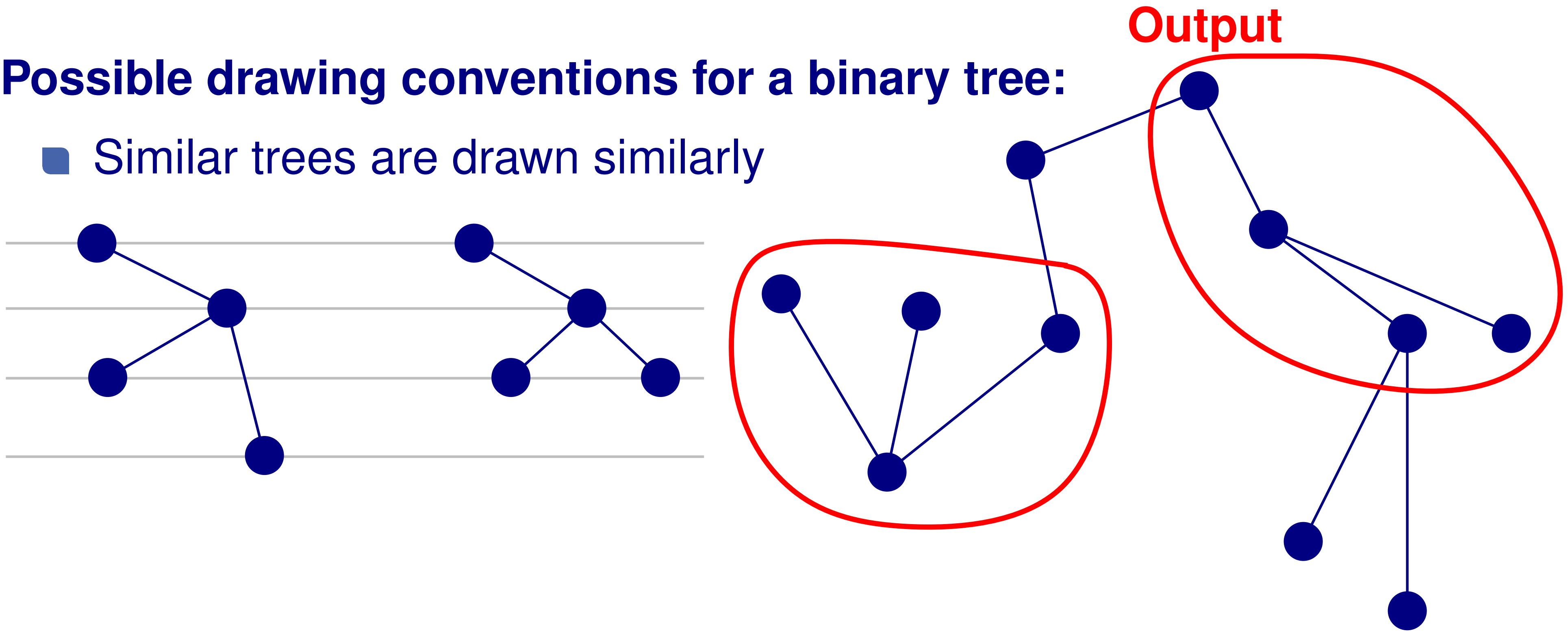


- Are we happy with such a drawing? Probably not...
- We need **rules** which capture the notion of an admissible drawing of a binary tree... (Drawing conventions)

A Nice Drawing of a Tree

Possible drawing conventions for a binary tree:

- Similar trees are drawn similarly

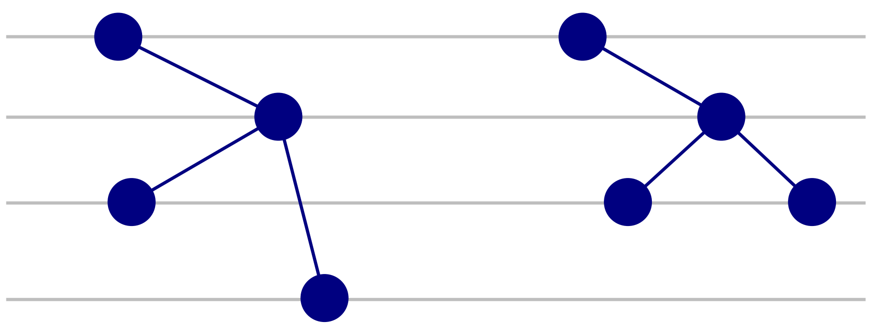


- Are we happy with such a drawing? Probably not...
- We need **rules** which capture the notion of an admissible drawing of a binary tree... (Drawing conventions)

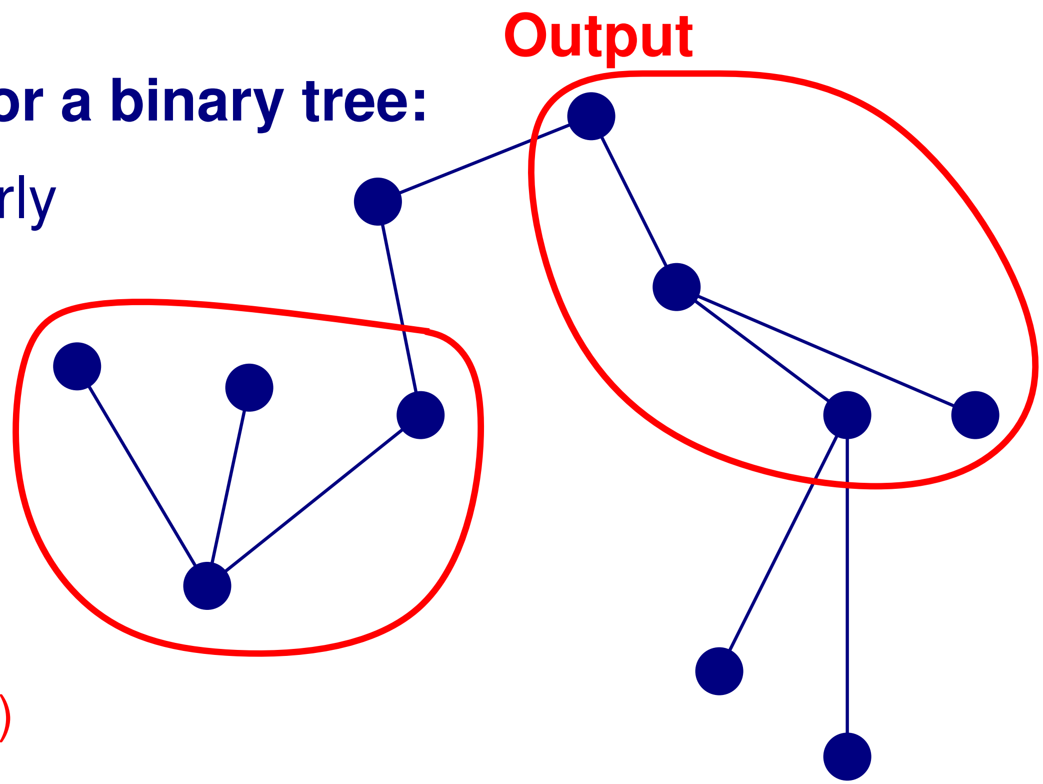
A Nice Drawing of a Tree

Possible drawing conventions for a binary tree:

- Similar trees are drawn similarly



- Vertices are plased on layers
  (layered drawing)

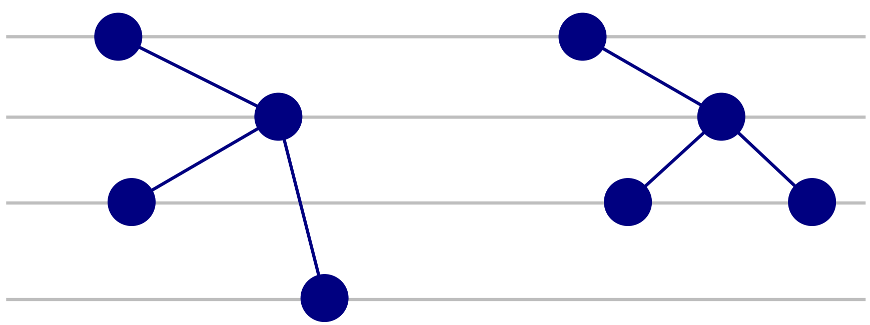


- Are we happy with such a drawing? Probably not...
- We need **rules** which capture the notion of an admissible drawing of a binary tree... (Drawing conventions)

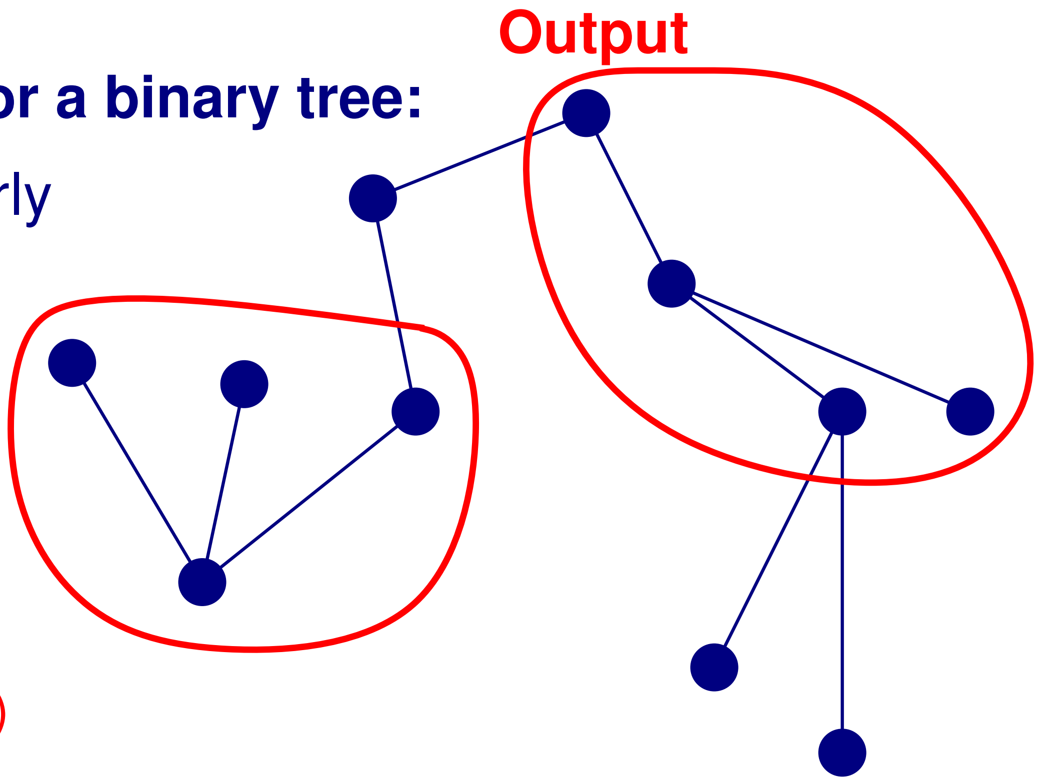
A Nice Drawing of a Tree

Possible drawing conventions for a binary tree:

- Similar trees are drawn similarly



- Vertices are plased on layers
  (layered drawing)
- A parent is centered with respect to its children



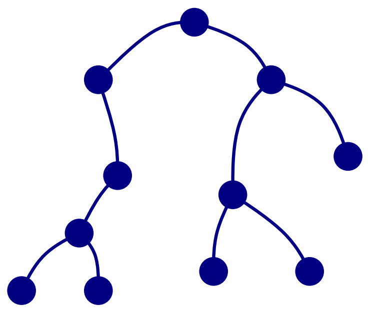
- Are we happy with such a drawing? Probably not...
- We need **rules** which capture the notion of an admissible drawing of a binary tree... (Drawing conventions)

Reingold & Tilford

Algorithm Outline:

Input: A binary tree

Output: A layered drawing of T



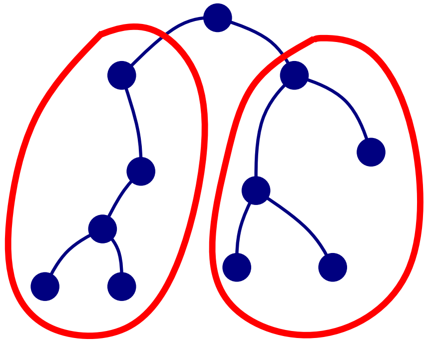
Base case: A single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:

Reingold & Tilford

Algorithm Outline:



Input: A binary tree

Output: A layered drawing of T

Base case: A single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

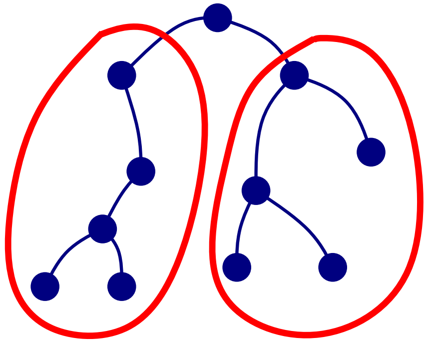
Conquer:

Reingold & Tilford

Algorithm Outline:

Input: A binary tree

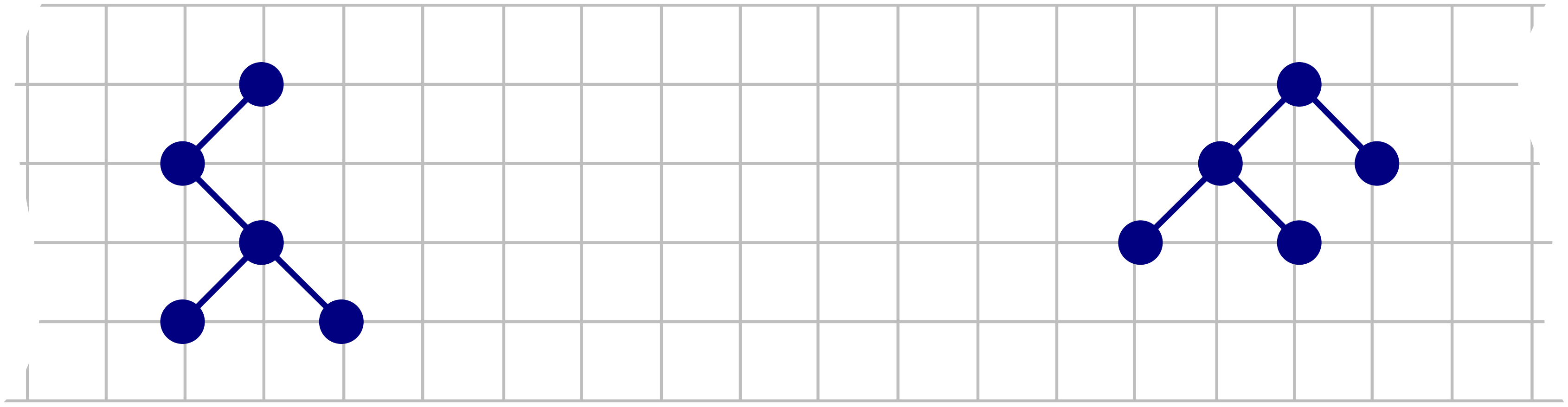
Output: A layered drawing of T



Base case: A single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:

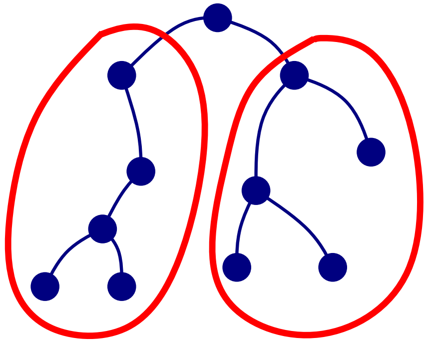


Reingold & Tilford

Algorithm Outline:

Input: A binary tree

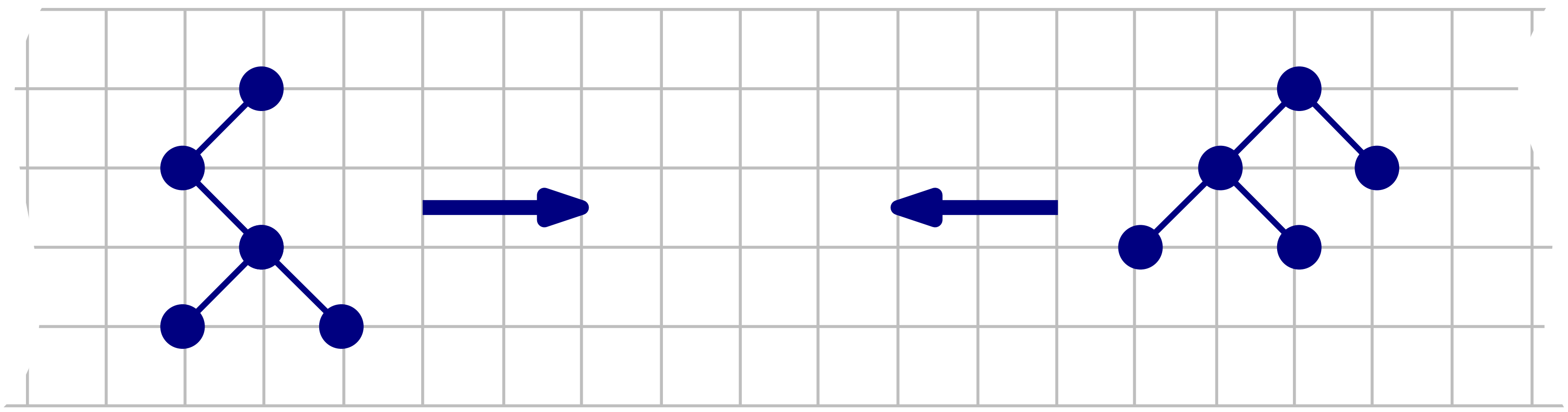
Output: A layered drawing of T



Base case: A single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:

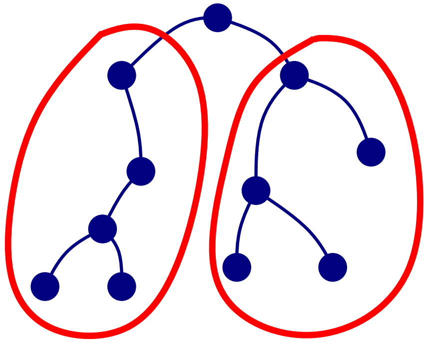


Reingold & Tilford

Algorithm Outline:

Input: A binary tree

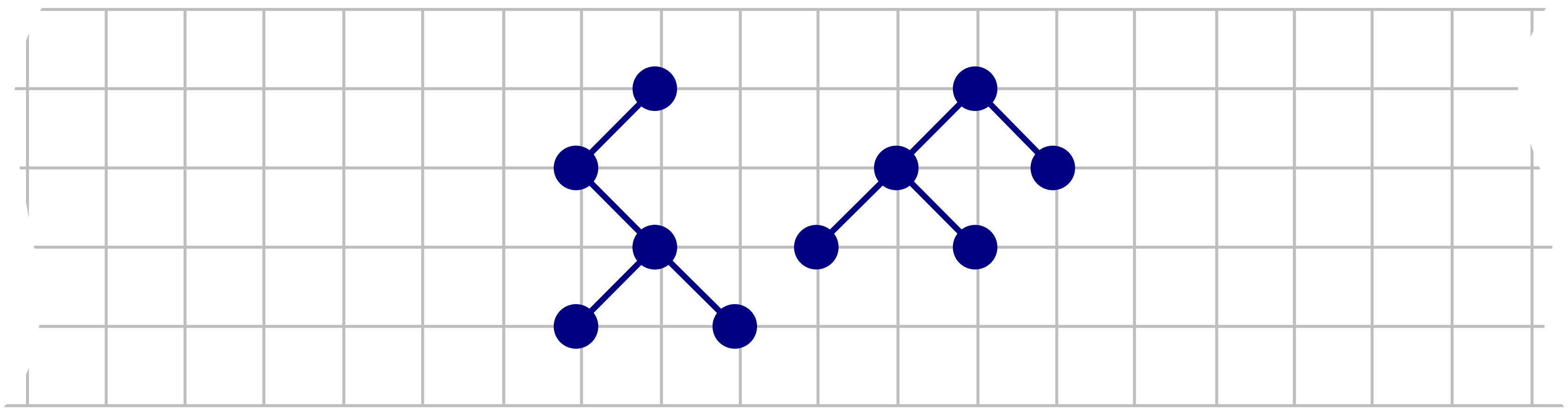
Output: A layered drawing of T



Base case: A single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:

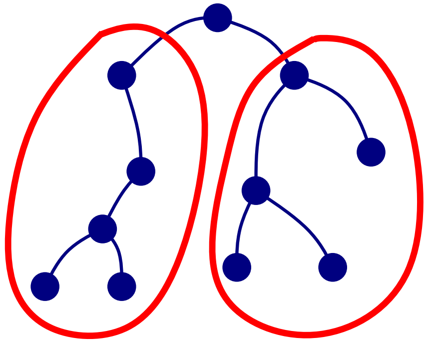


Reingold & Tilford

Algorithm Outline:

Input: A binary tree

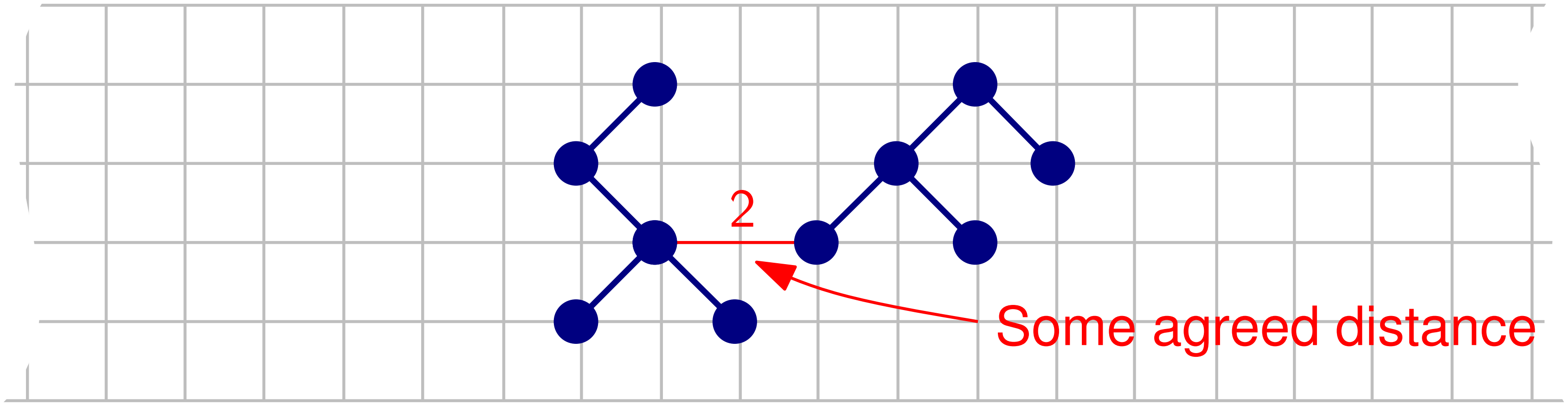
Output: A layered drawing of T



Base case: A single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:

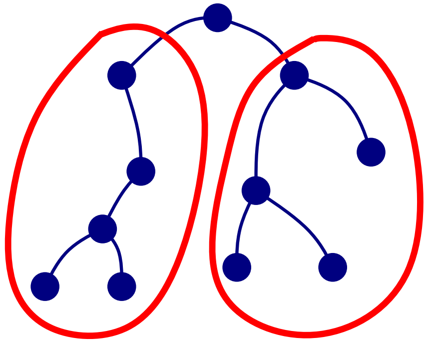


Reingold & Tilford

Algorithm Outline:

Input: A binary tree

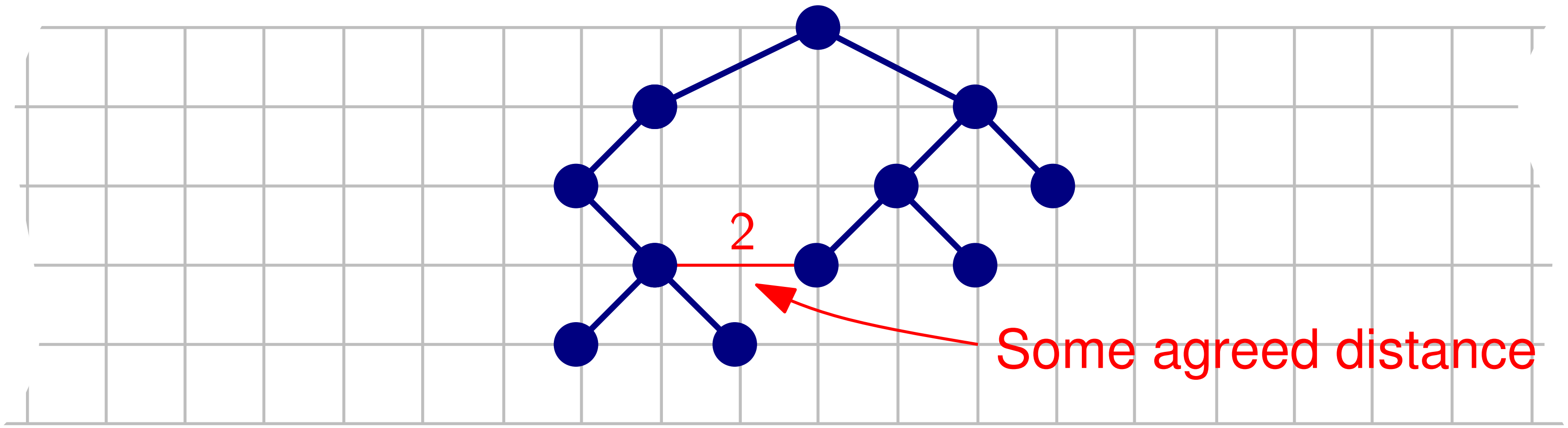
Output: A layered drawing of T



Base case: A single vertex

Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:

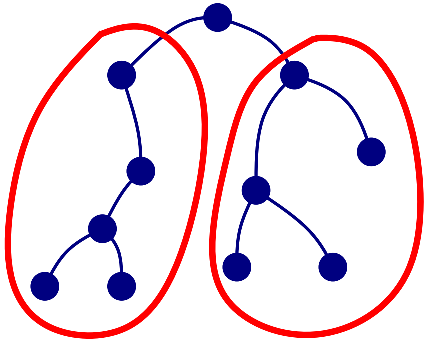


Reingold & Tilford

Algorithm Outline:

Input: A binary tree

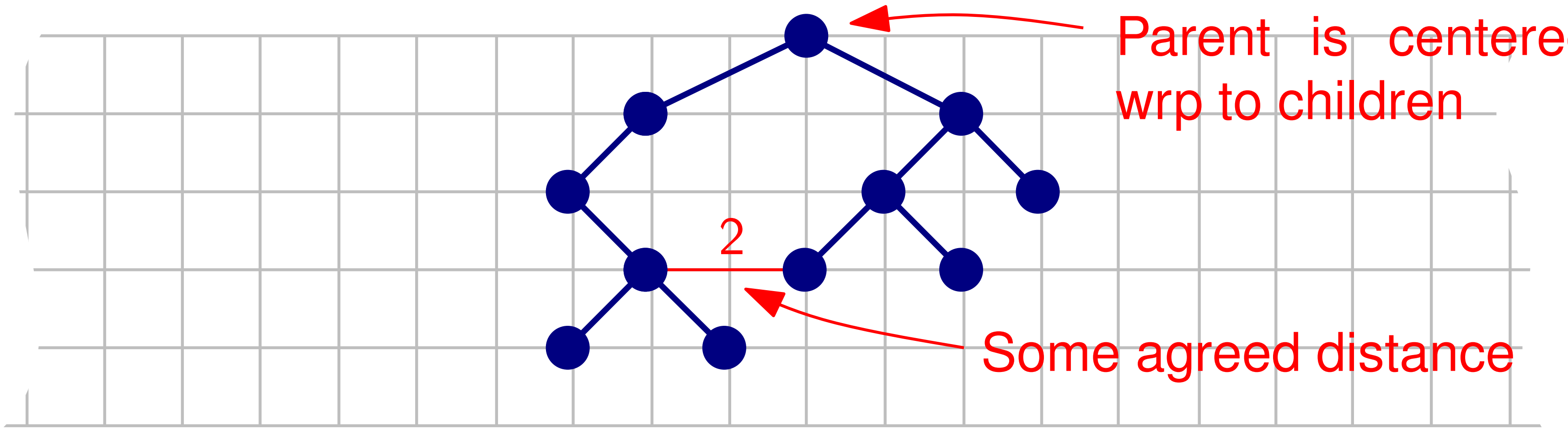
Output: A layered drawing of T



Base case: A single vertex

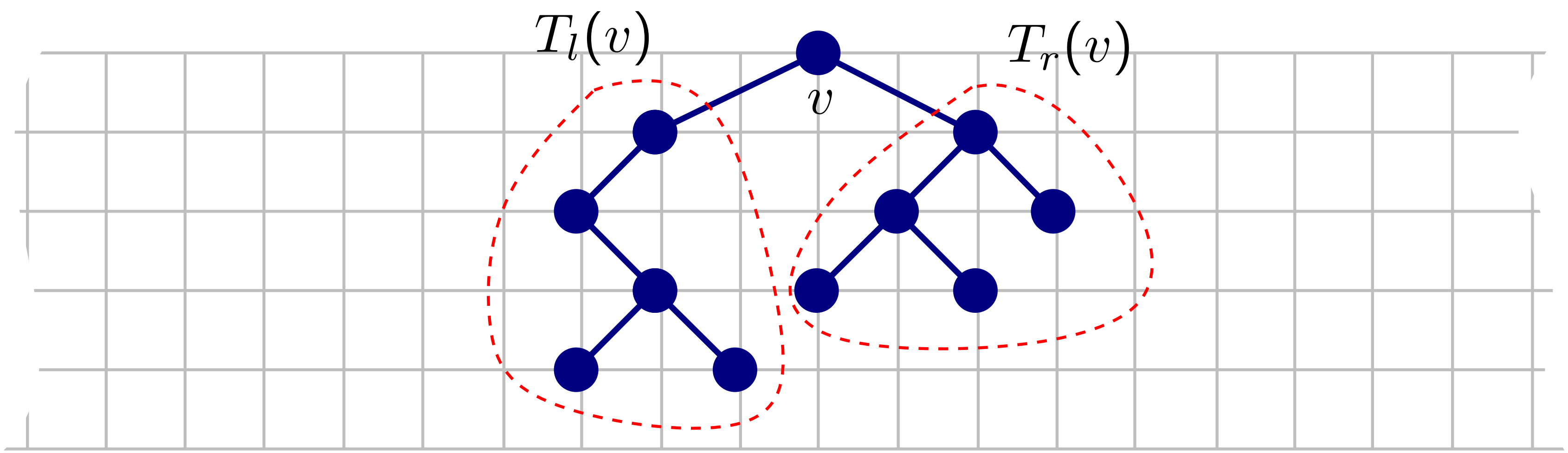
Divide: Recursively apply the algorithm to draw the left and the right subtrees of T

Conquer:



Implementation Details (postorder and preorder traversals)

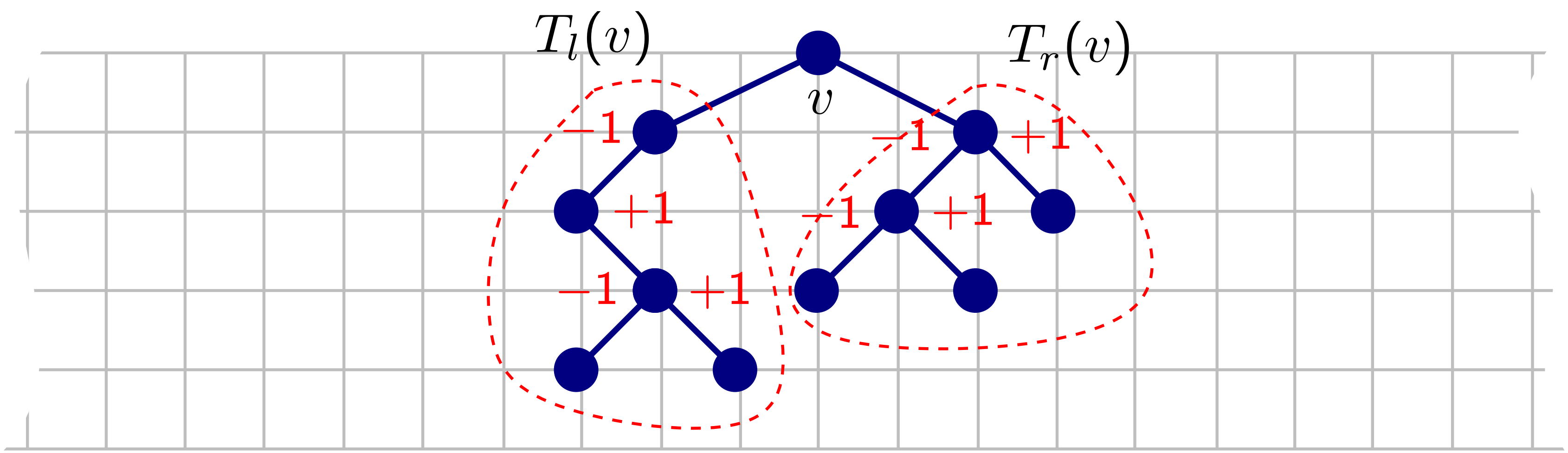
Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child



Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

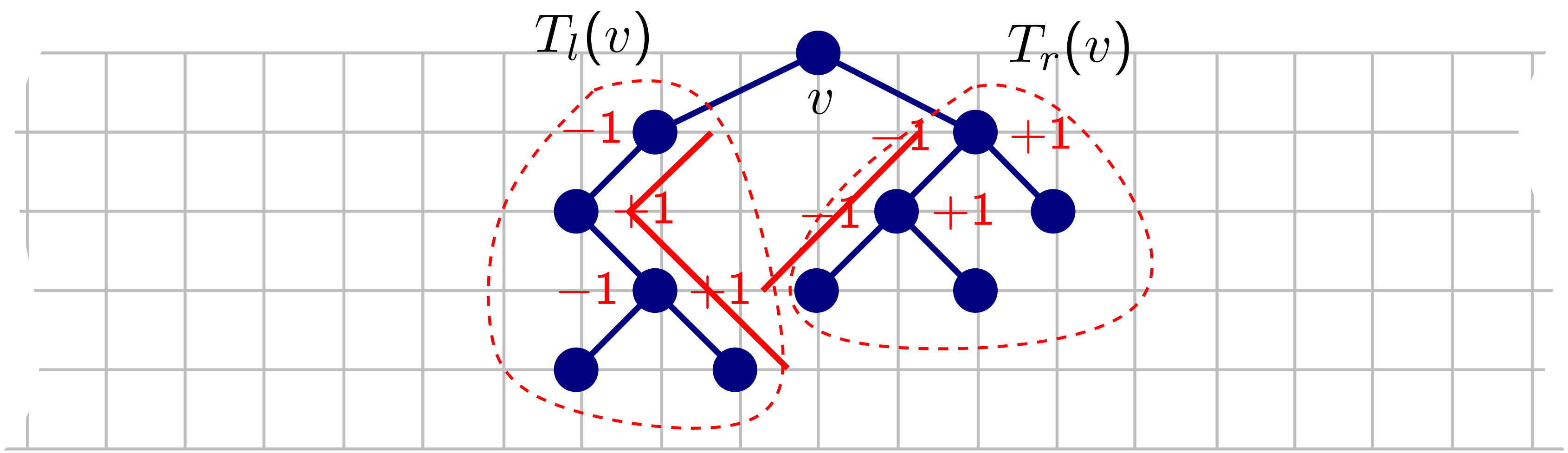


Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$

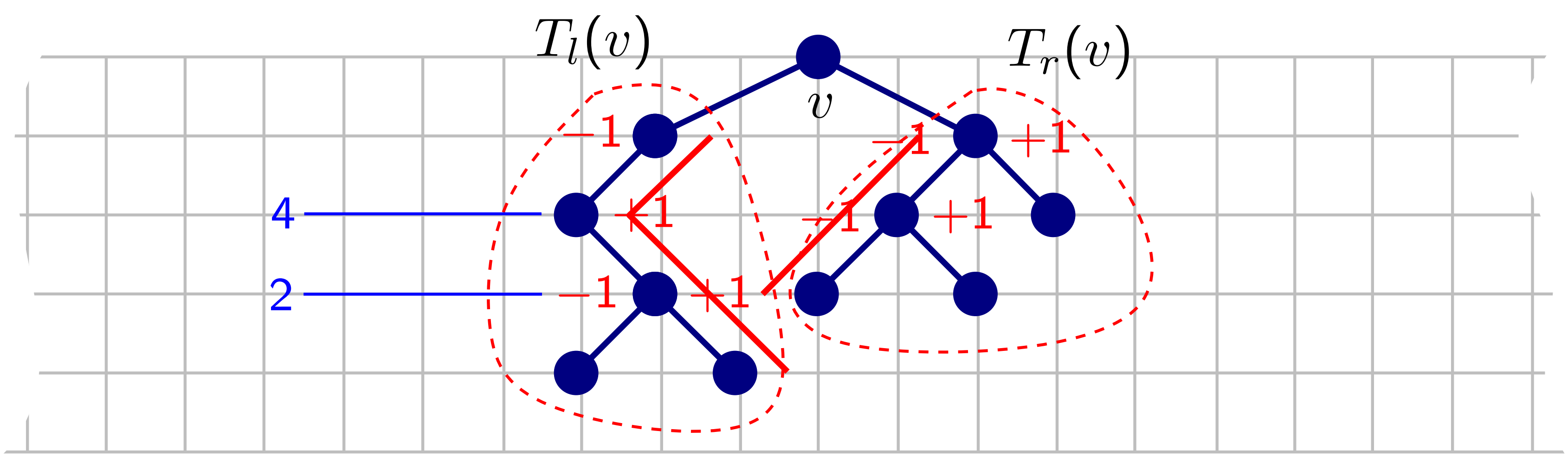


Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$

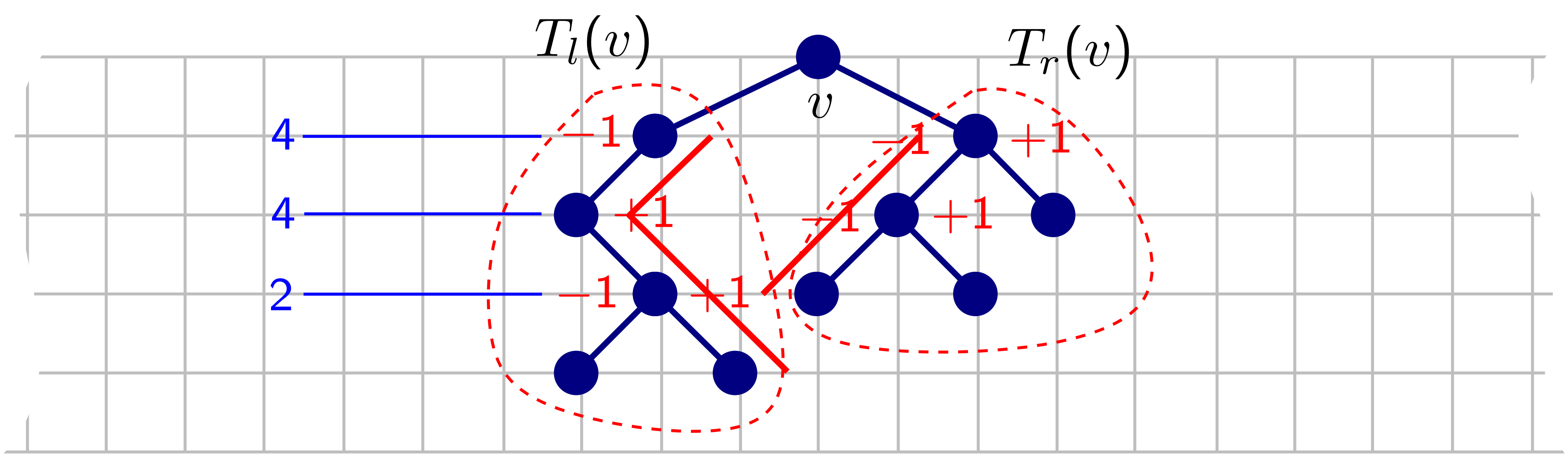


Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$

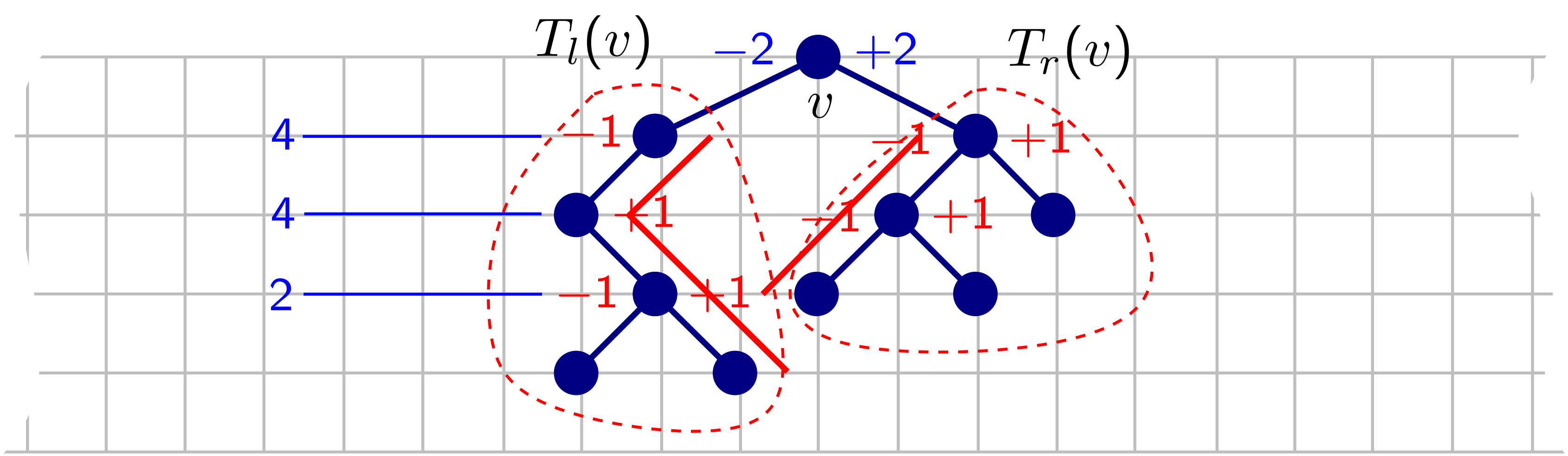


Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$

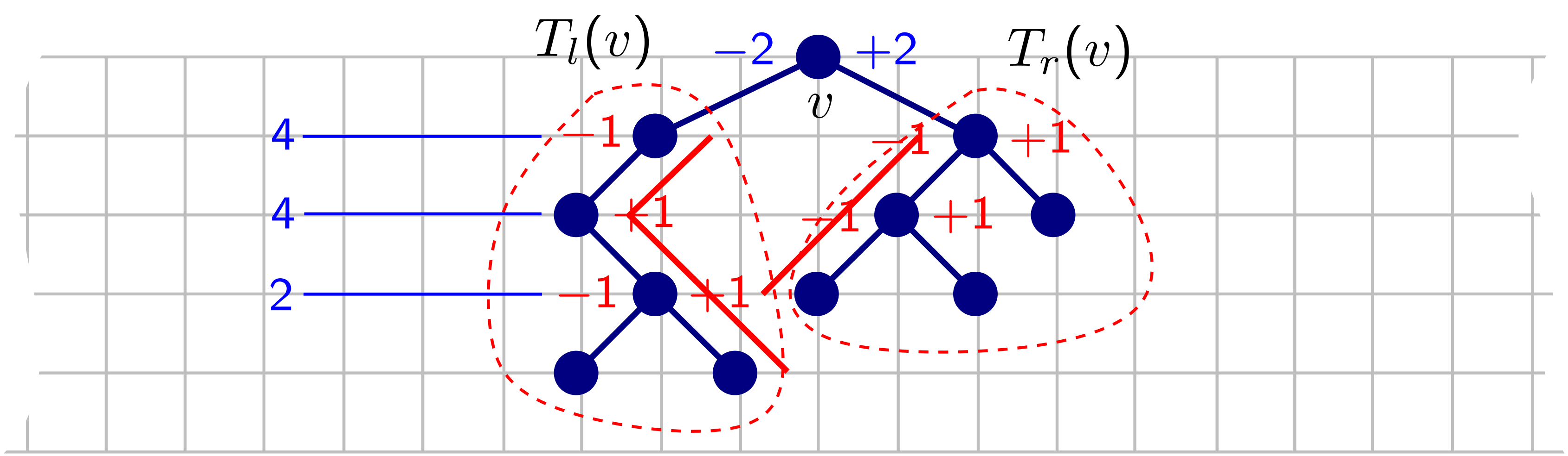


Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- "Summ up" the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of v

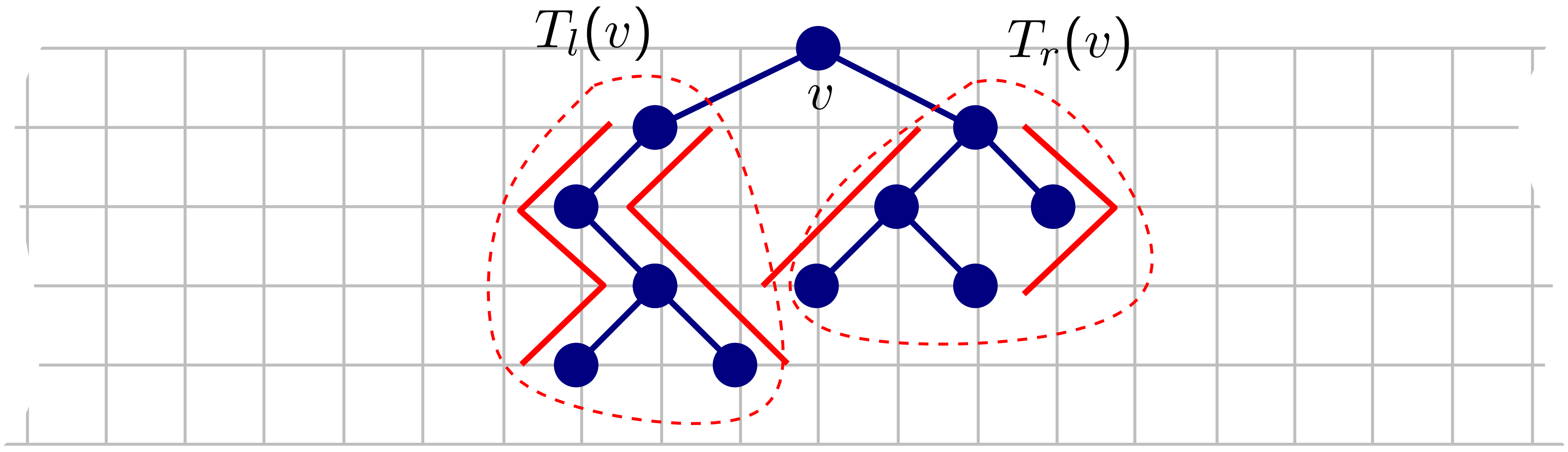


Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

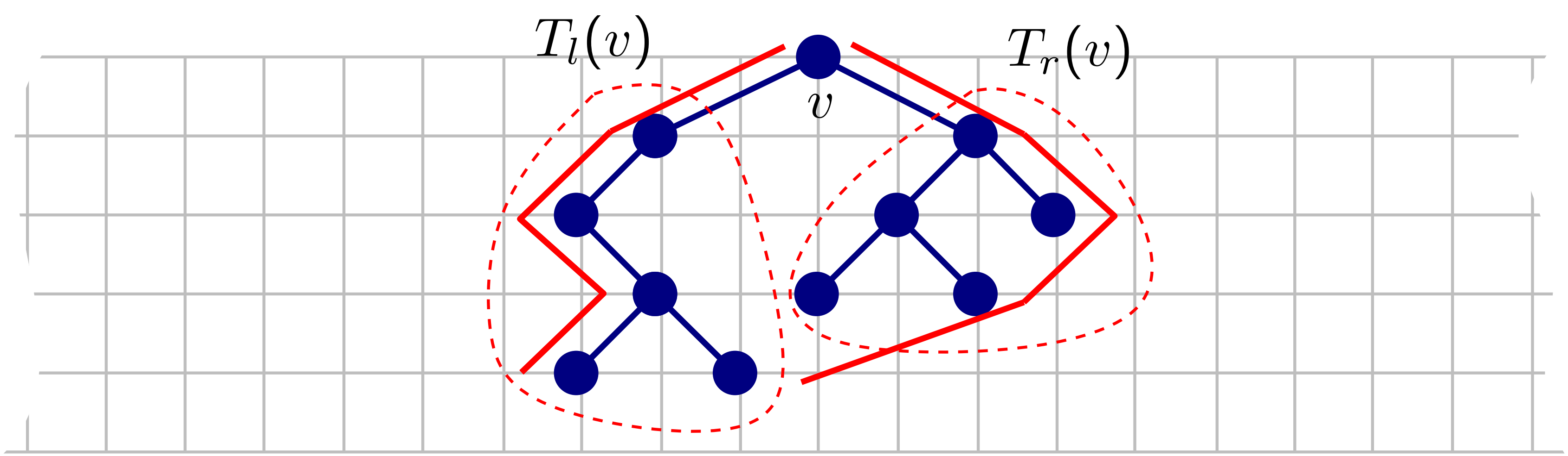
- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- "Summ up" the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of v



Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- “Summ up” the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of v

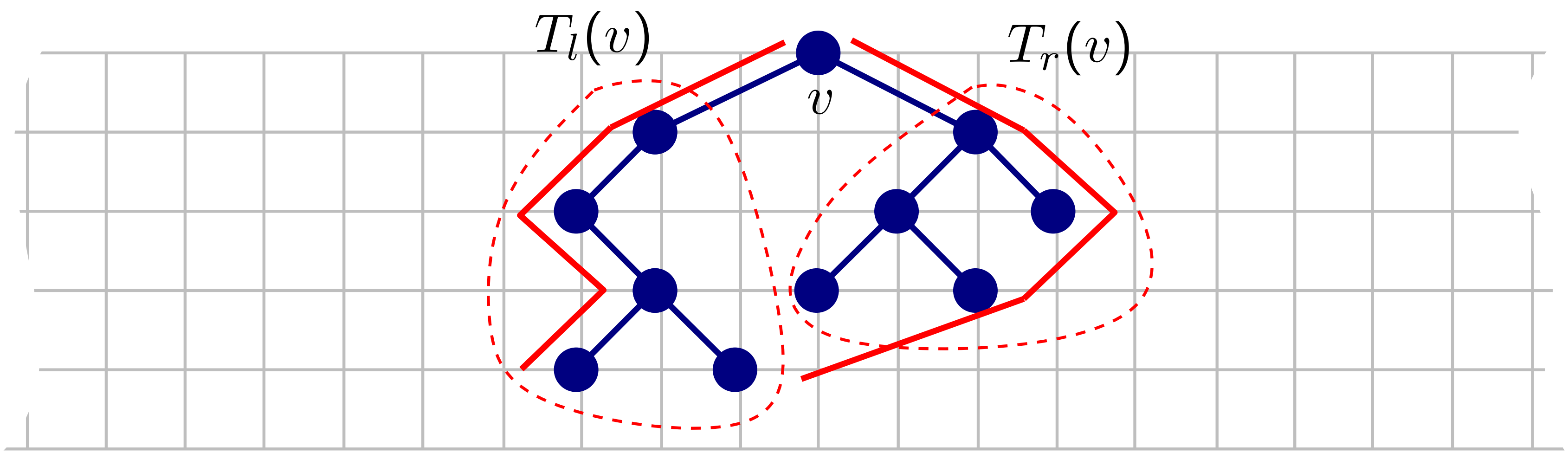


Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- “Summ up” the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$ to obtain the displ. of the children of v
- Store at v the left and the right boundaries of $T(v)$

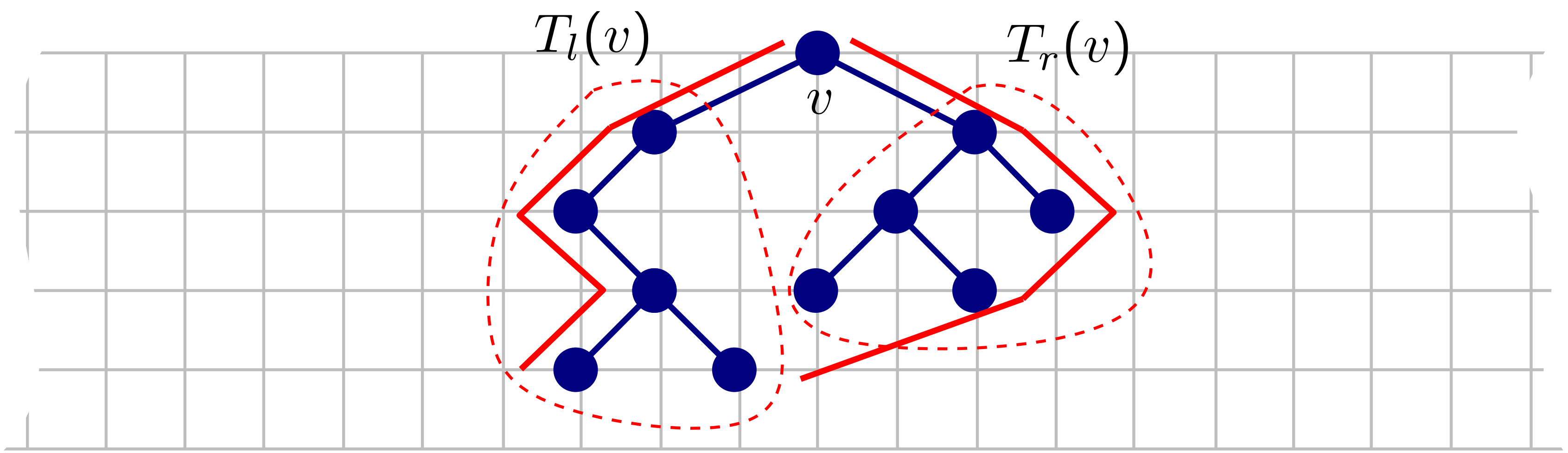


Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

Preorder traversal: Compute x- and y-coordinates.

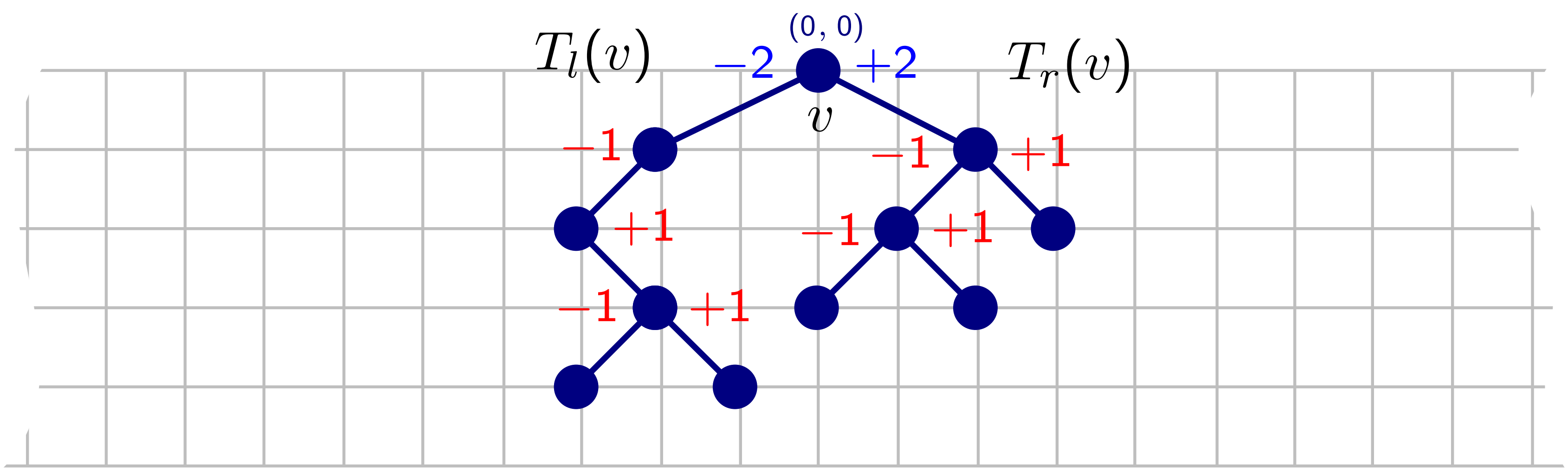


Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

Preorder traversal: Compute x- and y-coordinates.

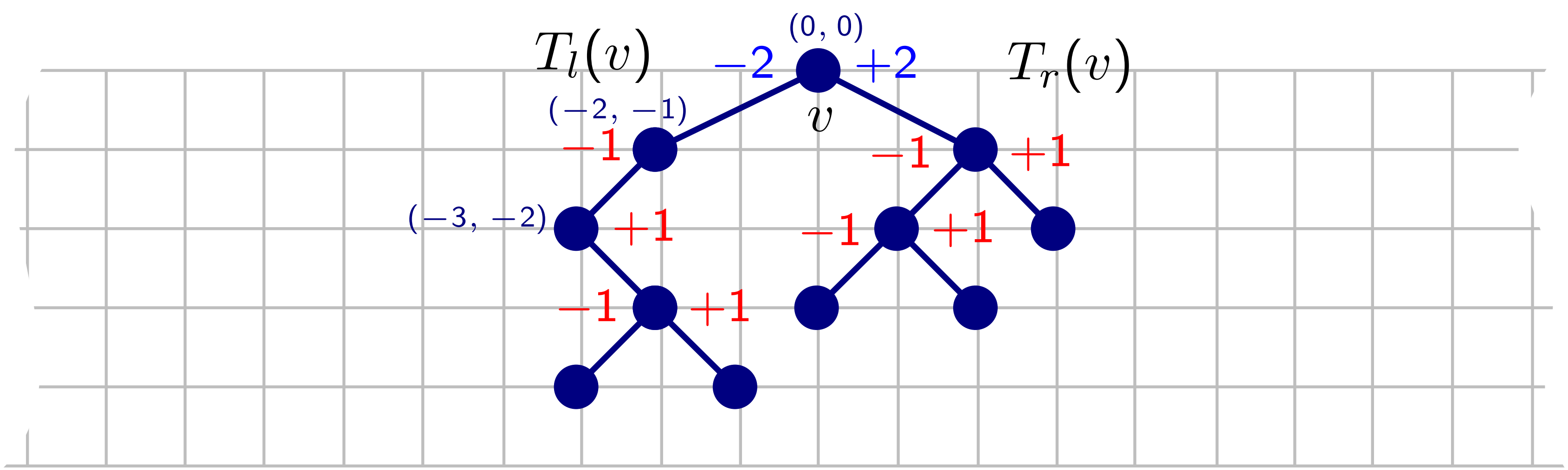


Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

Preorder traversal: Compute x- and y-coordinates.

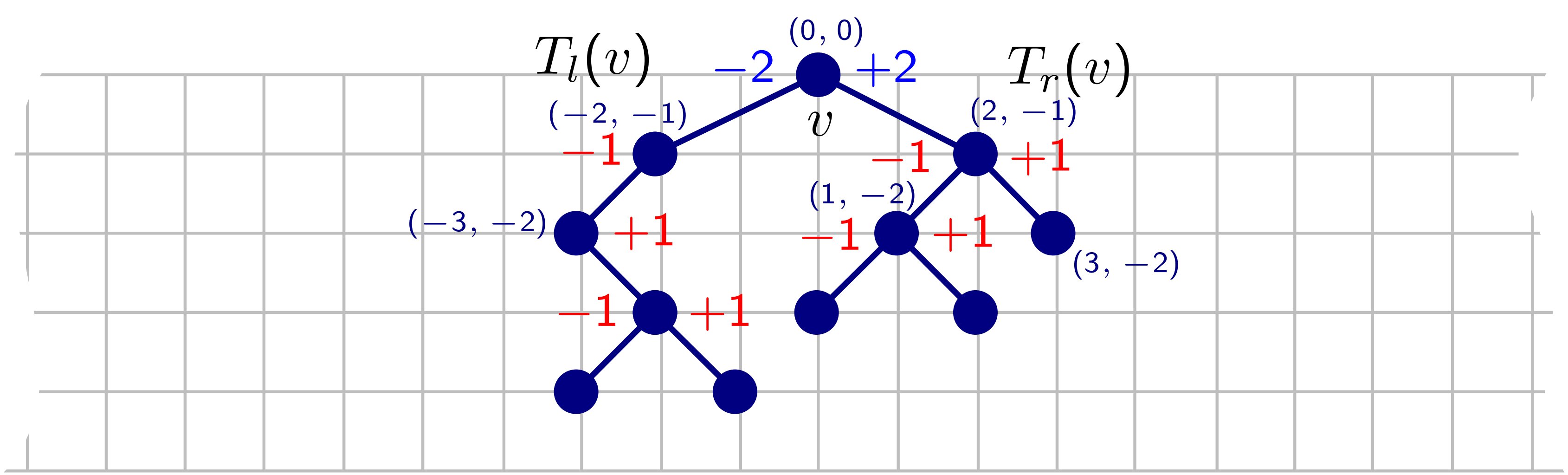


Reingold & Tilford

Implementation Details (postorder and preorder traversals)

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

Preorder traversal: Compute x- and y-coordinates.



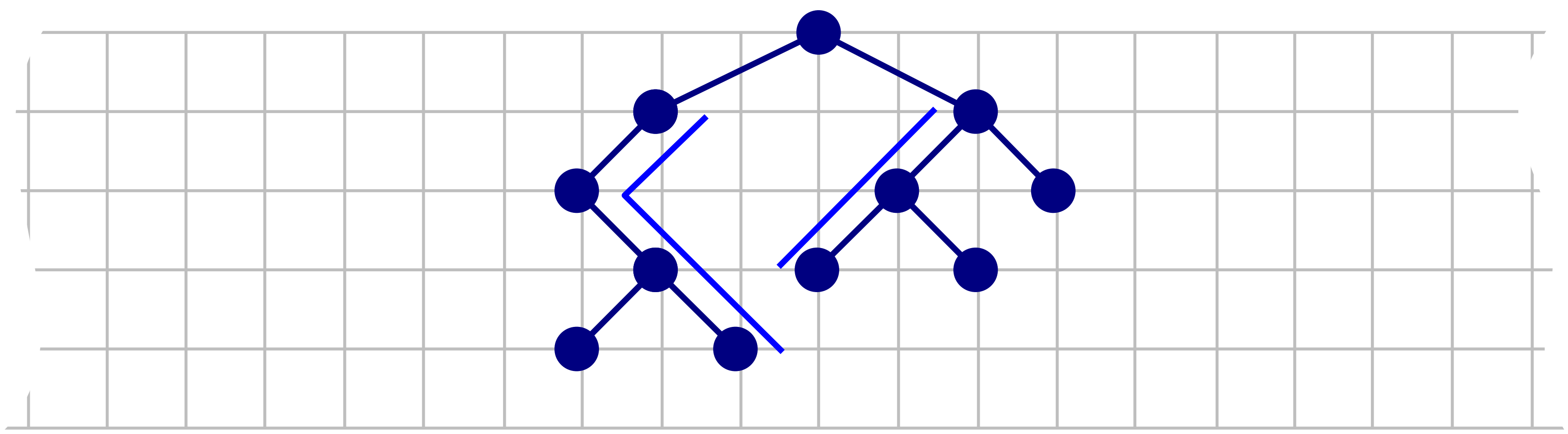
Reingold & Tilford

Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.



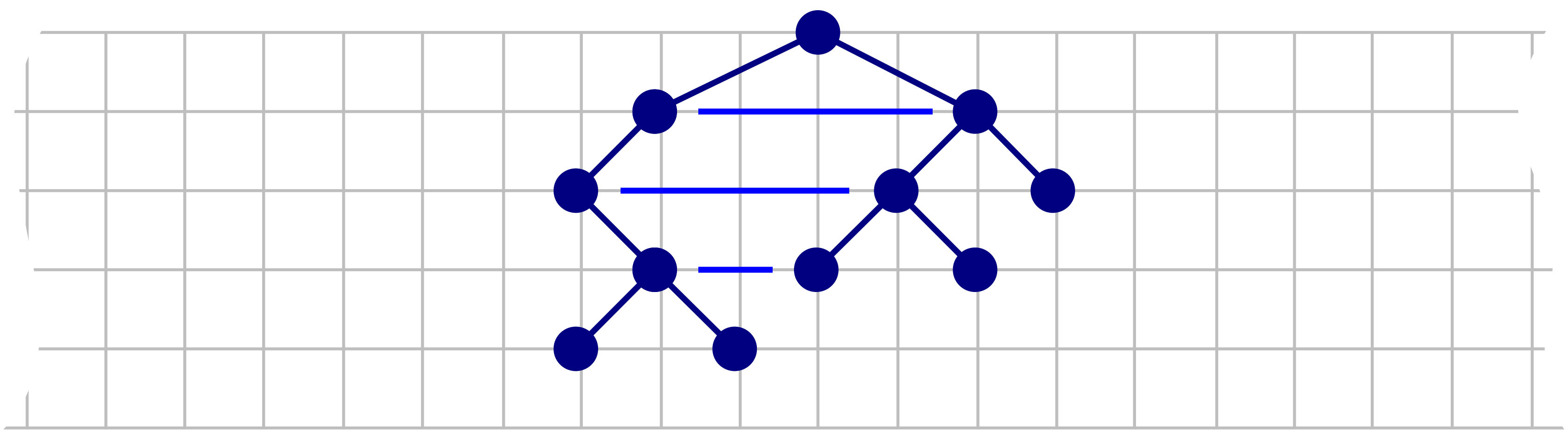
Reingold & Tilford

Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.



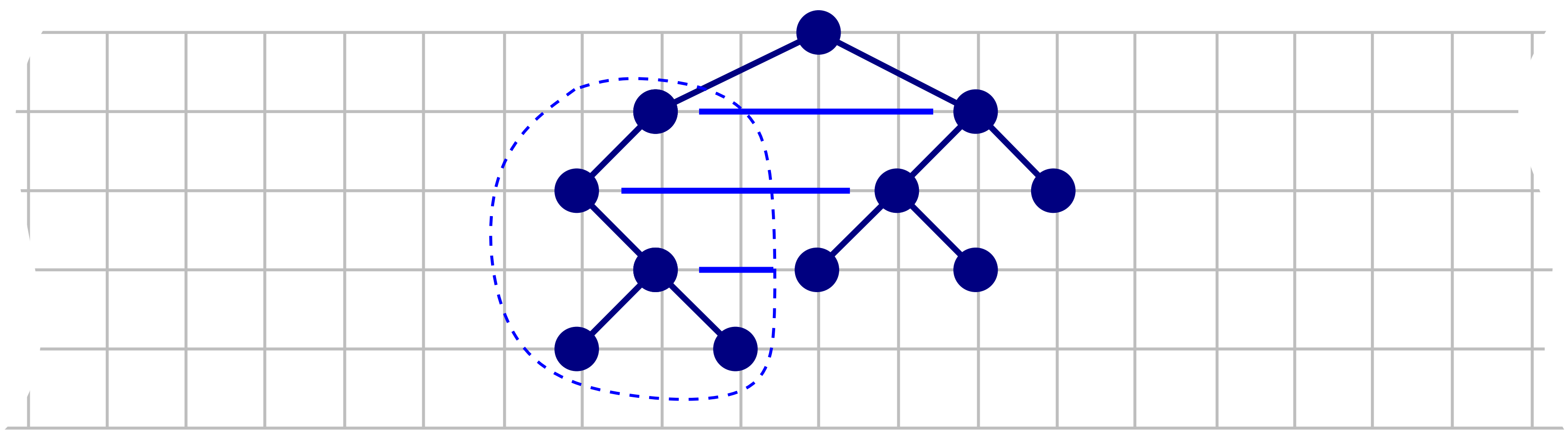
Reingold & Tilford

Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.



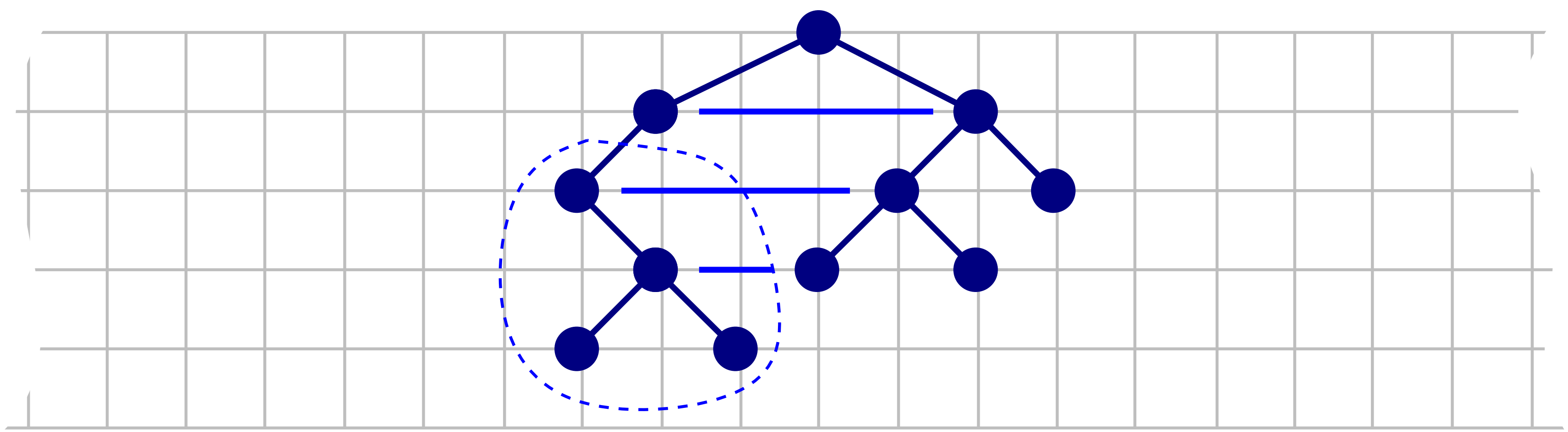
Reingold & Tilford

Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.



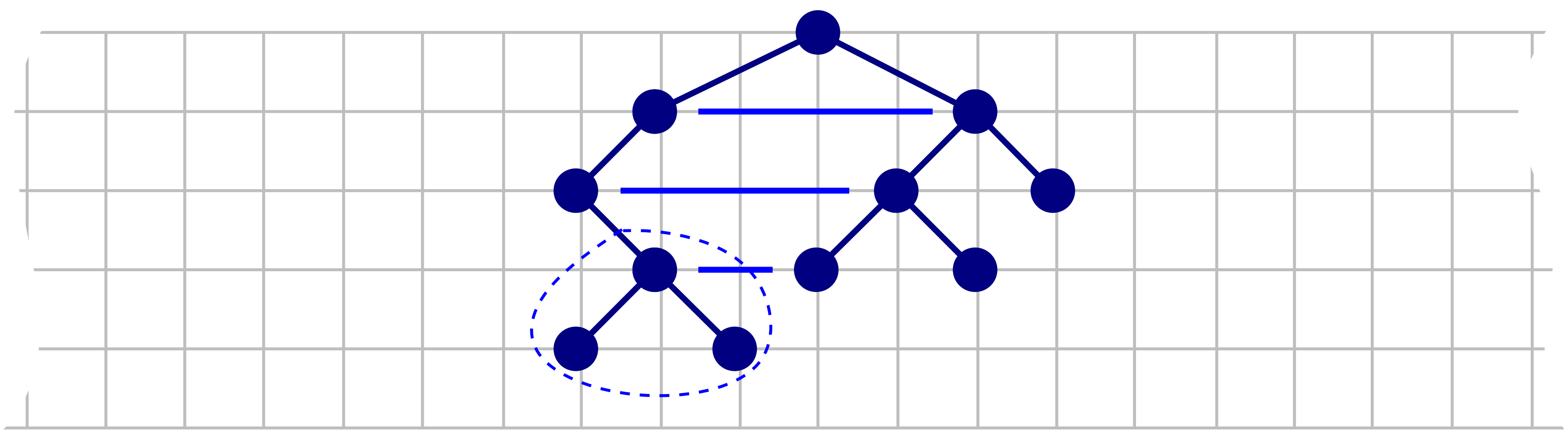
Reingold & Tilford

Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.



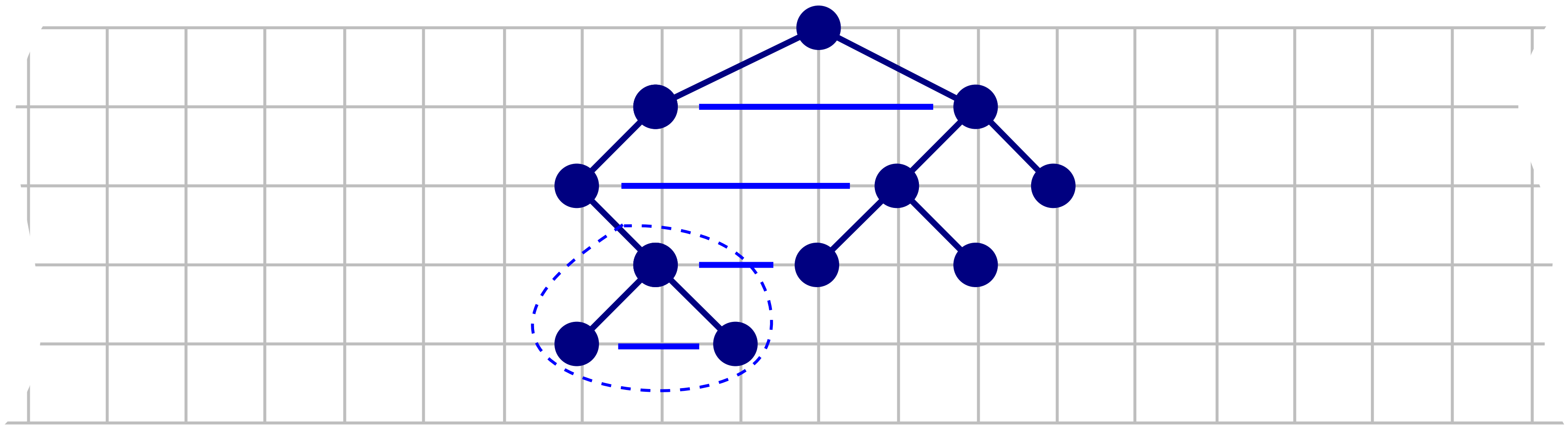
Reingold & Tilford

Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.



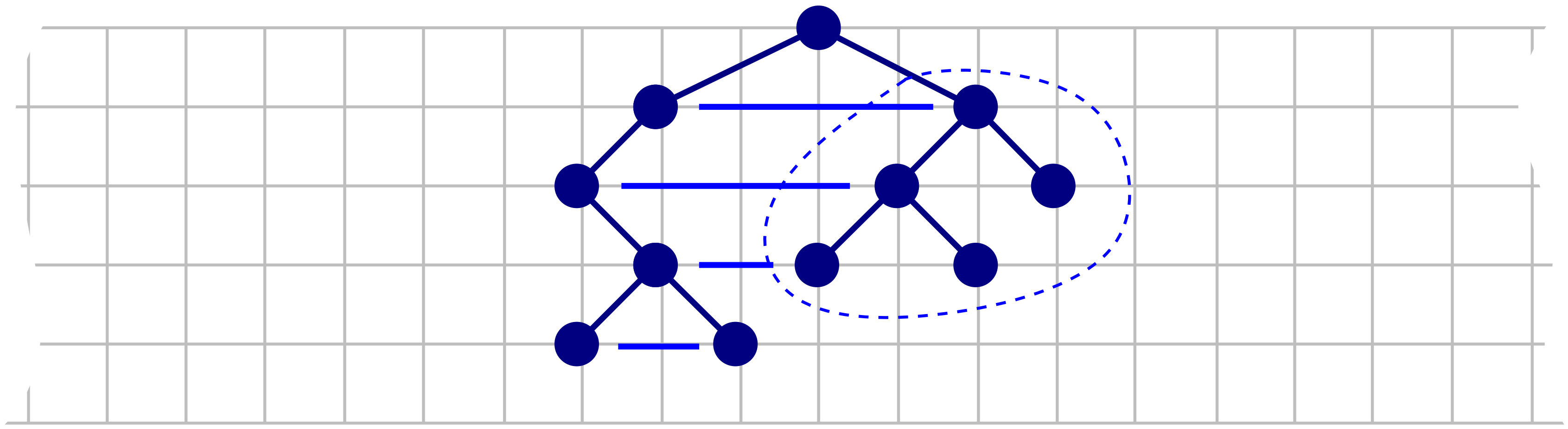
Reingold & Tilford

Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.



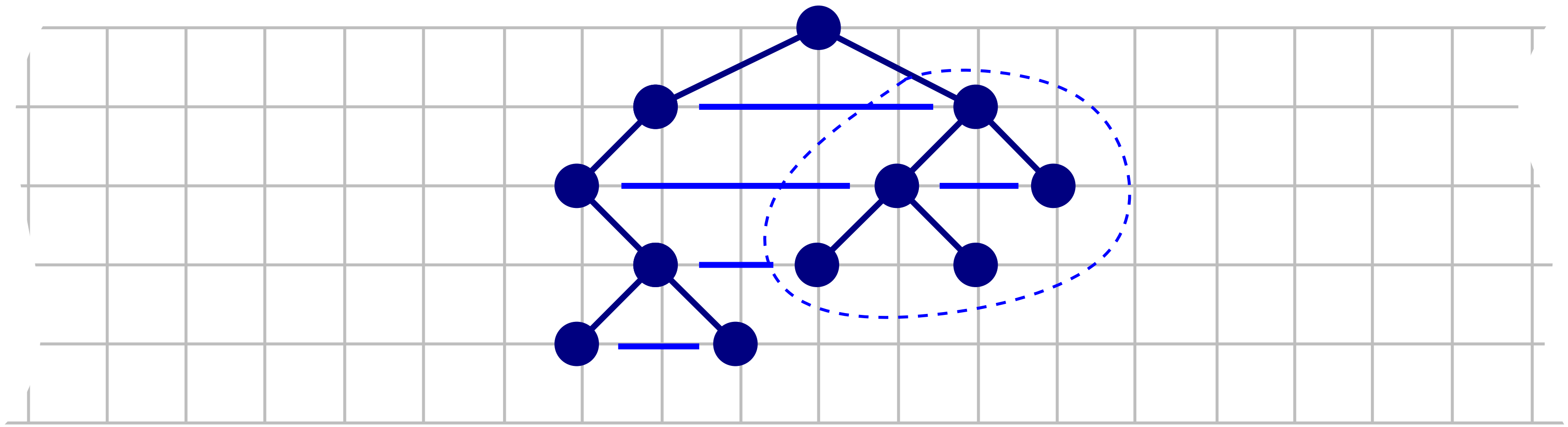
Reingold & Tilford

Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.



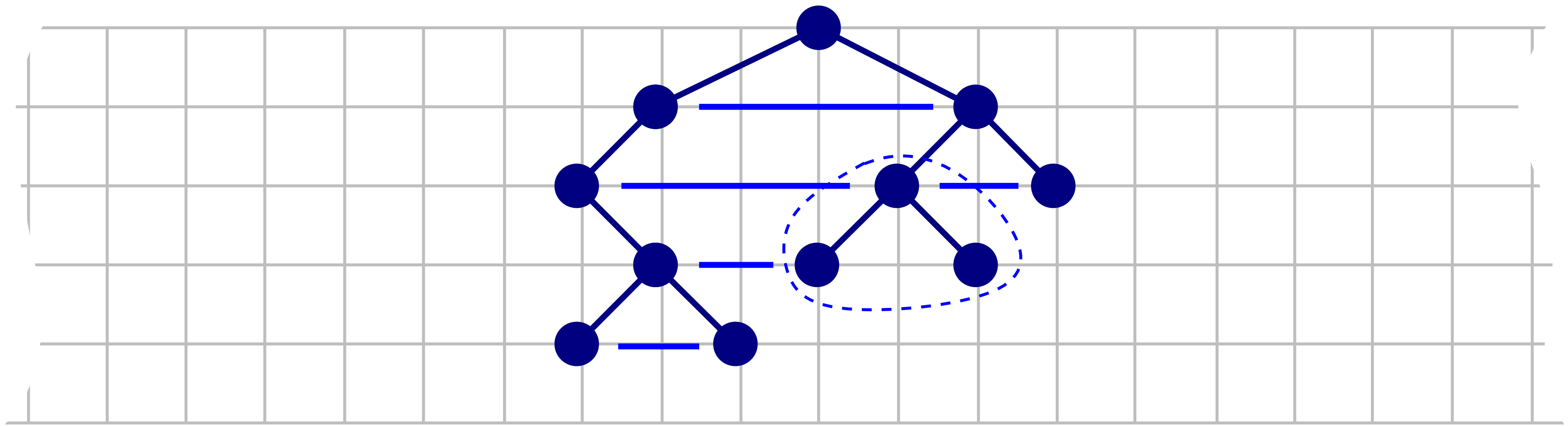
Reingold & Tilford

Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.



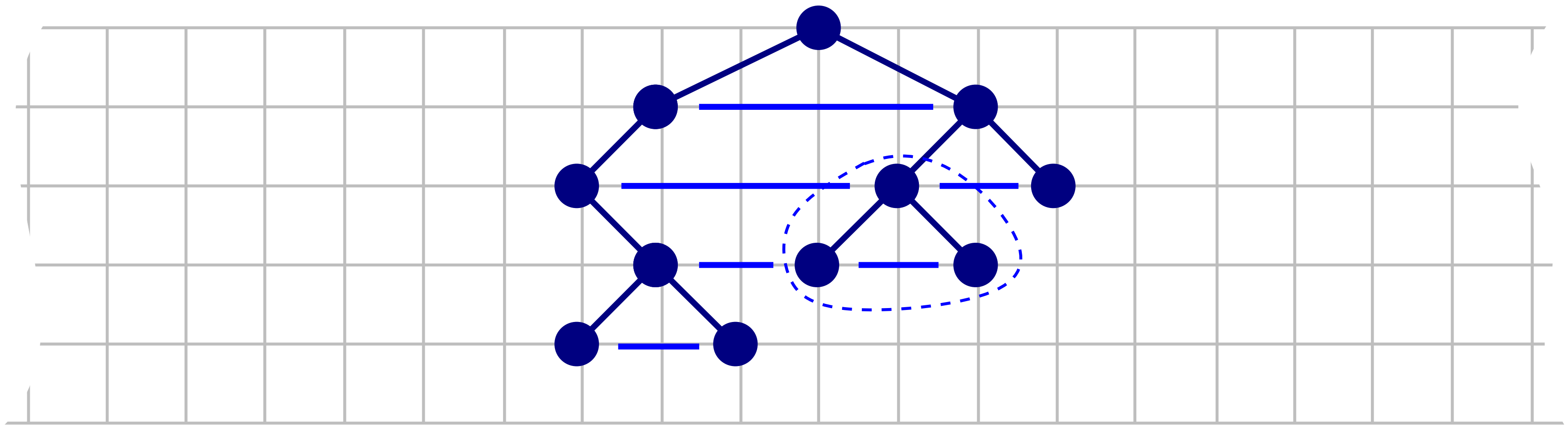
Reingold & Tilford

Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.



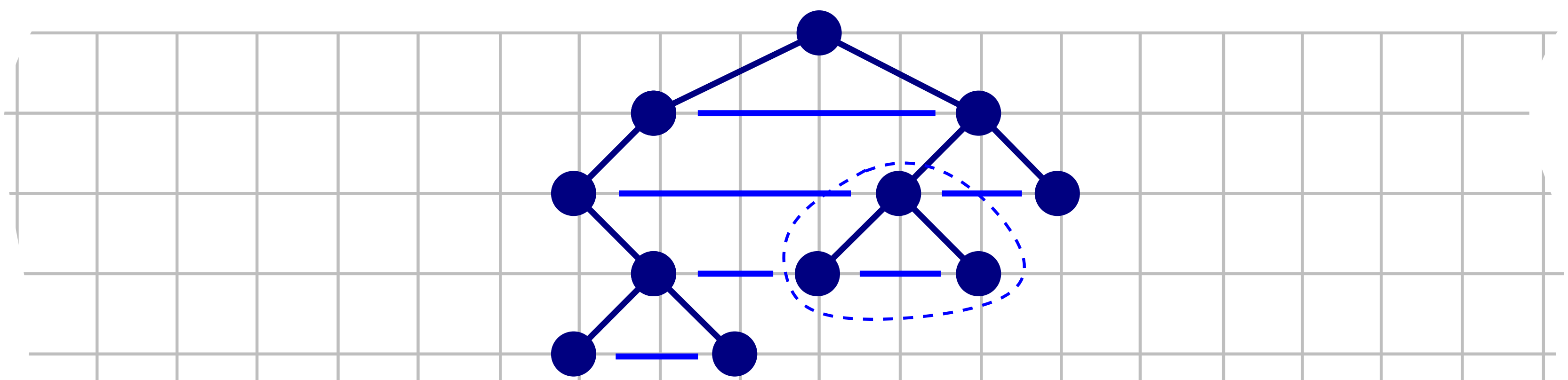
Reingold & Tilford

Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

Preorder traversal: Compute x- and y-coordinates.



- To compute the displacement: constant number of operations at each vertex

Reingold & Tilford

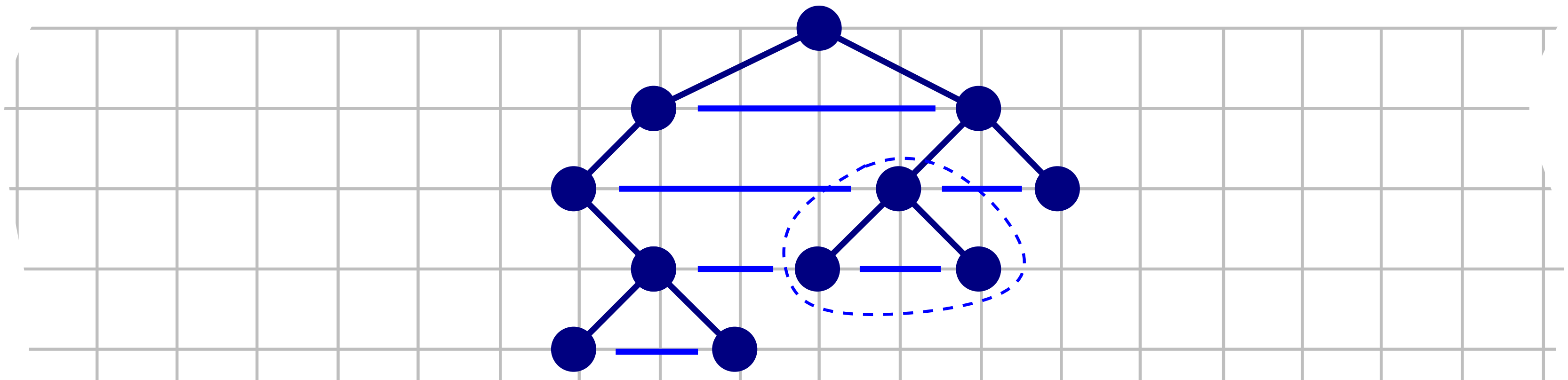
Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

$O(n)$

Preorder traversal: Compute x- and y-coordinates.



- To compute the displacement: constant number of operations at each vertex

Reingold & Tilford

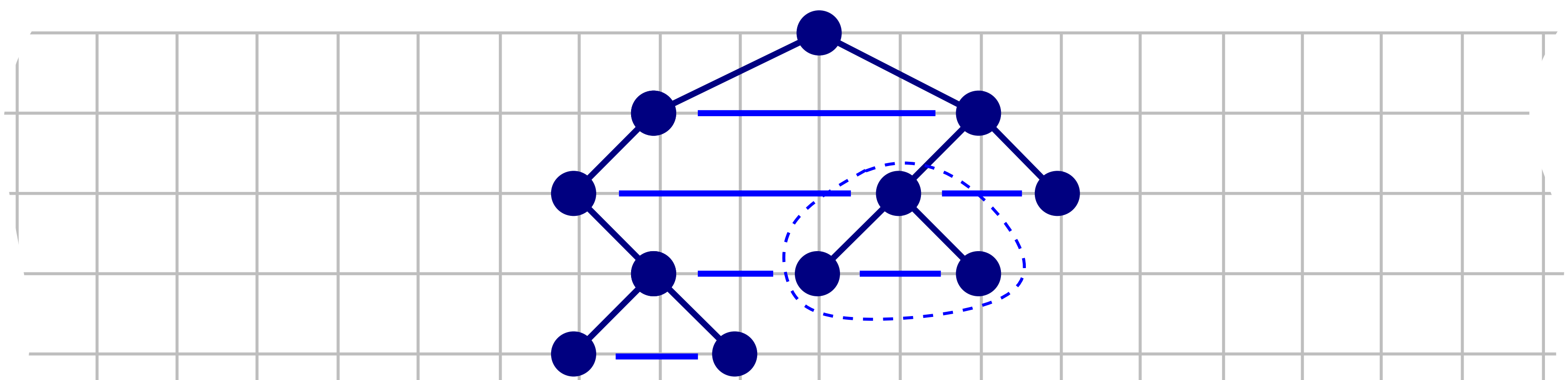
Time Complexity

Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

$O(n)$

Preorder traversal: Compute x- and y-coordinates.



- To compute the displacement: constant number of operations at each vertex

Reingold & Tilford

Time Complexity

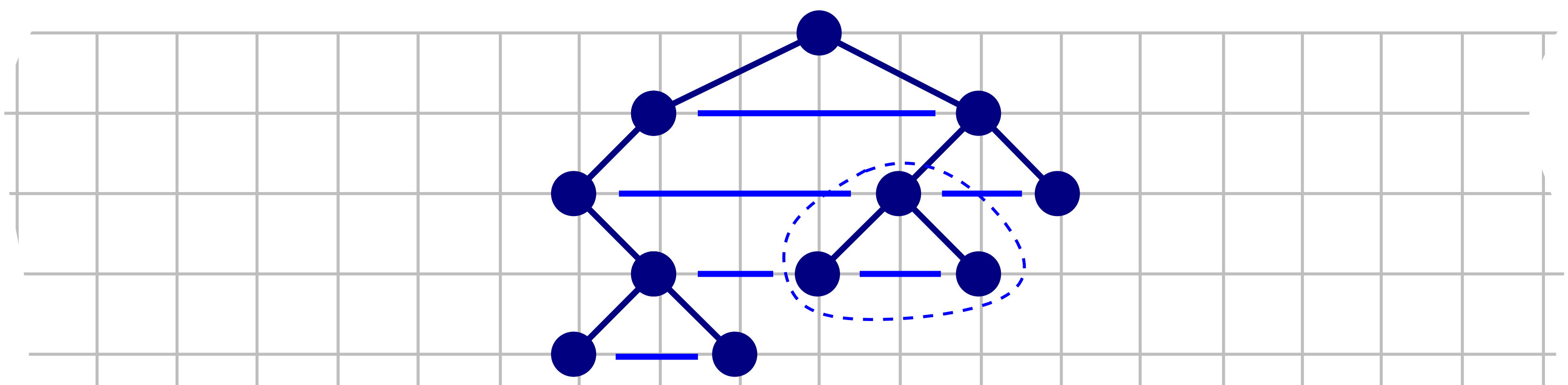
Postorder traversal: For each vertex v compute horizontal displacement of the left and the right child

- Assume at each vertex u (below v) we have stored the left and the right boundary of the subtree $T(u)$
- Summ up the horizontal displacements of the right boundary of $T_l(v)$ and the left boundary of $T_r(v)$
- Store at v the left and the right boundaries of $T(v)$

$O(n)$

Preorder traversal: Compute x- and y-coordinates.

$O(n)$



- To compute the displacement: constant number of operations at each vertex

Reingold & Tilford

Theorem (Reingold & Tilford)

Let T be a binary tree with n vertices. Algorithm (R & T) constructs a drawing Γ of T in $O(n)$ time, such that:

- Γ is planar and straight-line
- $\forall v \in T$ y-coordinate of v is $-depth(v)$
- Vertical and horizontal distance is at least 1
- Area of Γ is

Reingold & Tilford

Theorem (Reingold & Tilford)

Let T be a binary tree with n vertices. Algorithm (R & T) constructs a drawing Γ of T in $O(n)$ time, such that:

- Γ is planar and straight-line
- $\forall v \in T$ y-coordinate of v is $-depth(v)$
- Vertical and horizontal distance is at least 1
- Area of Γ is $O(n^2)$

Reingold & Tilford

Theorem (Reingold & Tilford)

Let T be a binary tree with n vertices. Algorithm (R & T) constructs a drawing Γ of T in $O(n)$ time, such that:

- Γ is planar and straight-line
- $\forall v \in T$ y-coordinate of v is $-depth(v)$
- Vertical and horizontal distance is at least 1
- Area of Γ is $O(n^2)$
- Each vertex is centered with respect to its children

Reingold & Tilford

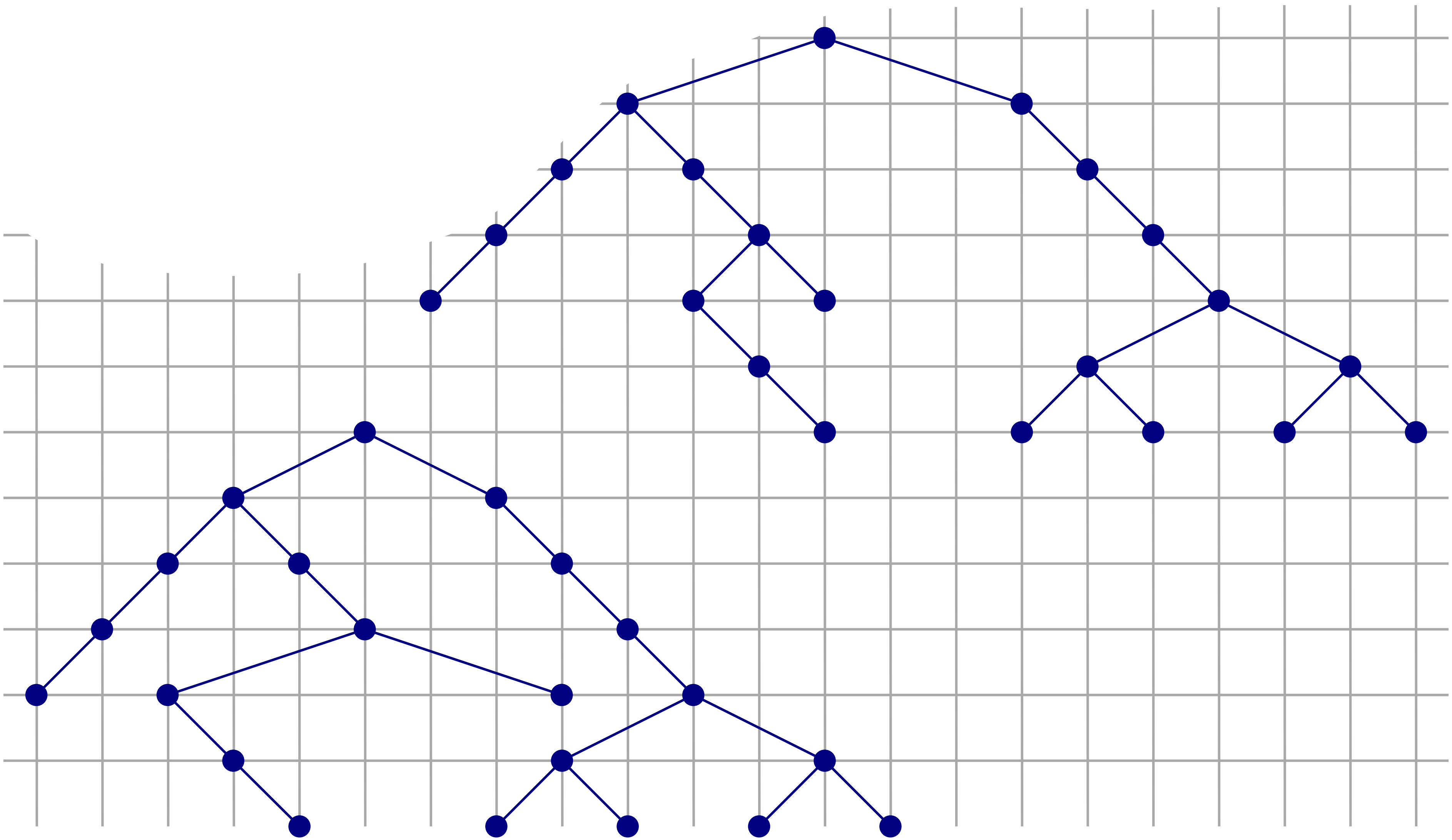
Theorem (Reingold & Tilford)

Let T be a binary tree with n vertices. Algorithm (R & T) constructs a drawing Γ of T in $O(n)$ time, such that:

- Γ is planar and straight-line
- $\forall v \in T$ y-coordinate of v is $-depth(v)$
- Vertical and horizontal distance is at least 1
- Area of Γ is $O(n^2)$
- Each vertex is centered with respect to its children
- Simply isomorphic subtrees have congruent (coincident) drawing, up to translation
- Axially isomorphic trees have congruent drawing, up to translation and reflection around y-axis

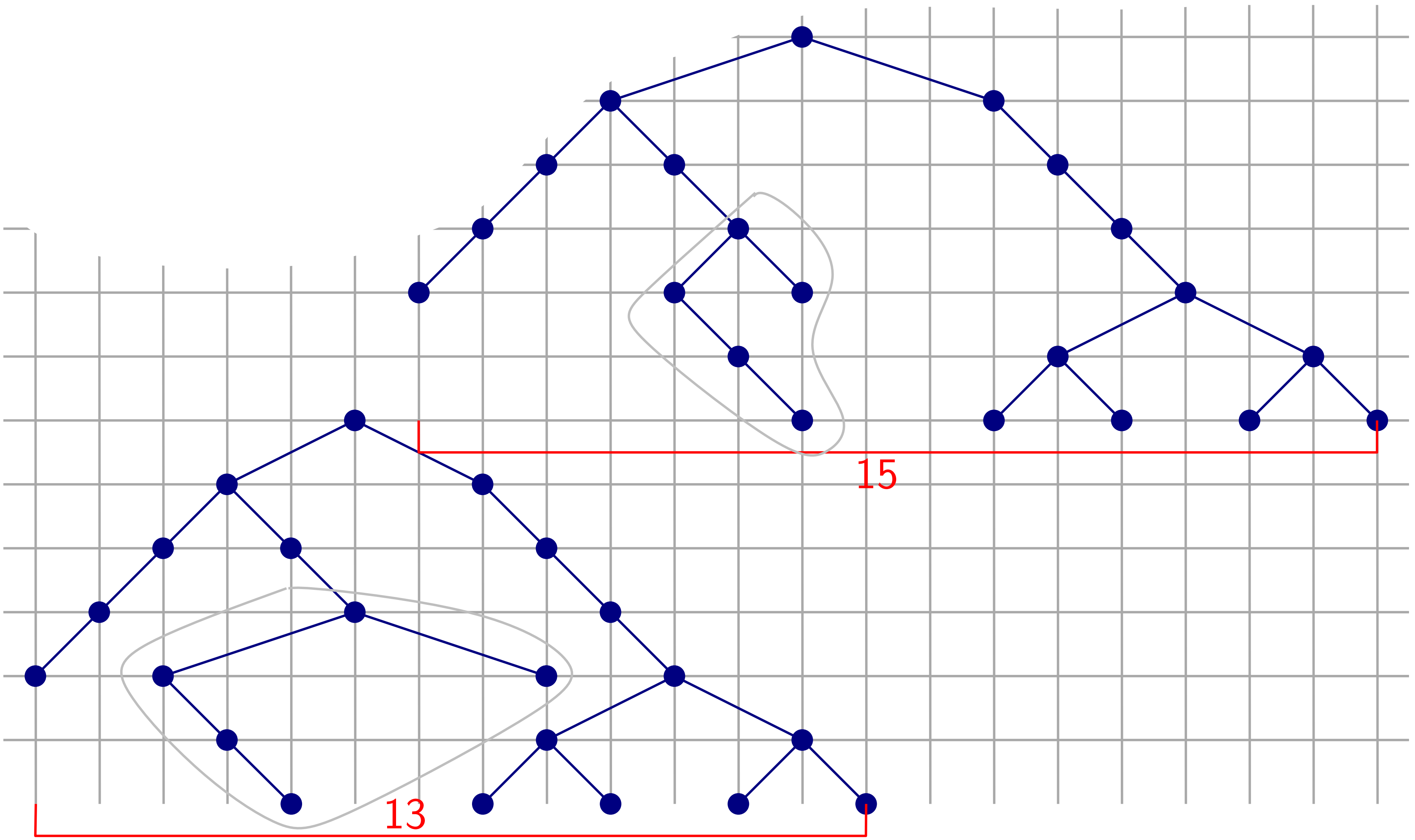
Reingold & Tilford

- The algorithm of Reingold & Tilford tries to minimize width



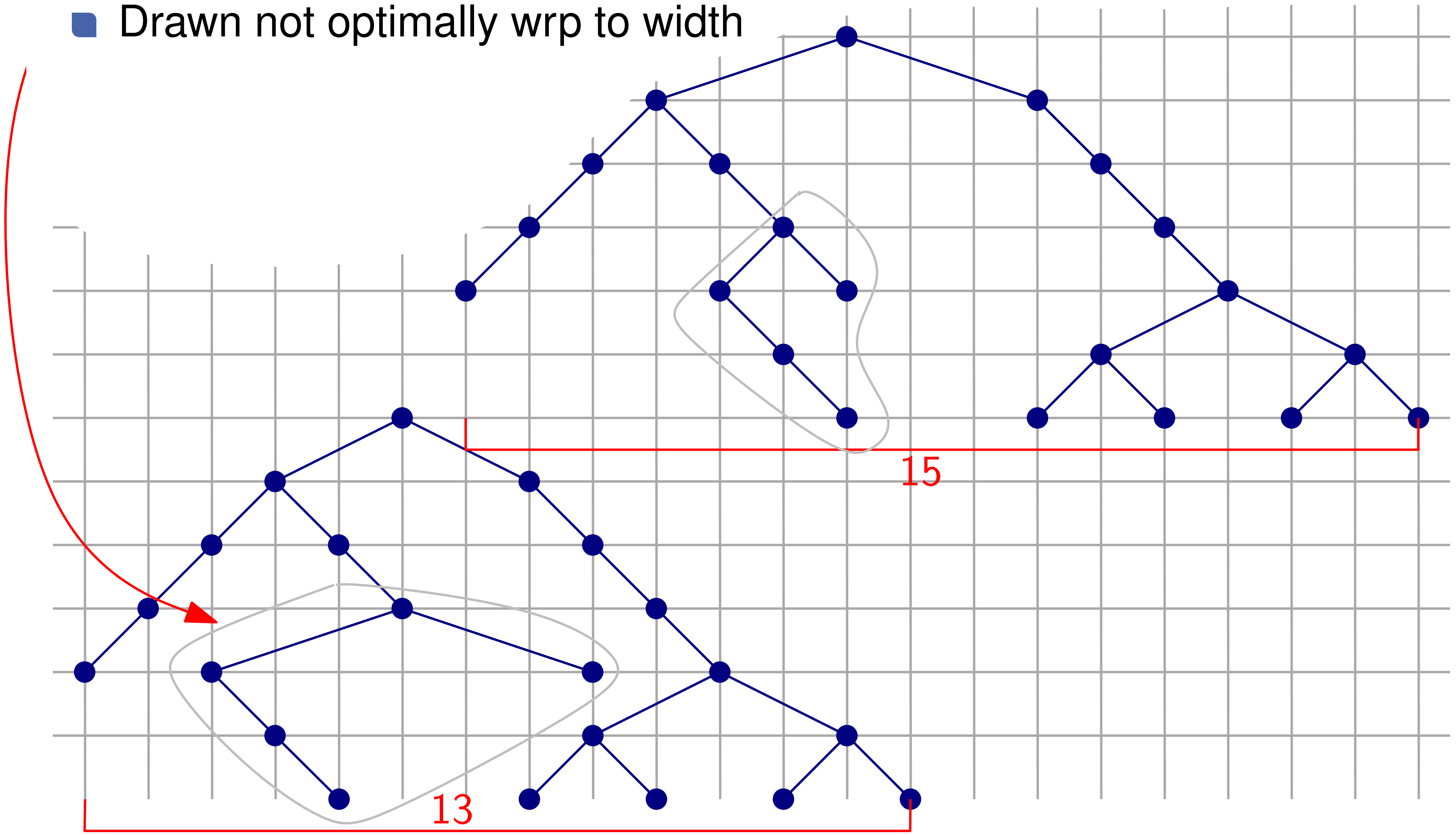
Reingold & Tilford

- The algorithm of Reingold & Tilford tries to minimize width



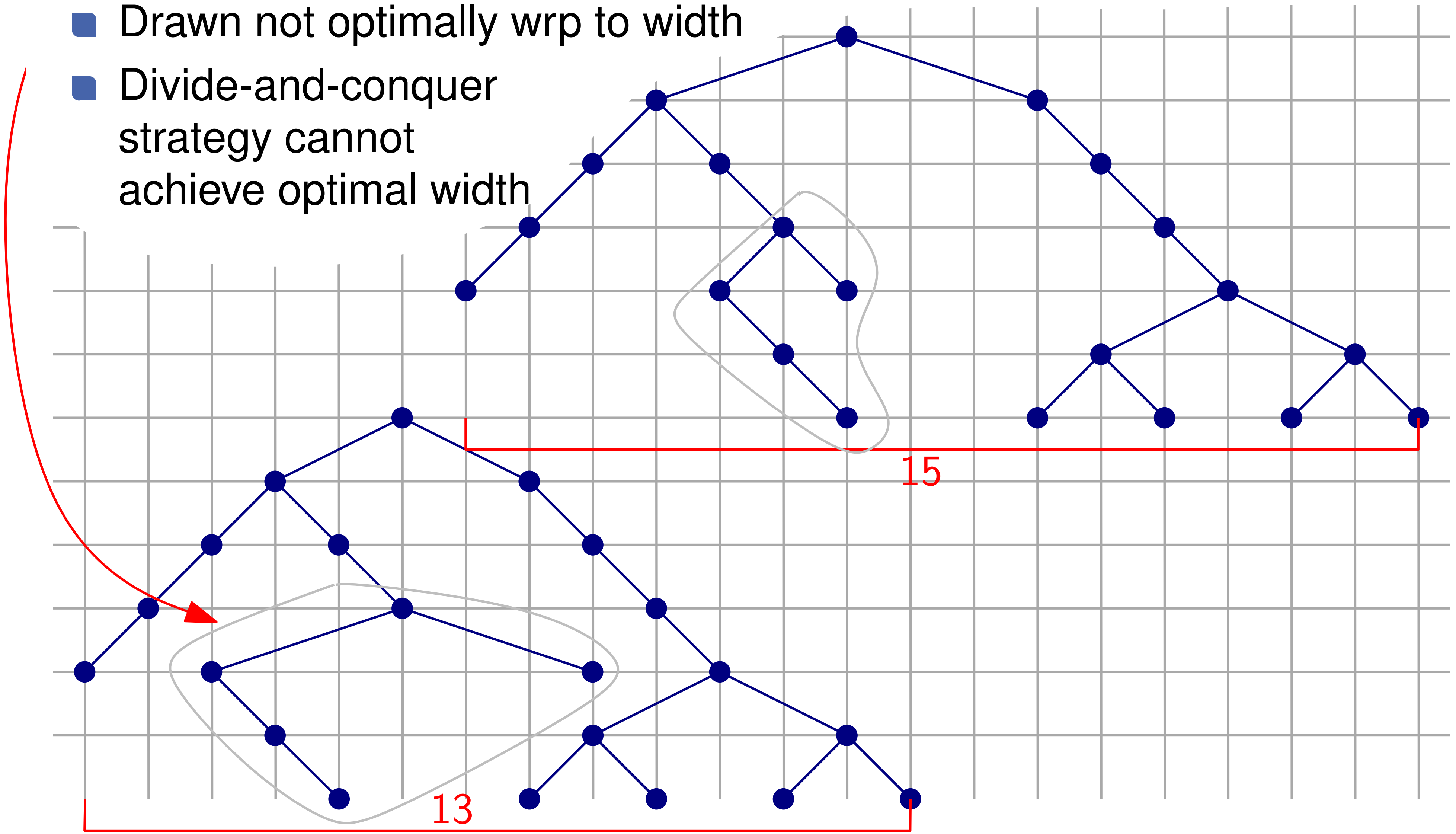
Reingold & Tilford

- The algorithm of Reingold & Tilford tries to minimize width
- Drawn not optimally wrp to width



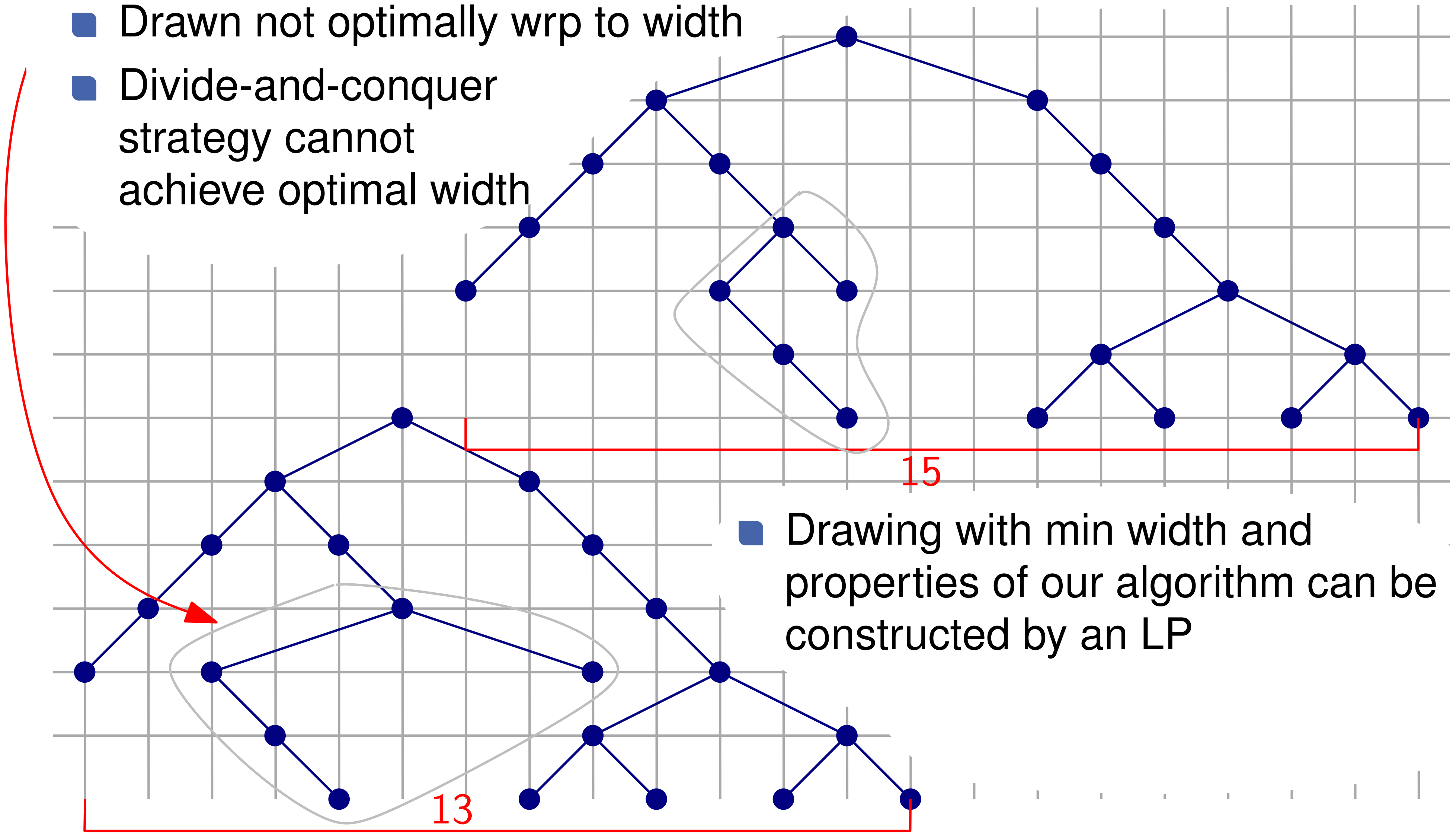
Reingold & Tilford

- The algorithm of Reingold & Tilford tries to minimize width
- Drawn not optimally wrp to width
- Divide-and-conquer strategy cannot achieve optimal width



Reingold & Tilford

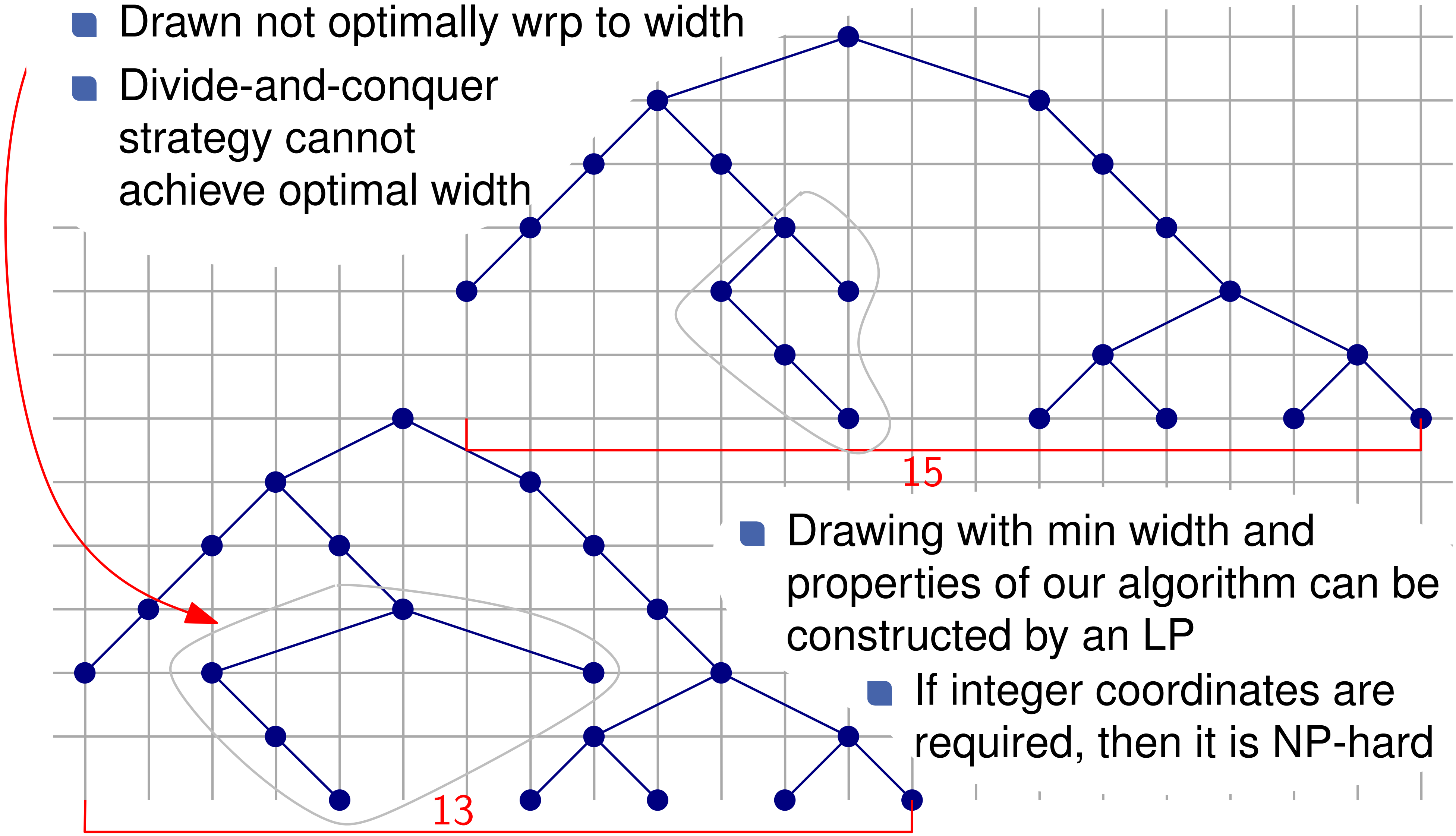
- The algorithm of Reingold & Tilford tries to minimize width
- Drawn not optimally wrp to width
- Divide-and-conquer strategy cannot achieve optimal width



- Drawing with min width and properties of our algorithm can be constructed by an LP

Reingold & Tilford

- The algorithm of Reingold & Tilford tries to minimize width
- Drawn not optimally wrp to width
- Divide-and-conquer strategy cannot achieve optimal width



- Drawing with min width and properties of our algorithm can be constructed by an LP
  - If integer coordinates are required, then it is NP-hard

Reingold & Tilford. General trees.

Algorithm Outline:

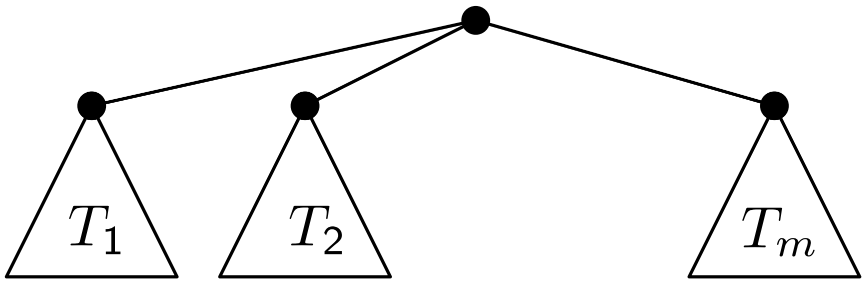
Input: A rooted tree

Output: A layered drawing of T

Base case: A single vertex

Divide: Assume that T has subtrees $T_1, \ldots T_m$. Draw each T_i recursively.

Conquer:



Reingold & Tilford. General trees.

Algorithm Outline:

Input: A rooted tree

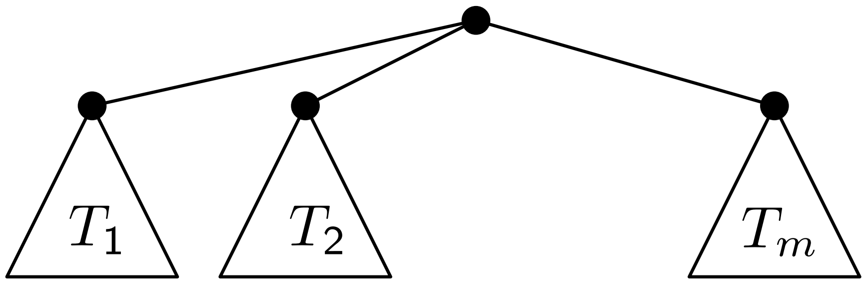
Output: A layered drawing of T

Base case: A single vertex

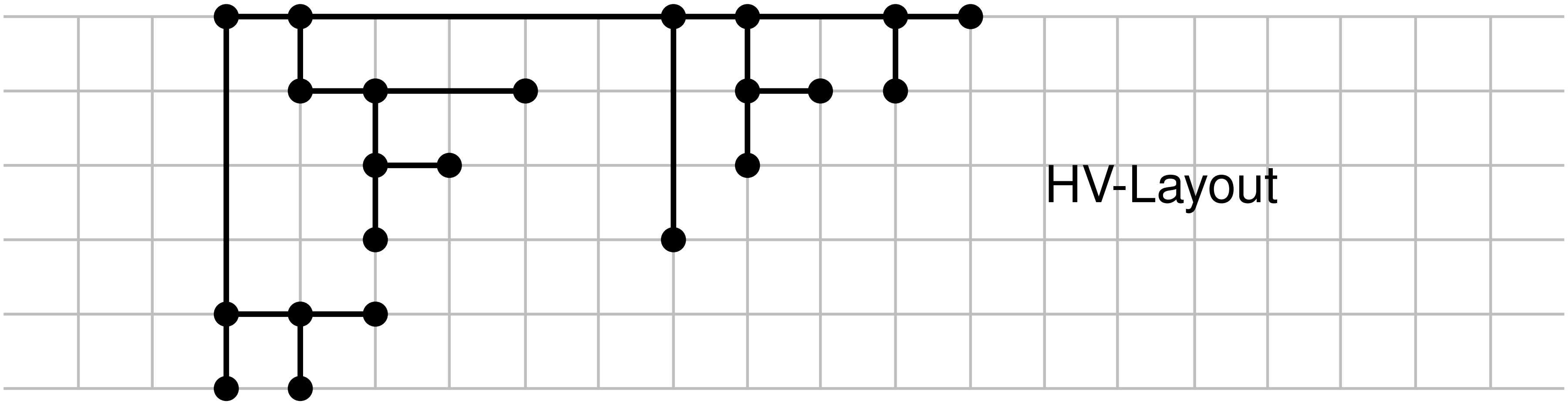
Divide: Assume that T has subtrees $T_1, \ldots T_m$. Draw each T_i recursively.

Conquer:

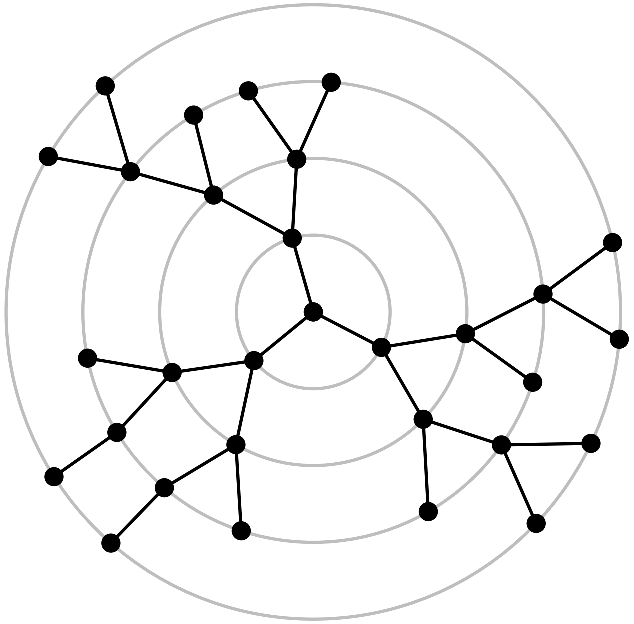
- For $i = 1, \ldots, m$ place the drawing of T_i to the right of the drawing of T_{i-1} and at horizontal distance at least 2 from it.
- Position the root half-way between the roots of T_1 and T_m.



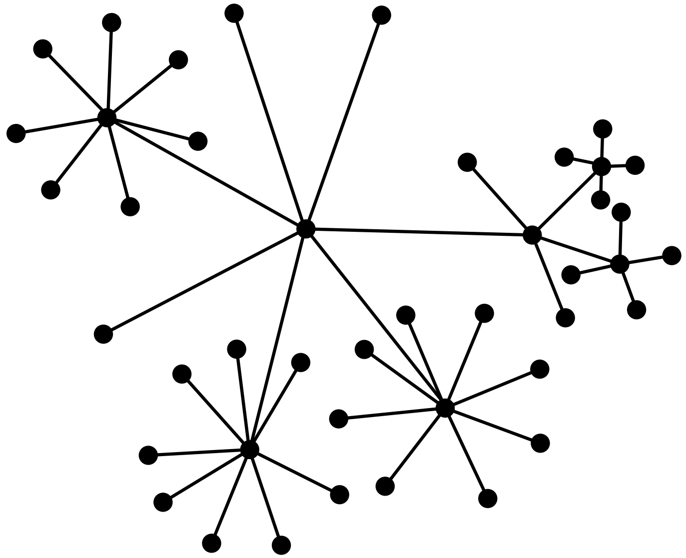
What’s next?



HV-Layout



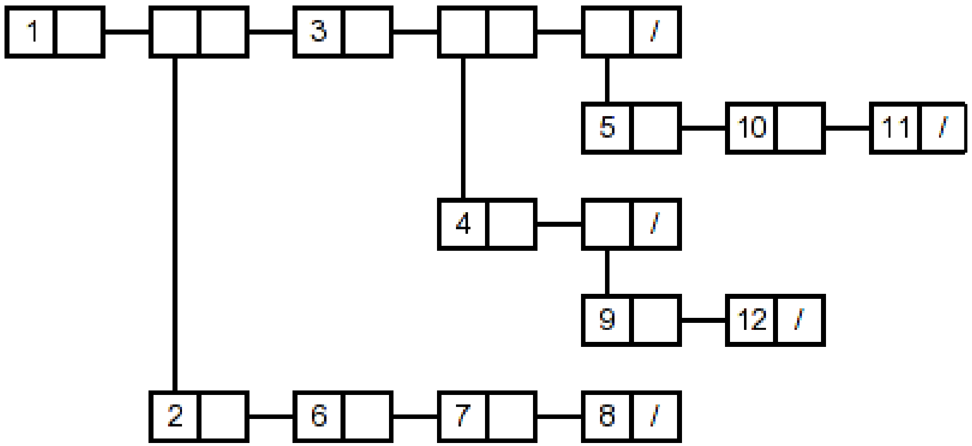
Radial Layout



Baloon Layout

HV-Layout

Cons cell diagram in LISP



`http://gajon.org/`

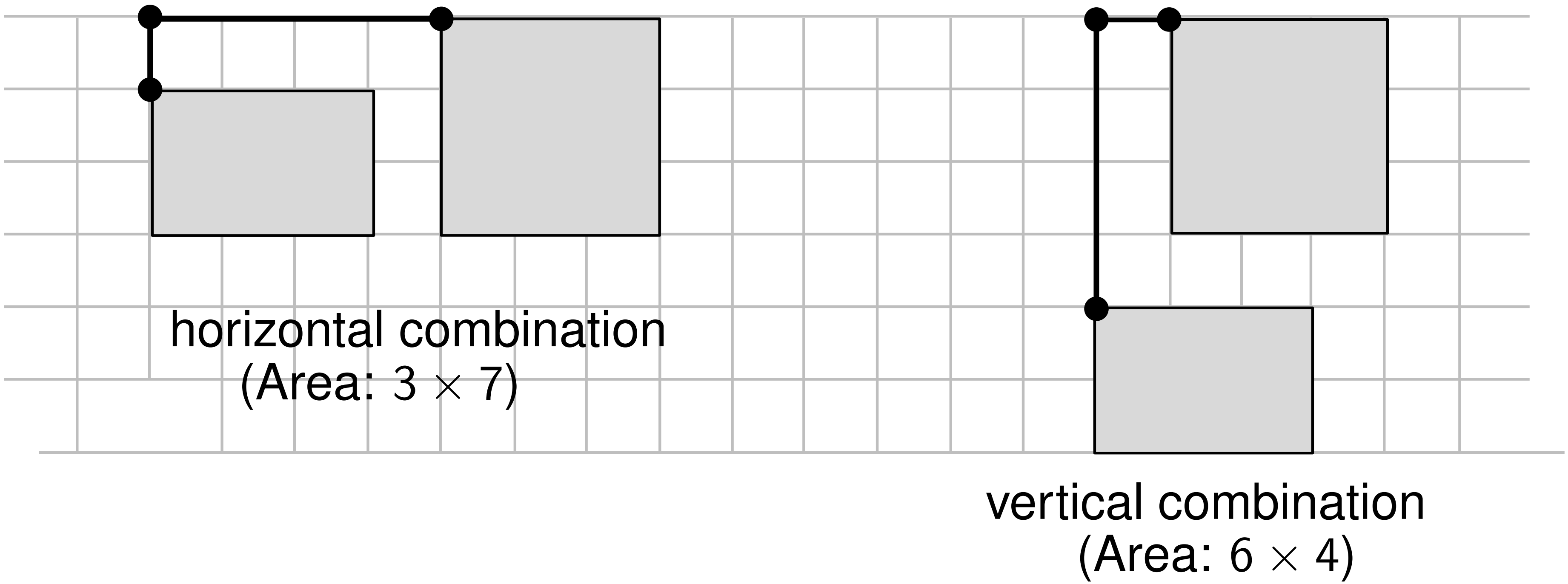
HV-Layout

Idea for binary trees:

- Children are vertically and horizontally aligned with the root
- The bounding boxes of the children do not intersect

Induction base: 

Induction step: combine layouts



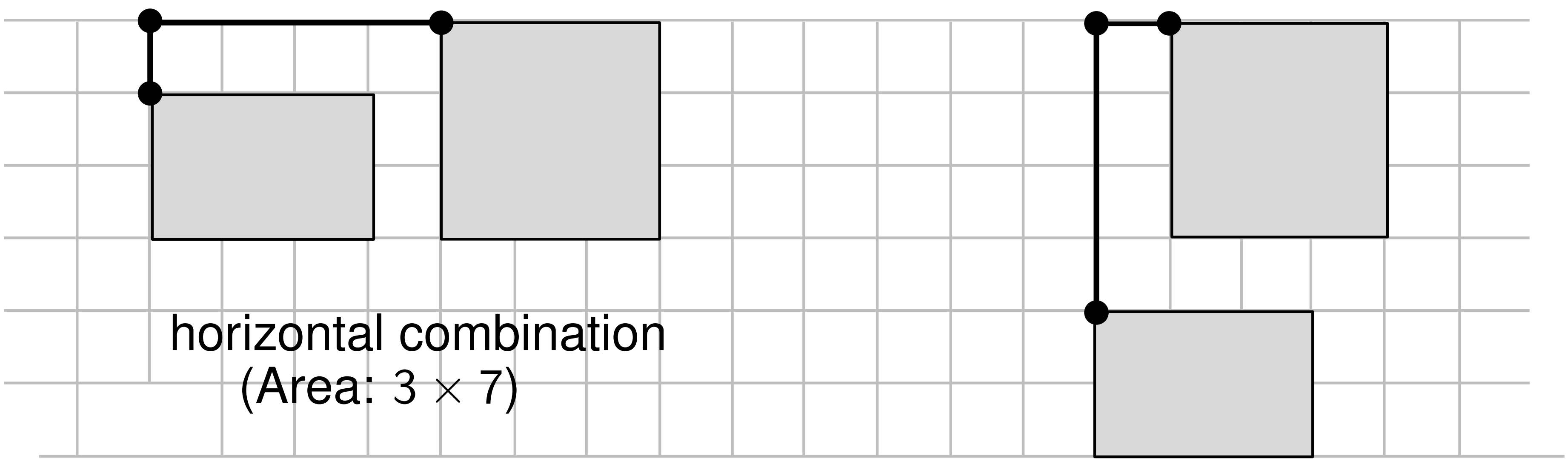
HV-Layout

Idea for binary trees:

- Children are vertically and horizontally aligned with the root
- The bounding boxes of the children do not intersect

Induction base:

Induction step: combine layouts



Compute minimum area using Dynamic Programming

Right-Heavy HV-Layout

Right-Heavy approach:

- At every induction step apply horizontal combination
- Place the larger subtree to the right

Right-Heavy HV-Layout

Right-Heavy approach:

- At every induction step apply horizontal combination
- Place the larger subtree to the right

Lemma

Let T be a binary tree. The hight of the drawing constructed by Right-Heavy approach is at most $\log n$.

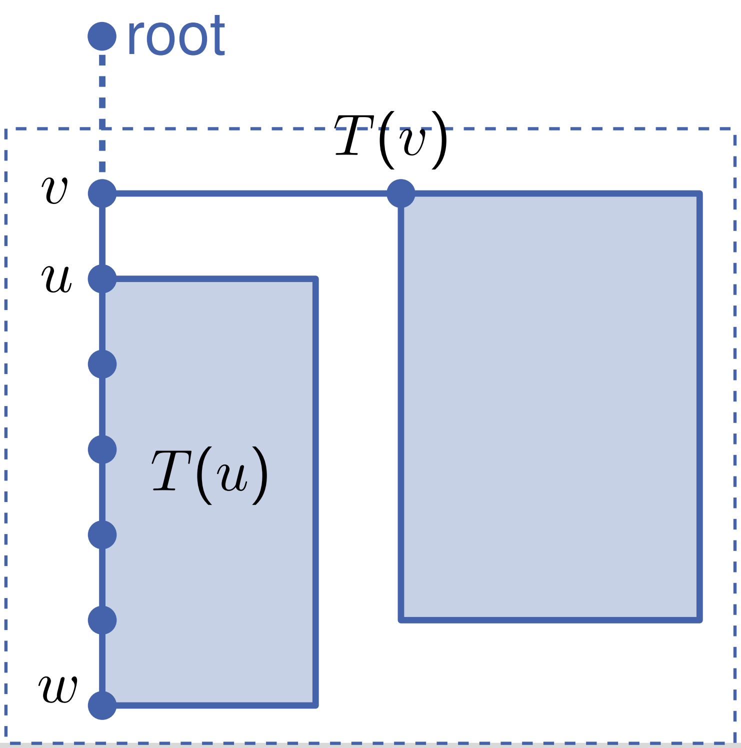
Right-Heavy HV-Layout

Right-Heavy approach:

- At every induction step apply horizontal combination
- Place the larger subtree to the right

Lemma

Let T be a binary tree. The hight of the drawing constructed by Right-Heavy approach is at most $\log n$.



Proof:

- Each vertical edge has length 1
- Let w be the lowest node in the drawing
- Let P be a path from w to the root of T
- For every edge (u, v) in P: $|T(v)| > 2|T(u)|$
- $\Rightarrow P$ contains at most $\log n$ edges

Right-Heavy HV-Layout

Theorem

Let T be a binary tree with n vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing Γ of T such that:

Right-Heavy HV-Layout

Theorem

Let T be a binary tree with n vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing Γ of T such that:

- Γ is HV-drawing (planar, orthogonal)

Right-Heavy HV-Layout

Theorem

Let T be a binary tree with n vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing Γ of T such that:

- Γ is HV-drawing (planar, orthogonal)
- The width of Γ is at most

Right-Heavy HV-Layout

Theorem

Let T be a binary tree with n vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing Γ of T such that:

- Γ is HV-drawing (planar, orthogonal)
- The width of Γ is at most n-1

Right-Heavy HV-Layout

Karlsruhe Institute of Technology

Theorem

Let T be a binary tree with n vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing Γ of T such that:

- Γ is HV-drawing (planar, orthogonal)
- The width of Γ is at most n-1
- The height is at most $\log n$

Right-Heavy HV-Layout

Theorem

Let T be a binary tree with n vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing Γ of T such that:

- Γ is HV-drawing (planar, orthogonal)
- The width of Γ is at most n-1
- The height is at most $\log n$
- The area is $O(n \log n)$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik I

Right-Heavy HV-Layout

Theorem

Let T be a binary tree with n vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing Γ of T such that:

- Γ is HV-drawing (planar, orthogonal)
- The width of Γ is at most n-1
- The height is at most $\log n$
- The area is $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings, up to translation

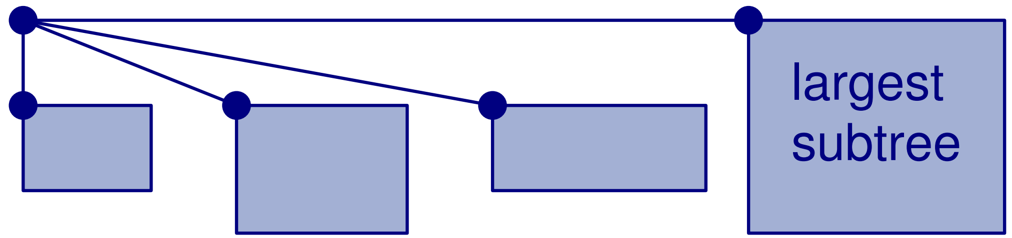
Right-Heavy HV-Layout

Theorem

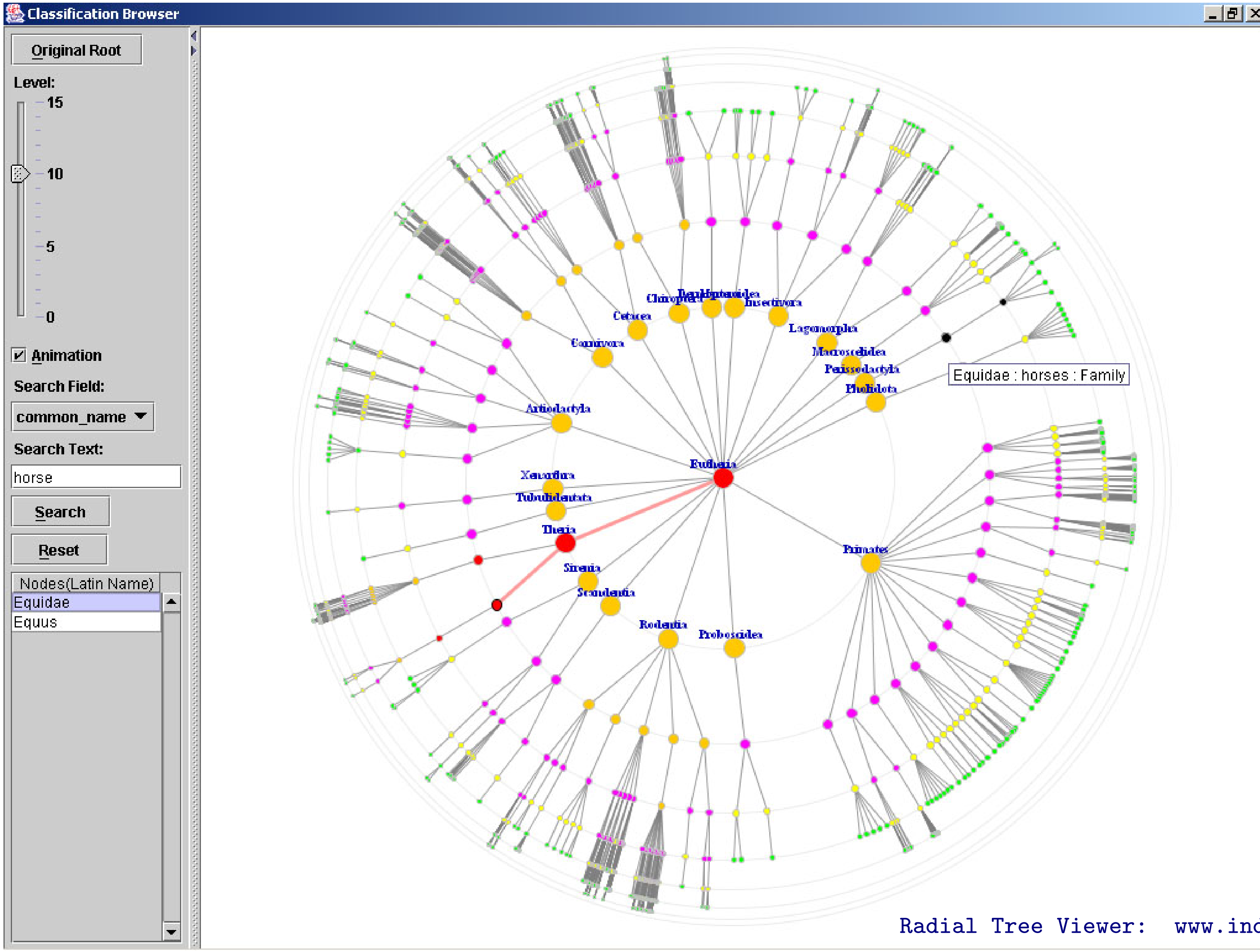
Let T be a binary tree with n vertices. The Right-Heavy algorithm constructs in $O(n)$ time a drawing Γ of T such that:

- Γ is HV-drawing (planar, orthogonal)
- The width of Γ is at most n-1
- The height is at most $\log n$
- The area is $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings, up to translation

General rooted tree:



Radial Layout

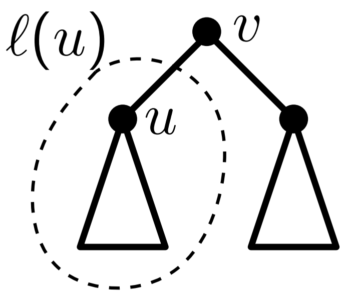


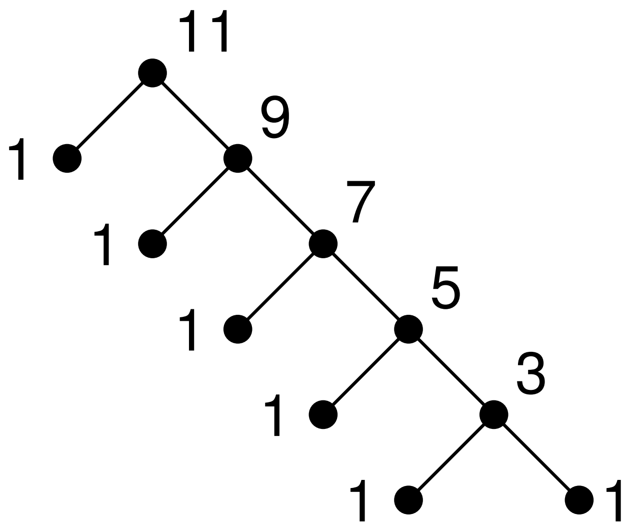
Radial Tree Viewer: www.indiana.edu

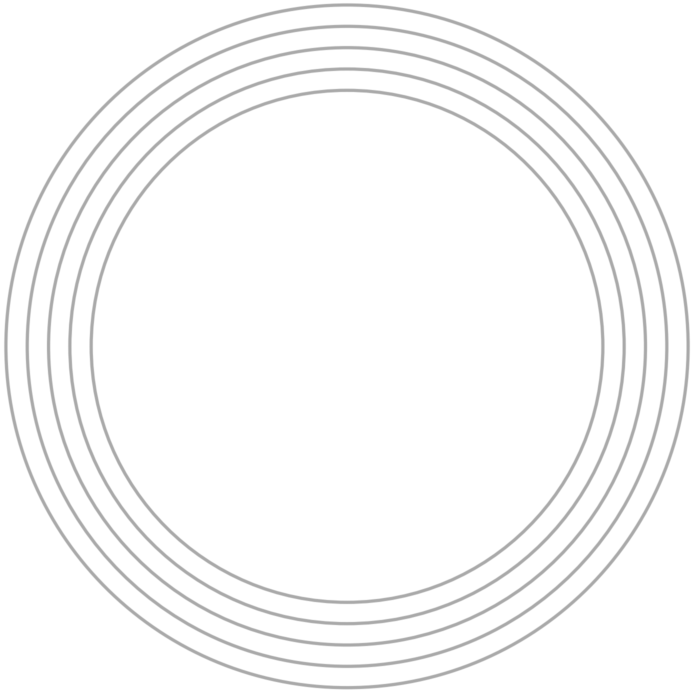
Radial Layout

Example:

- $\tau_u = \frac{\ell(u)}{\ell(v)-1}$



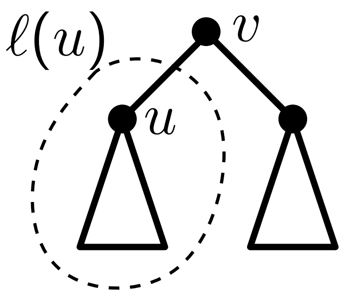


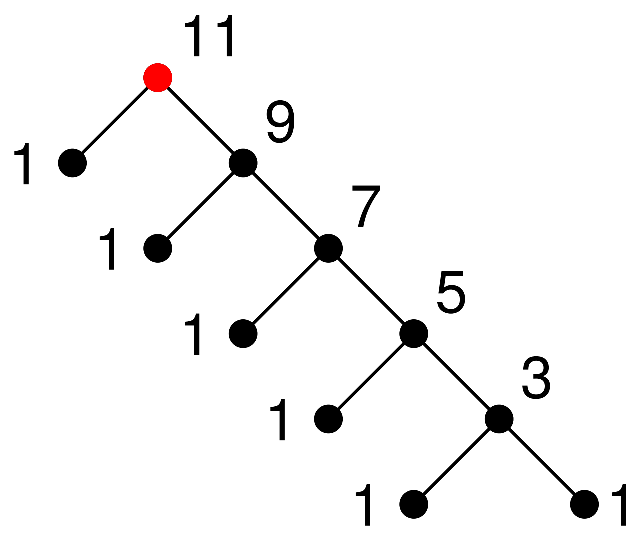


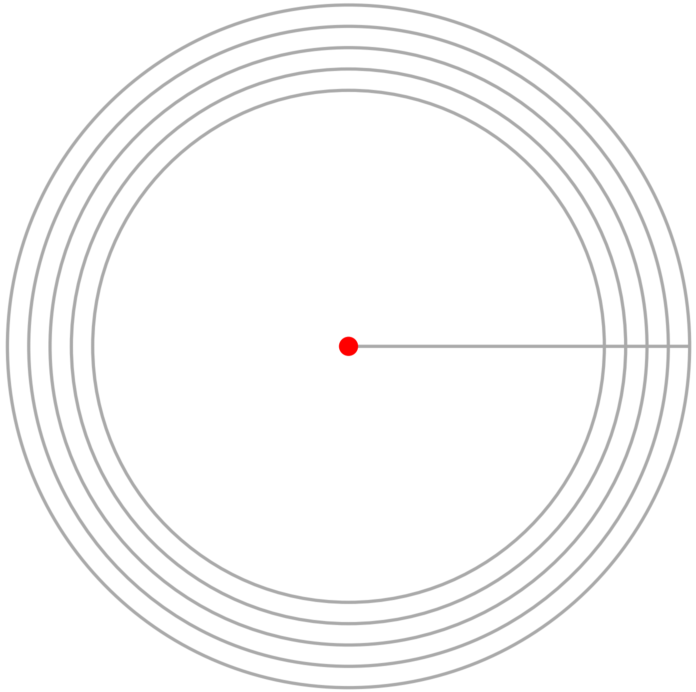
Radial Layout

Example:

- $\tau_u = \frac{\ell(u)}{\ell(v)-1}$



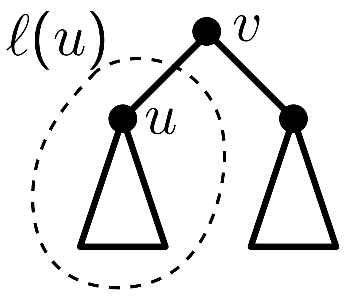


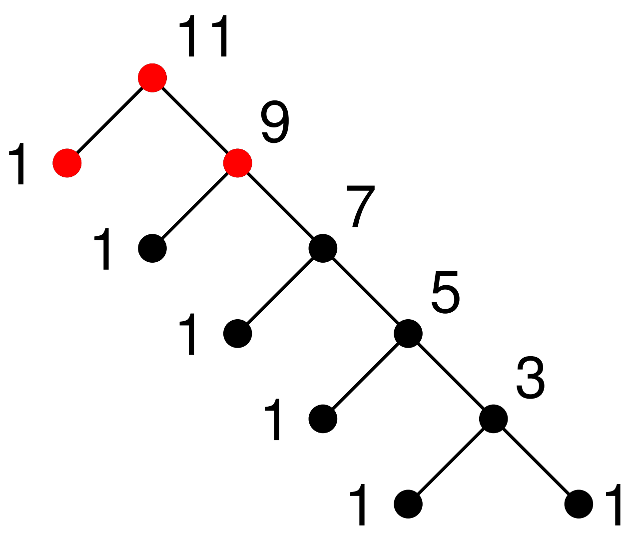


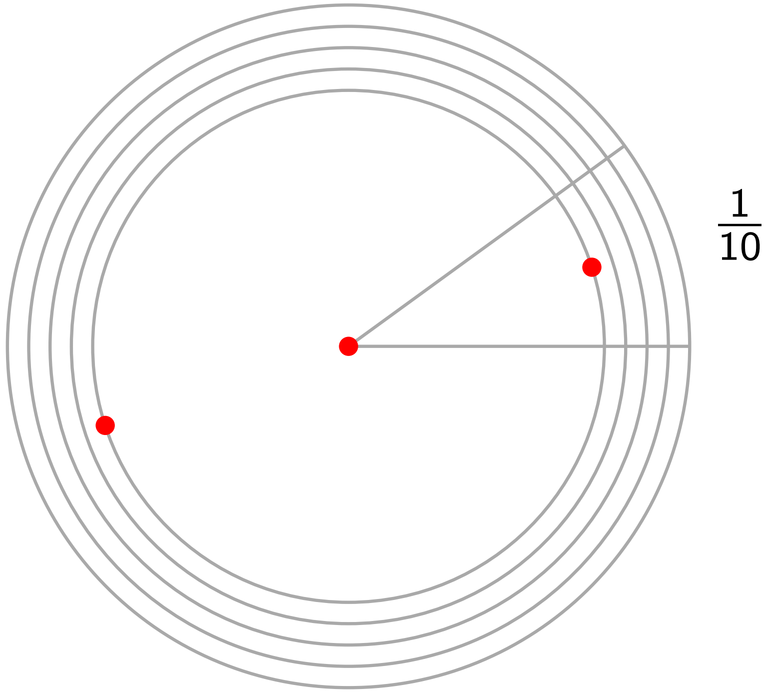
Radial Layout

Example:

- $\tau_u = \frac{\ell(u)}{\ell(v)-1}$



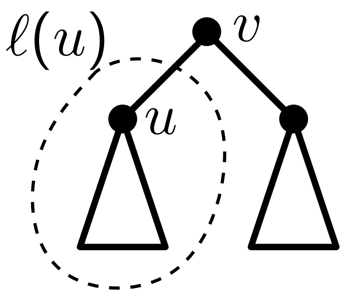


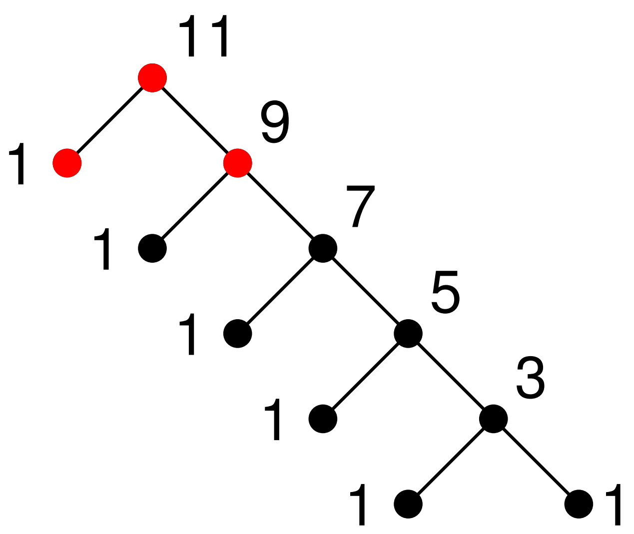


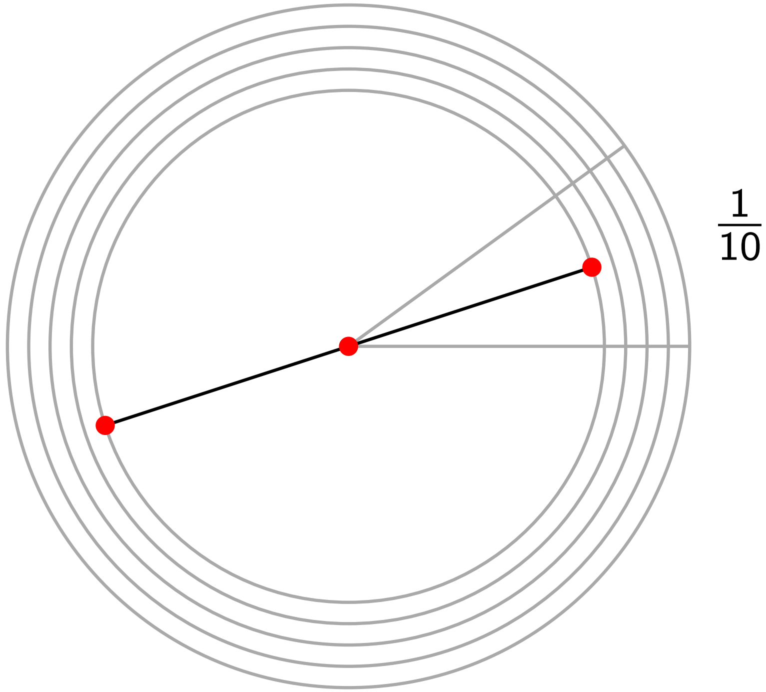
Radial Layout

Example:

- $\tau_u = \frac{\ell(u)}{\ell(v)-1}$



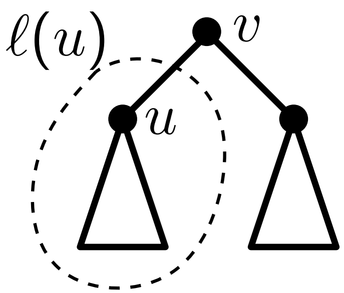


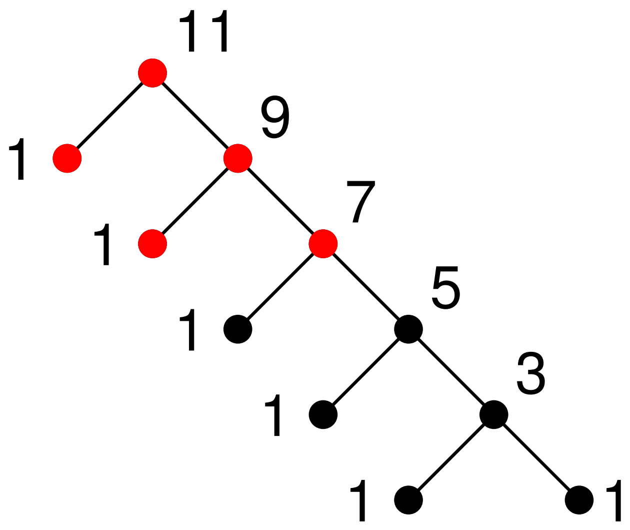


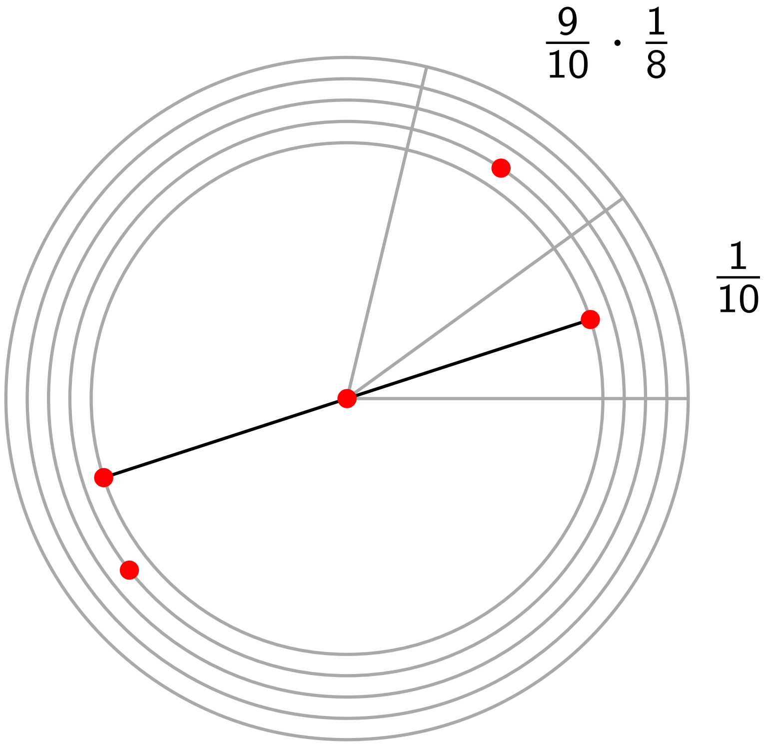
Radial Layout

Example:

- $\tau_u = \frac{\ell(u)}{\ell(v)-1}$



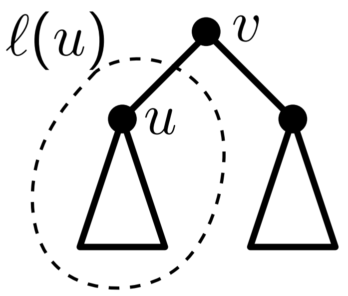


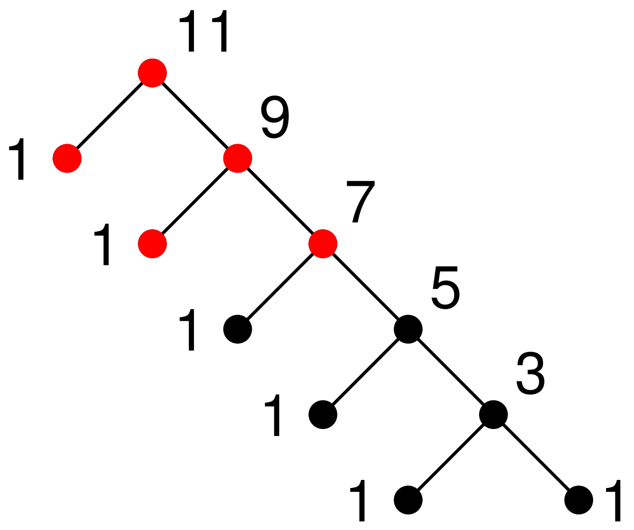


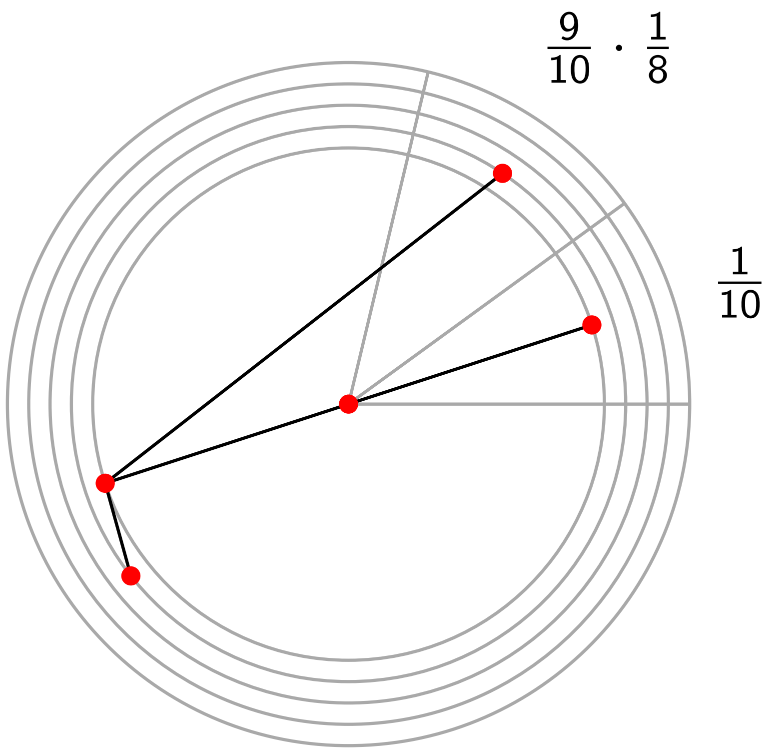
Radial Layout

Example:

- $\tau_u = \frac{\ell(u)}{\ell(v)-1}$



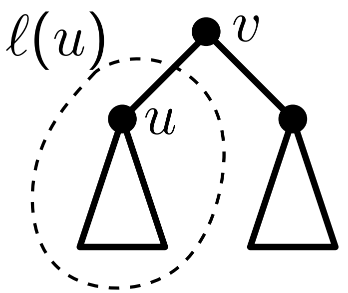


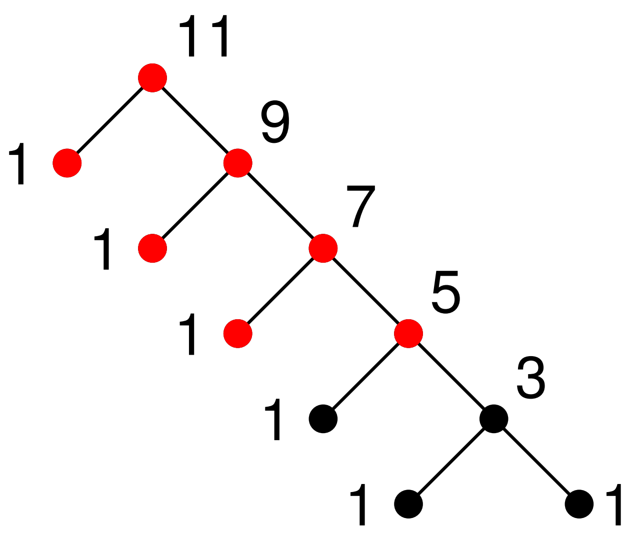


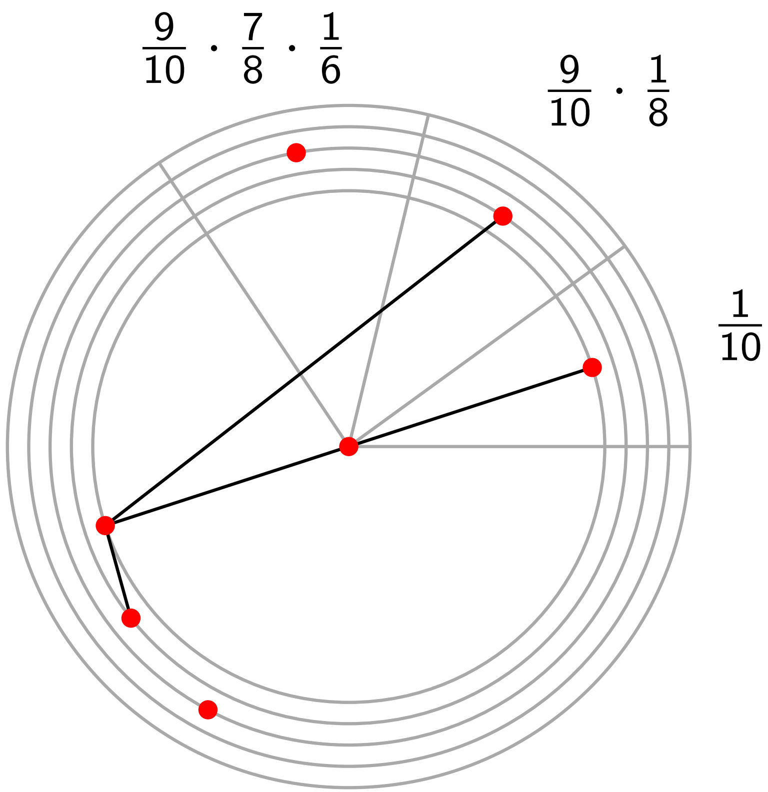
Radial Layout

Example:

- $\tau_u = \frac{\ell(u)}{\ell(v)-1}$



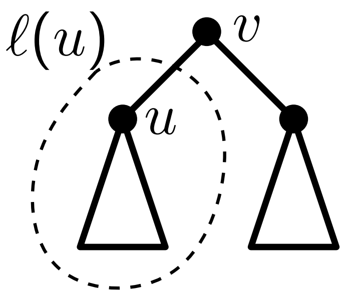


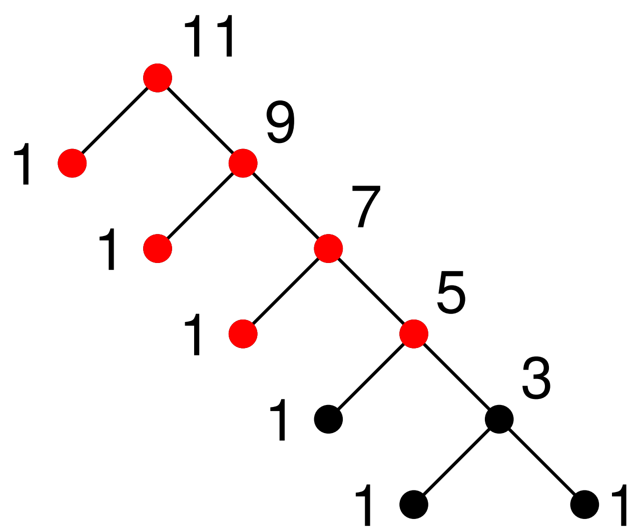


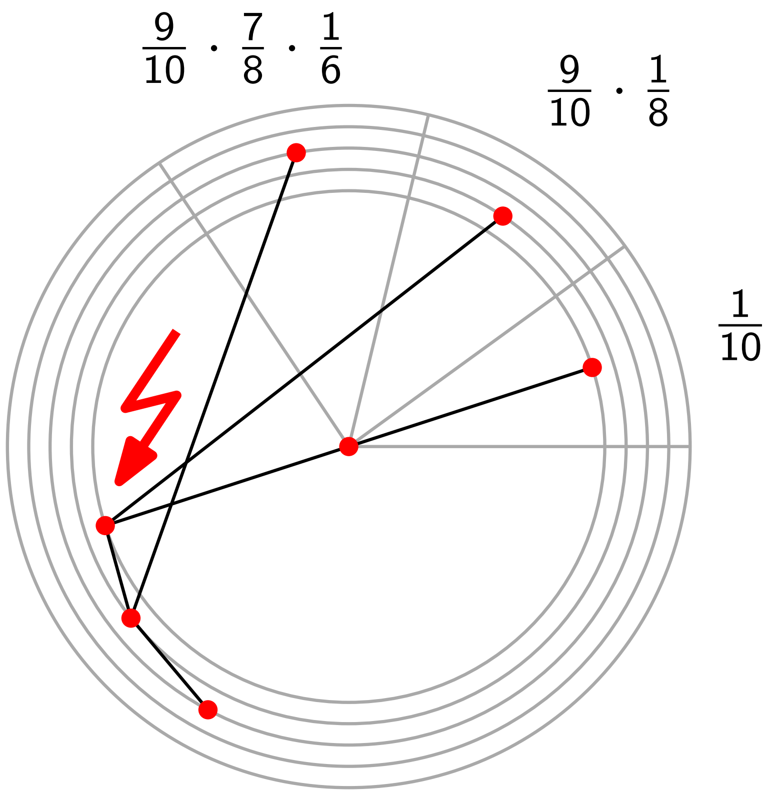
Radial Layout

Example:

- $\tau_u = \frac{\ell(u)}{\ell(v)-1}$

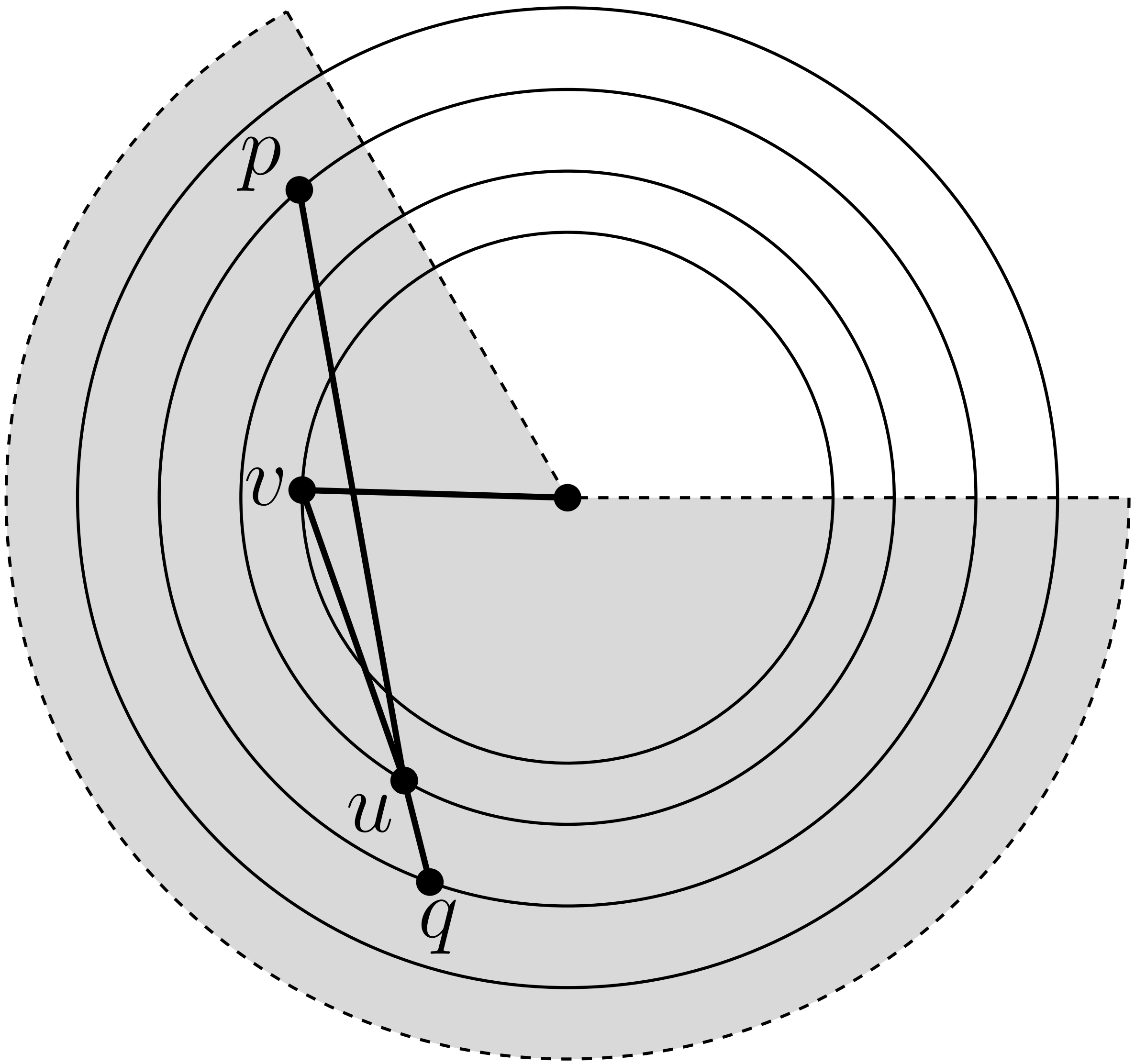






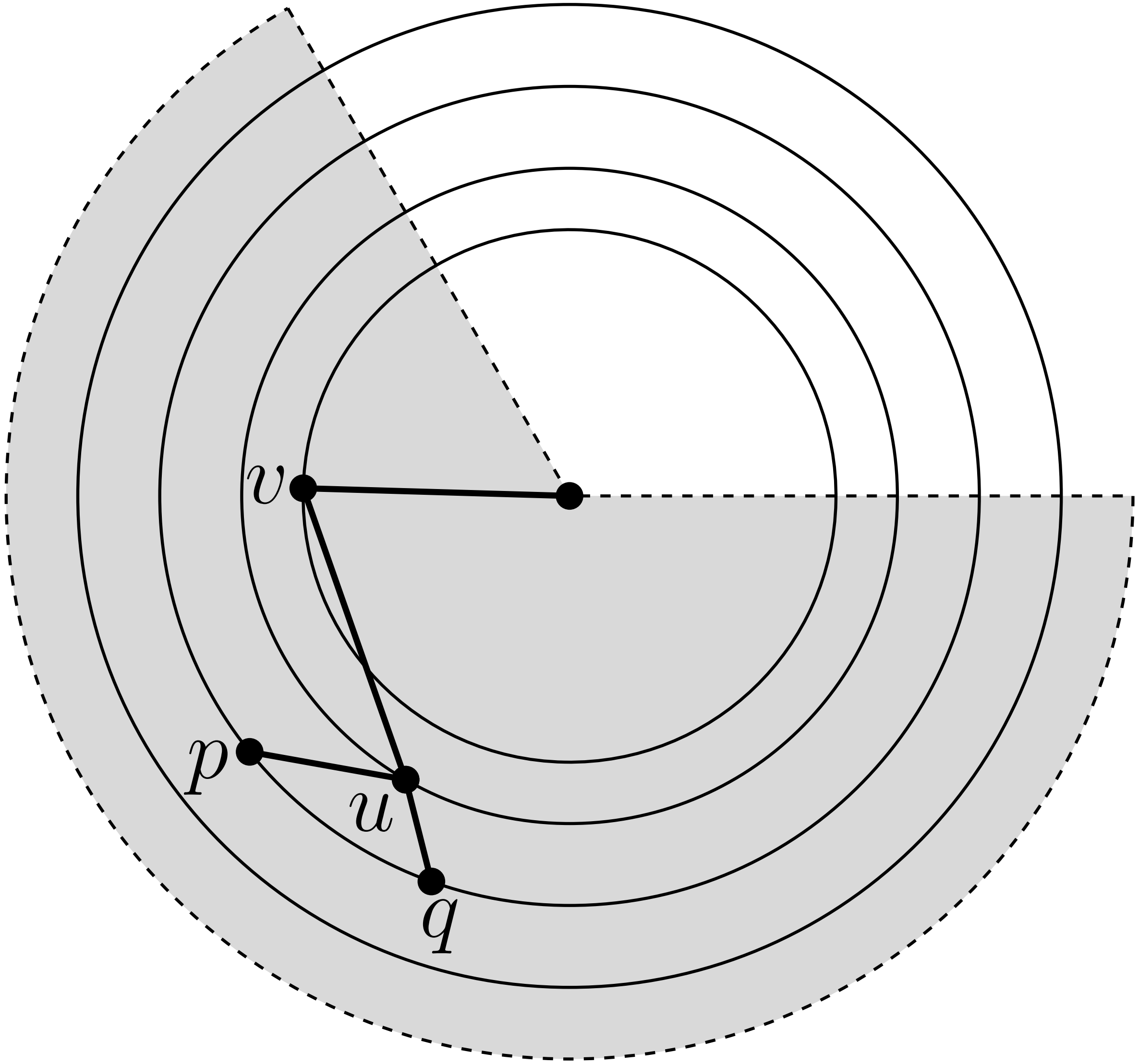
Radial Layout

How to avoid crossings:



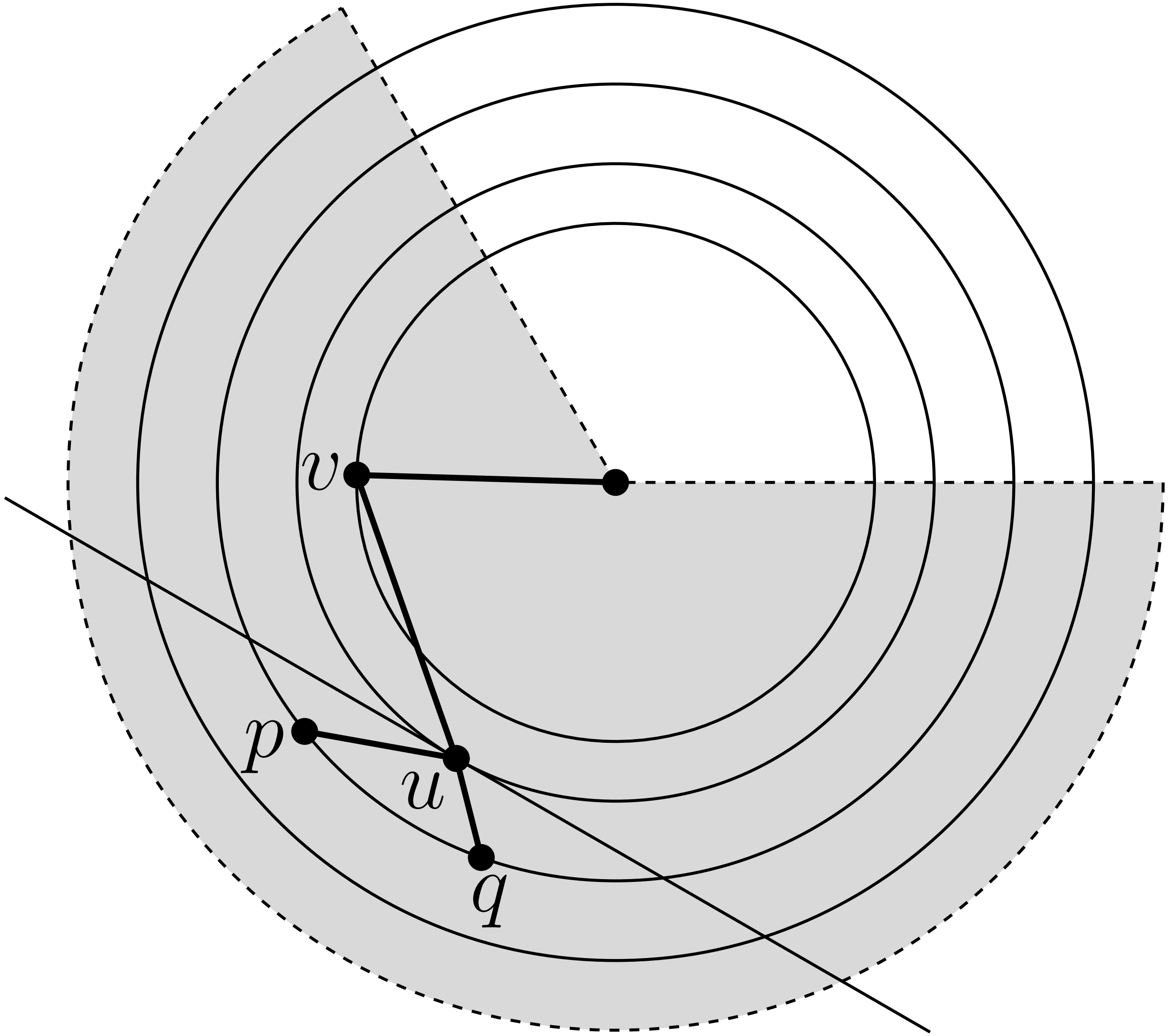
Radial Layout

How to avoid crossings:



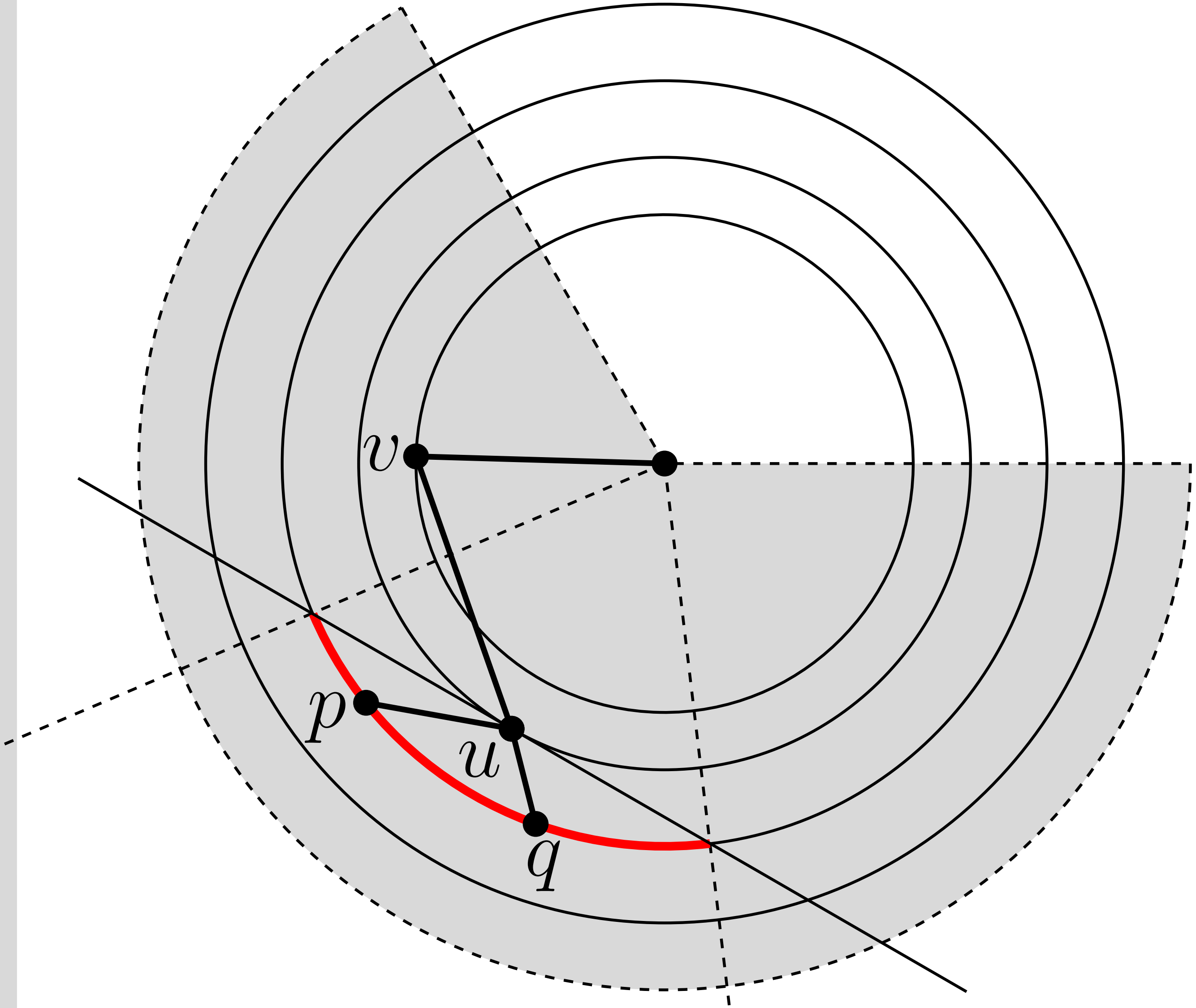
Radial Layout

How to avoid crossings:



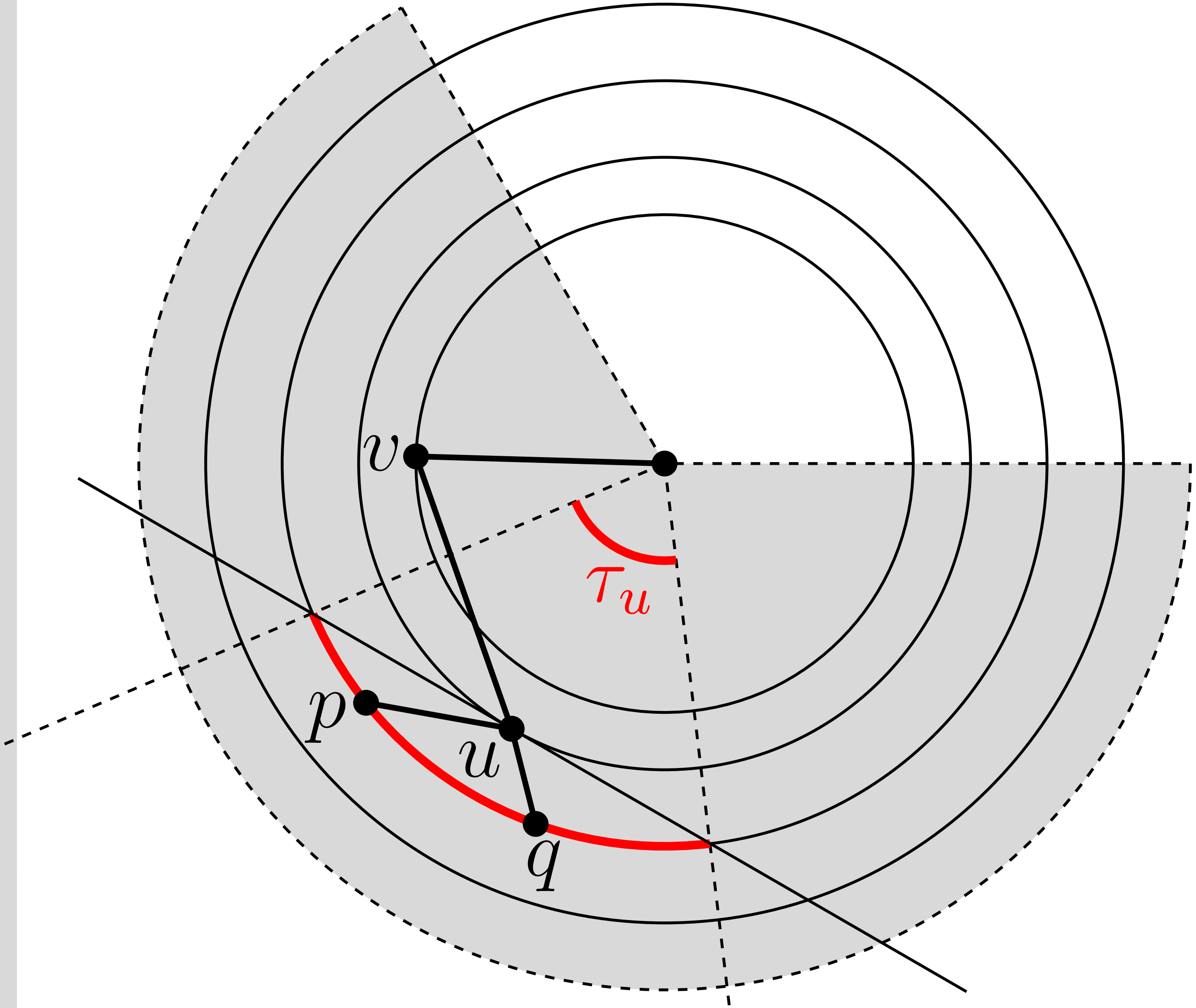
Radial Layout

How to avoid crossings:



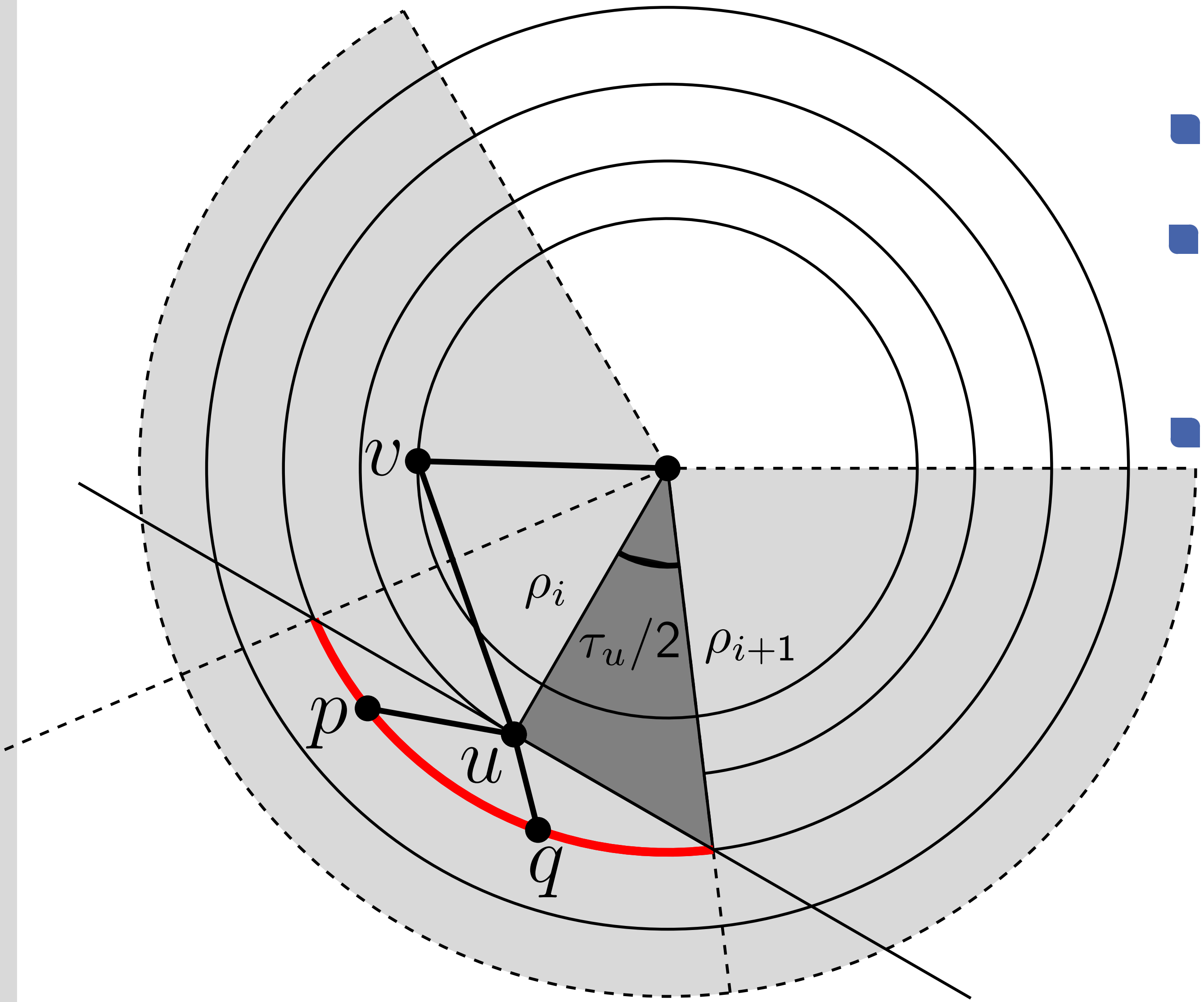
Radial Layout

How to avoid crossings:



Radial Layout

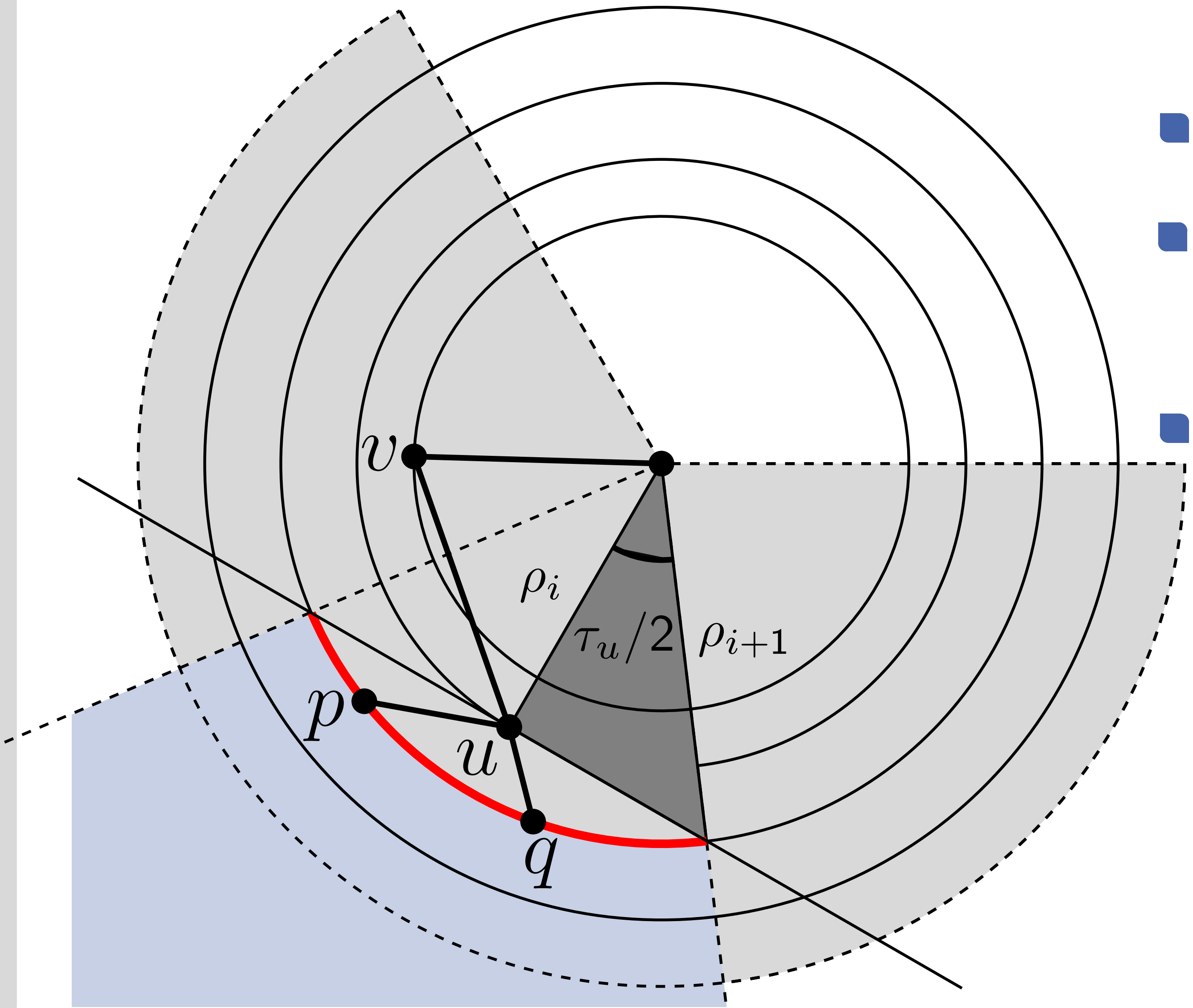
How to avoid crossings:



- τ_u - angle of the wedge corresponding to vertex u
- ρ_i - raduis of layer i
- $\ell(v)$-number of nodes in the subtree rooted at v
- $\cos\frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$

Radial Layout

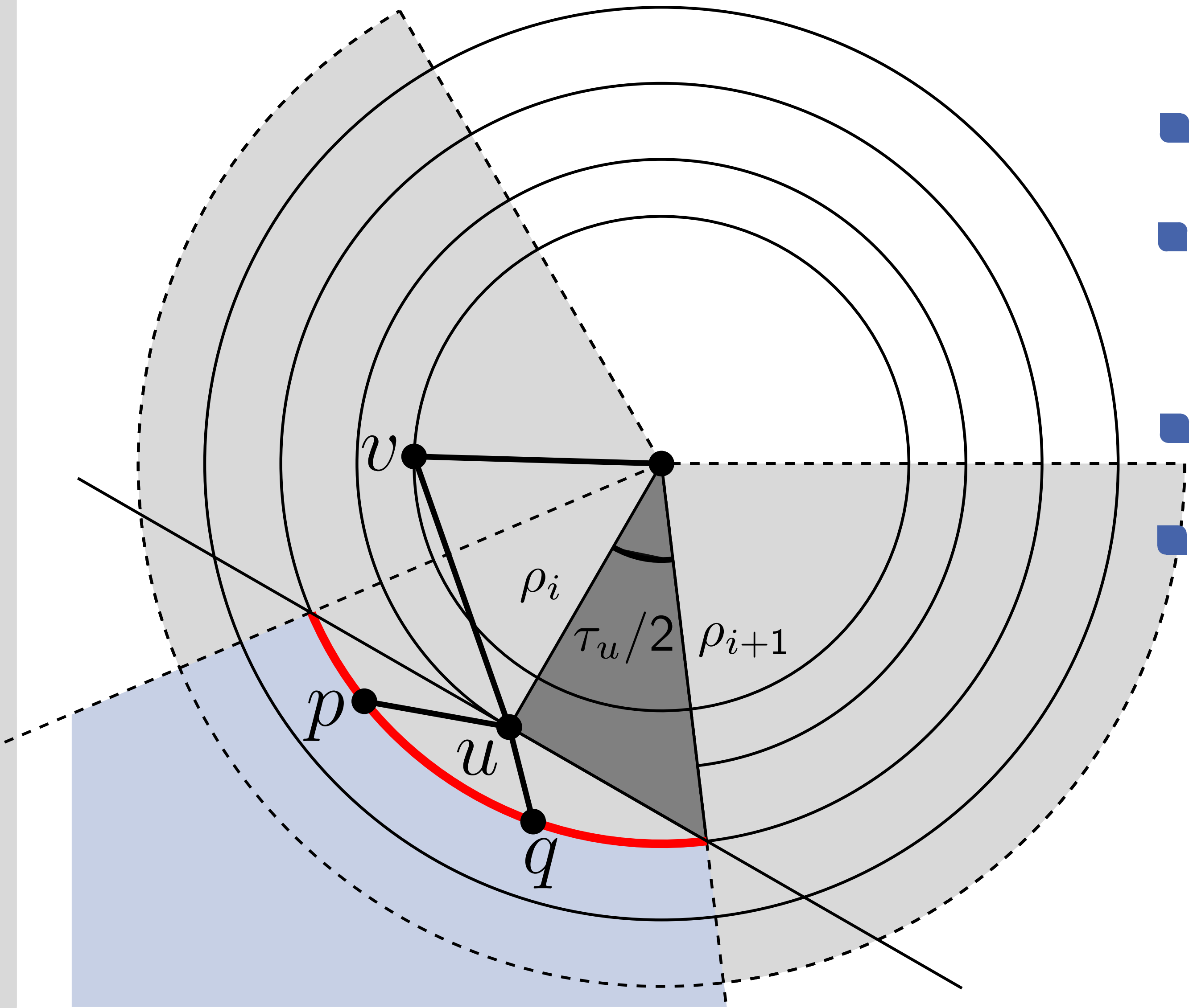
How to avoid crossings:



- τ_u - angle of the wedge corresponding to vertex u
- ρ_i - raduis of layer i
- $\ell(v)$-number of nodes in the subtree rooted at v
- $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$

Radial Layout

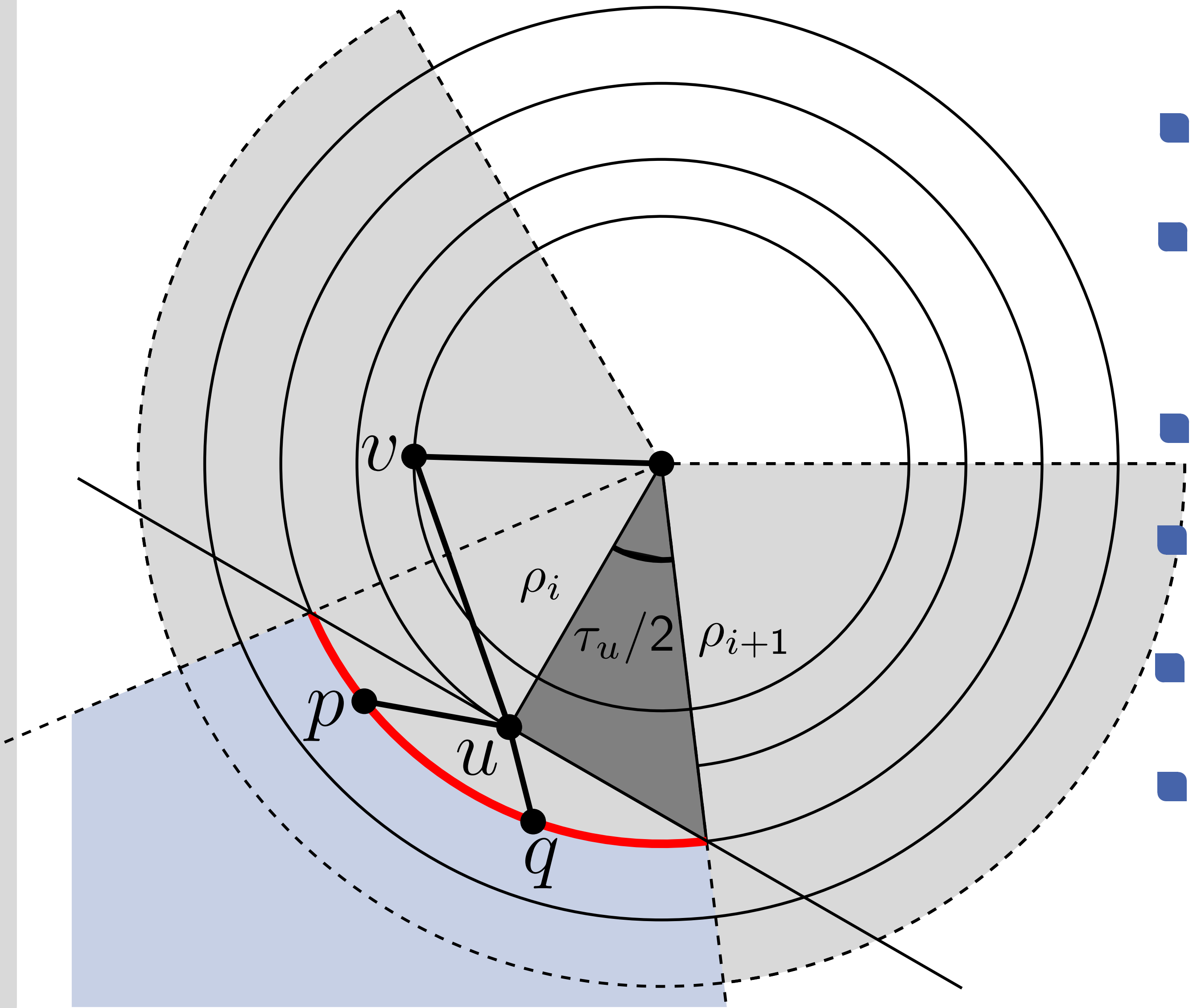
How to avoid crossings:



- τ_u - angle of the wedge corresponding to vertex u
- ρ_i - raduis of layer i
- $\ell(v)$-number of nodes in the subtree rooted at v
- $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$
- $\tau_u = 2 \arccos \frac{\rho_i}{\rho_{i+1}}$ (correction)

Radial Layout

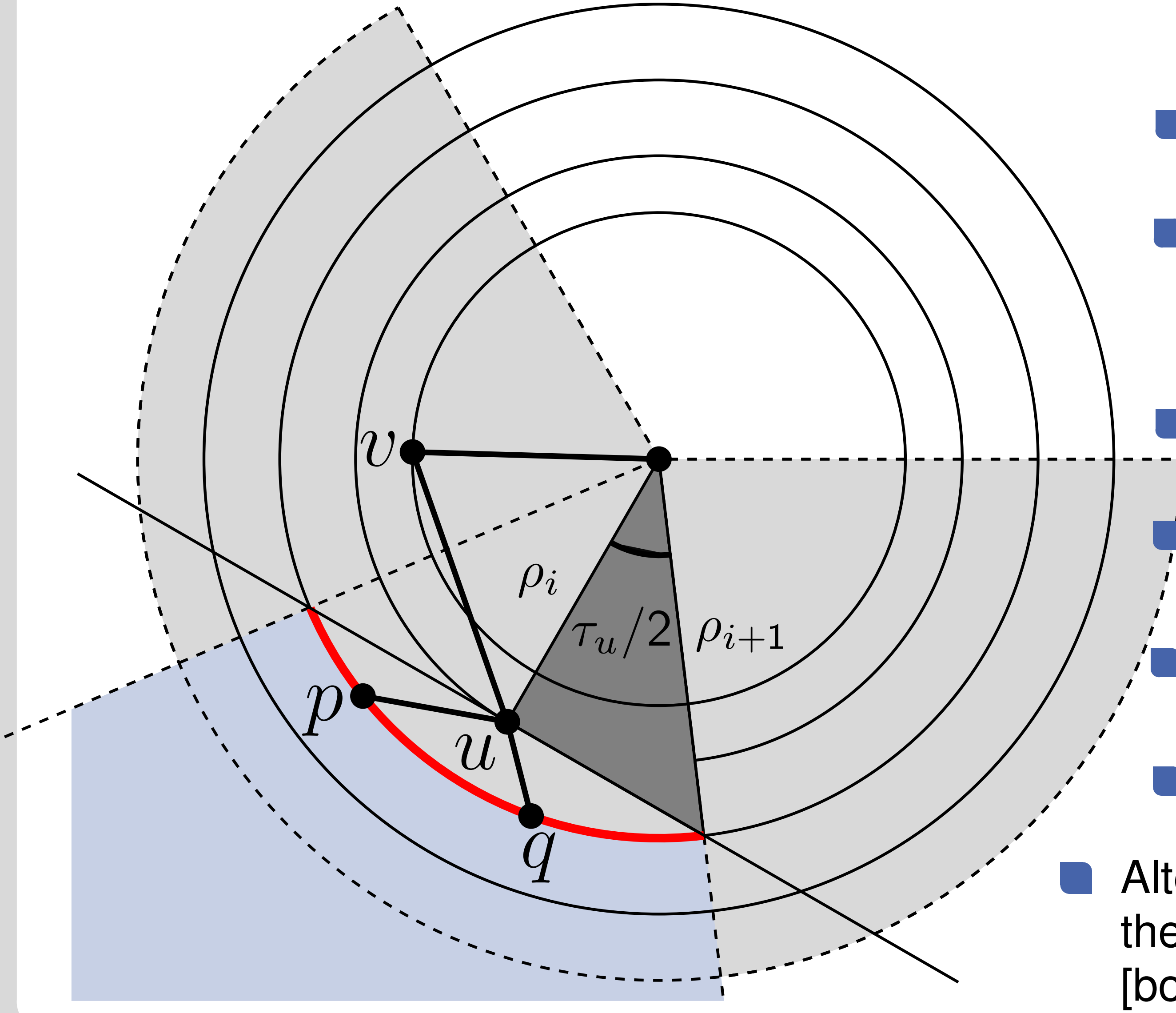
How to avoid crossings:



- τ_u - angle of the wedge corresponding to vertex u
- ρ_i - raduis of layer i
- $\ell(v)$-number of nodes in the subtree rooted at v
- $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$
- $\tau_u = 2 \arccos \frac{\rho_i}{\rho_{i+1}}$ (correction)
- $\tau_p = \frac{\ell(p)}{\ell(u)-1} \tau_u$
- $\tau_q = \frac{\ell(q)}{\ell(u)-1} \tau_u$

Radial Layout

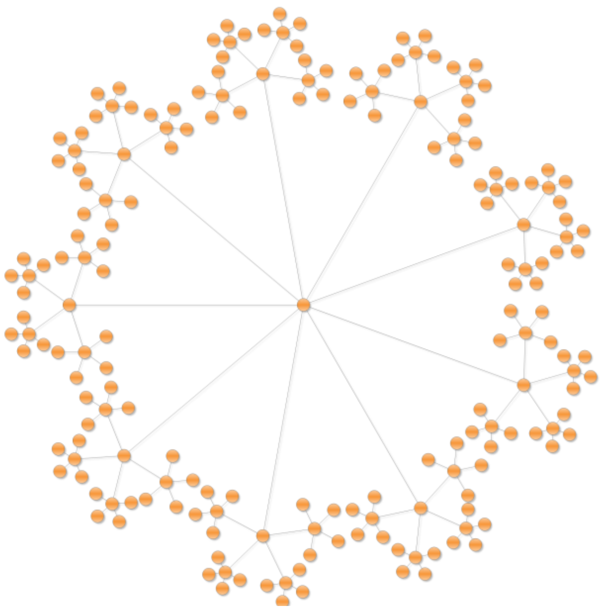
How to avoid crossings:



- τ_u - angle of the wedge corresponding to vertex u
- ρ_i - raduis of layer i
- $\ell(v)$-number of nodes in the subtree rooted at v
- $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$
- $\tau_u = 2 \arccos \frac{\rho_i}{\rho_{i+1}}$ (correction)
- $\tau_p = \frac{\ell(p)}{\ell(u)-1}\tau_u$
- $\tau_q = \frac{\ell(q)}{\ell(u)-1}\tau_u$
- Alternatively use number of leaves in the subtree to subdivide the angles [book Di Battista et al.]

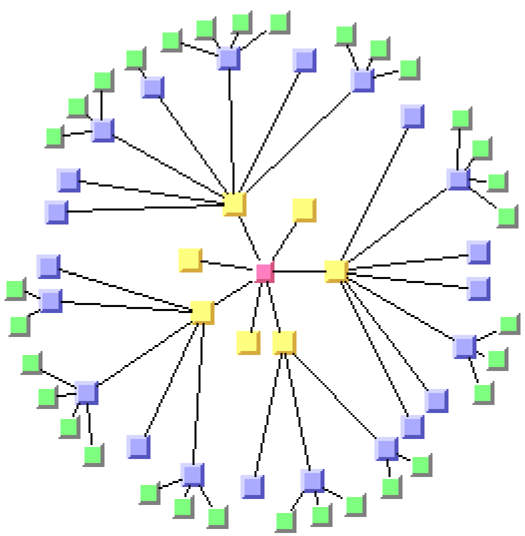
Baloon Layout

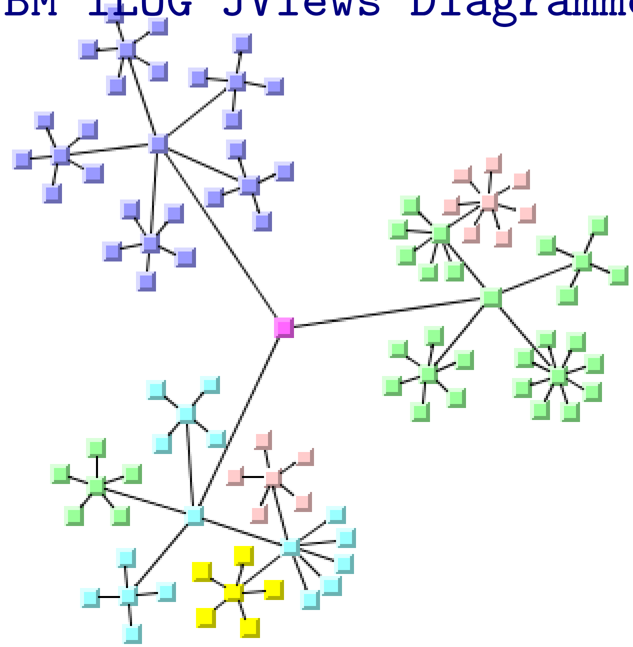
Karlsruhe Institute of Technology

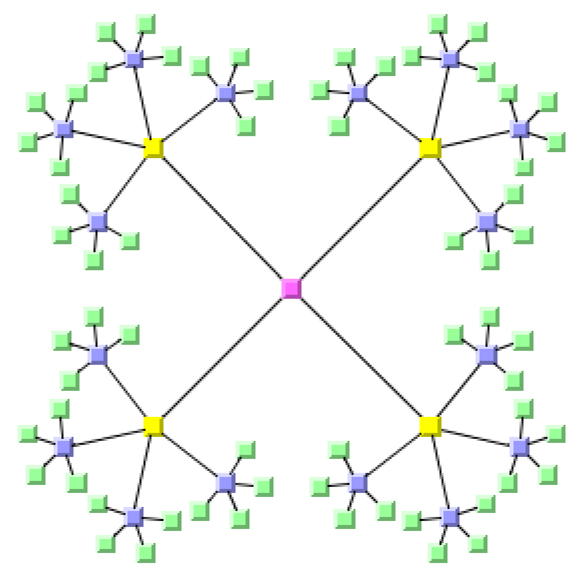


NEVRON-Visualize your success: www.nevron.com

IBM ILOG JViews Diagrammer: www.ibm.com



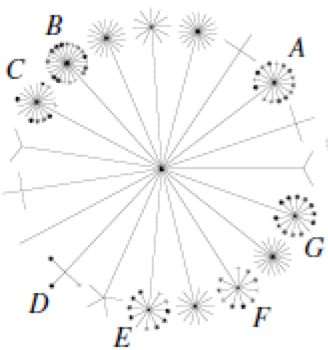




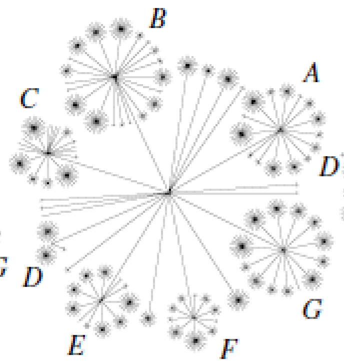
Baloon Layout

A baloon drawing has the following properties:

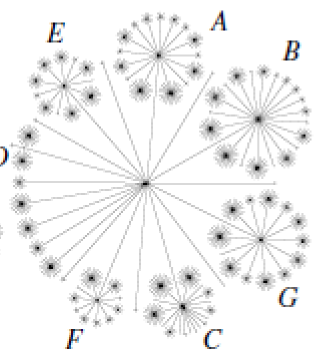
- All the children of the same parent lie on circle centered at their parent
- The drawing is planar
- The further an edge from the root is, the shorter it becomes



All subtrees at the same depth have the same size.



Subtrees may have different size, the tree is ordered.

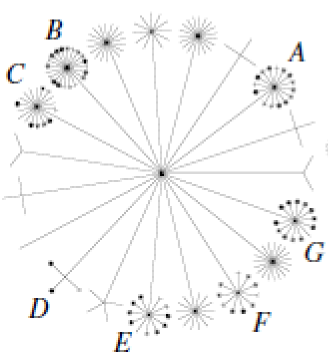


Subtrees may have different size, the tree is unordered. Drawn by Lin& Yen Algorithm.

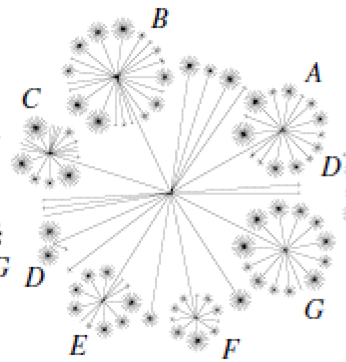
Baloon Layout

A baloon drawing has the following properties:

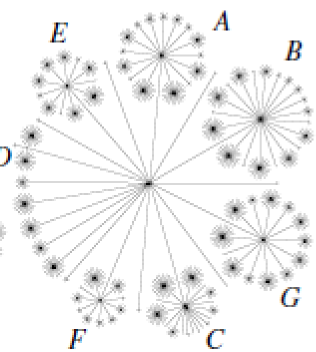
- All the children of the same parent lie on circle centered at their parent
- The drawing is planar
- The further an edge from the root is, the shorter it becomes



All subtrees at the same depth have the same size.



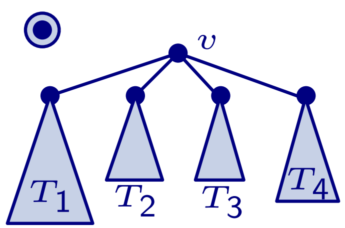
Subtrees may have different size, the tree is ordered.

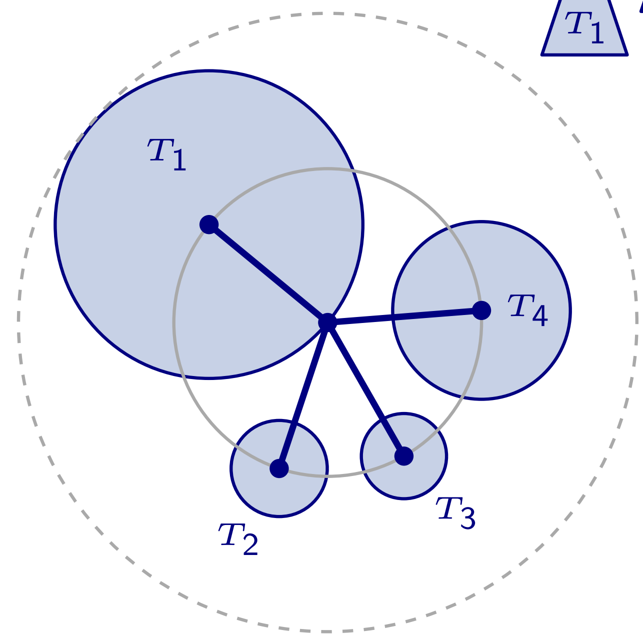


Subtrees may have different size, the tree is unordered. Drawn by Lin& Yen Algorithm.

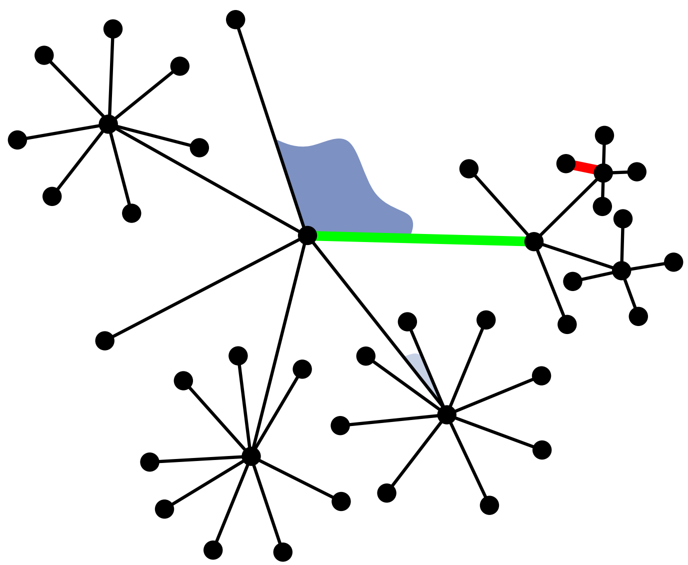
Induction base:

Induction step:





Baloon Layout

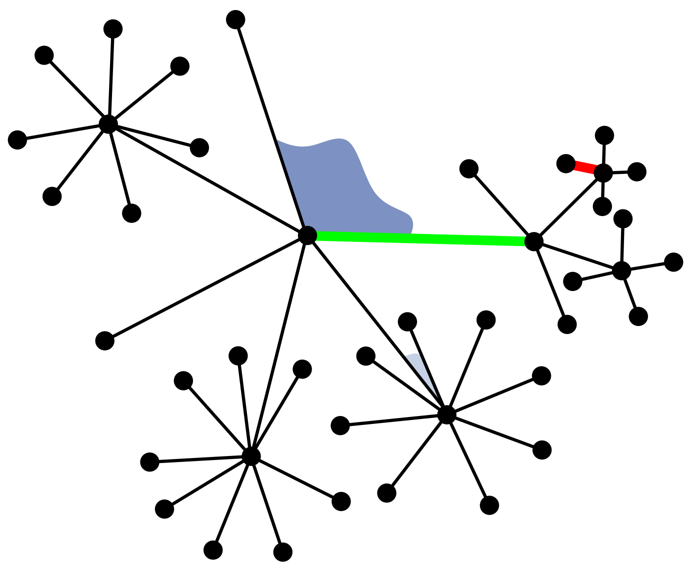


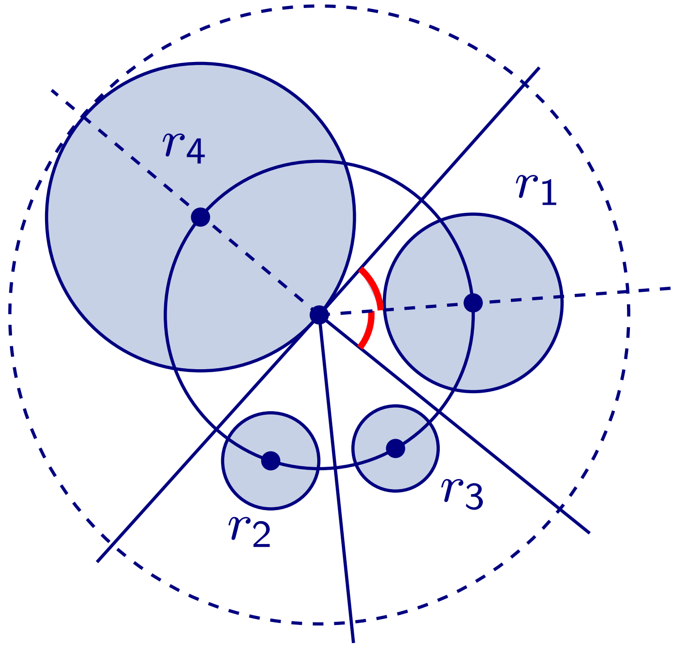
Aesthetics:

- Aspect ratio $= \frac{\text{largest angle}}{\text{smallest angle}}$
- Angular resolution $=$ $\min\{\text{angle between two adjacent edges}\}$

Question: Can we find a baloon drawing with max angular resolution and min aspect ratio in an unordered tree? (Algorithm by Lin & Yen)

Baloon Layout





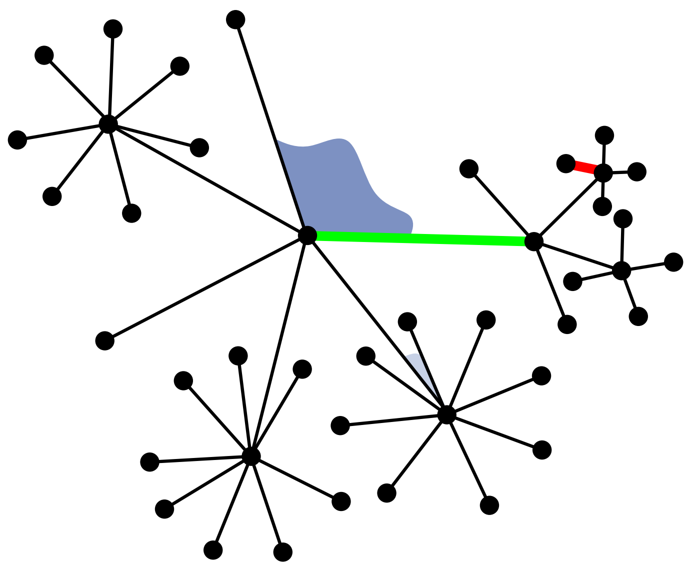
Aesthetics:

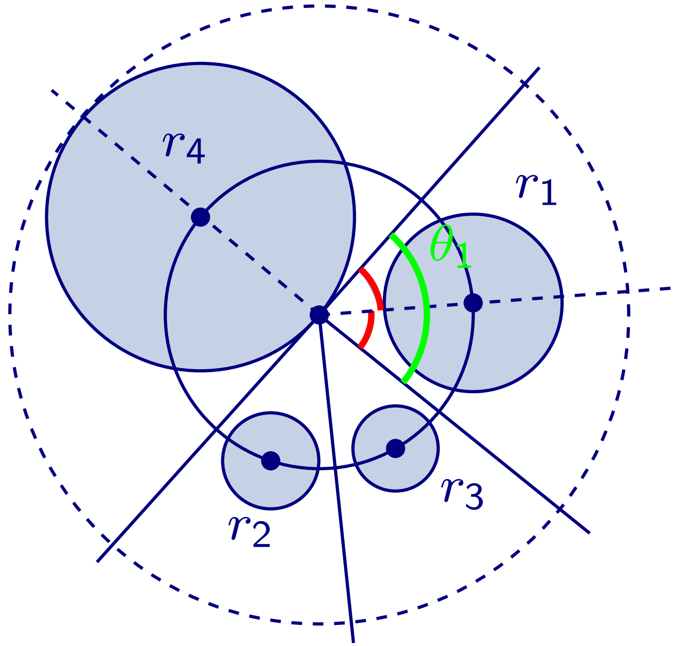
- Aspect ratio $= \frac{\text{largest angle}}{\text{smallest angle}}$
- Angular resolution $=$ $\min\{\text{angle between two adjacent edges}\}$

Question: Can we find a baloon drawing with max angular resolution and min aspect ratio in an unordered tree? (Algorithm by Lin & Yen)

- We investigate drawing with **even angles** (drawing with uneven angles might have a better area)

Baloon Layout





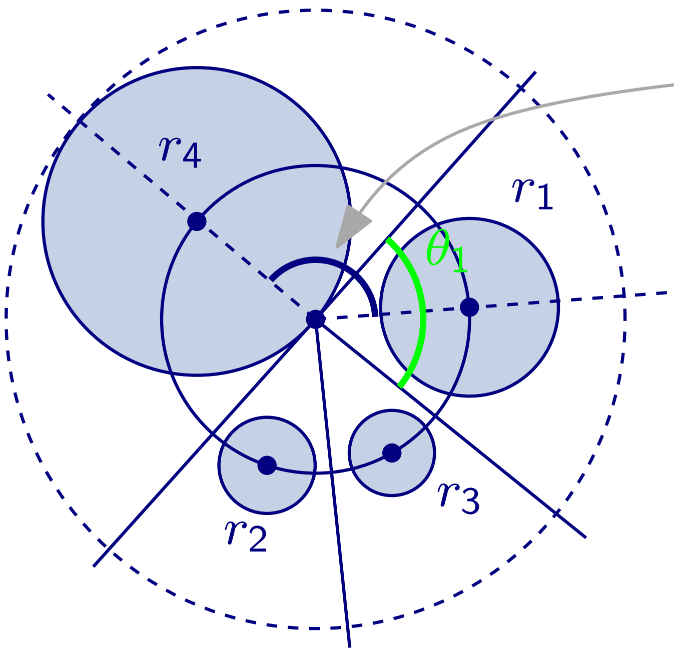
Aesthetics:

- Aspect ratio $= \frac{\text{largest angle}}{\text{smallest angle}}$
- Angular resolution $=$ $\min\{\text{angle between two adjacent edges}\}$

Question: Can we find a baloon drawing with max angular resolution and min aspect ratio in an unordered tree? (Algorithm by Lin & Yen)

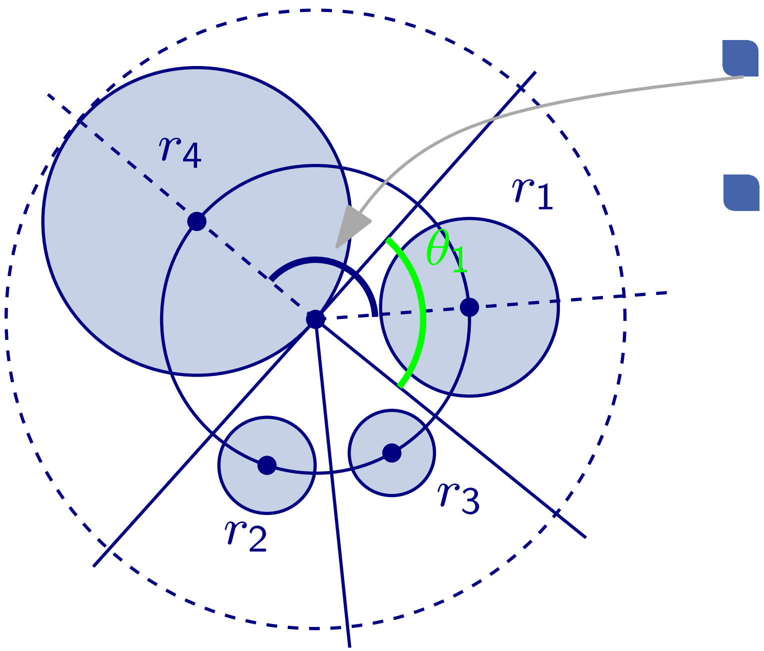
- We investigate drawing with **even angles** (drawing with uneven angles might have a better area)
- An arrangement of the subtree at a level can be describeed by a permutation $\sigma = \{1, 4, 2, 3\}$
- θ_i - angle of the wedge containing the circle r_i

Baloon Layout. Algorithm by Lin & Yen.



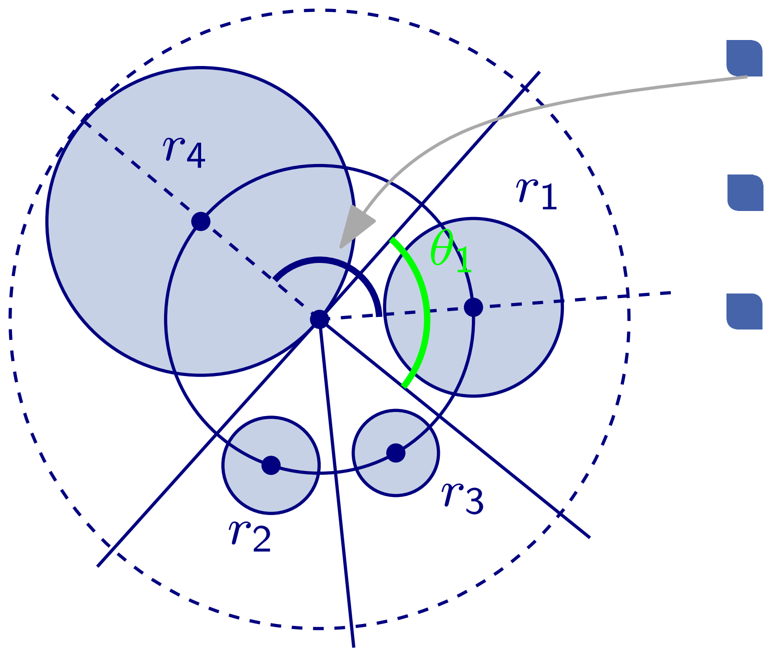
- $\frac{\theta_{\sigma_i} + \theta_{\sigma_{i+1}}}{2}$ - angle between two edges

Baloon Layout. Algorithm by Lin & Yen.



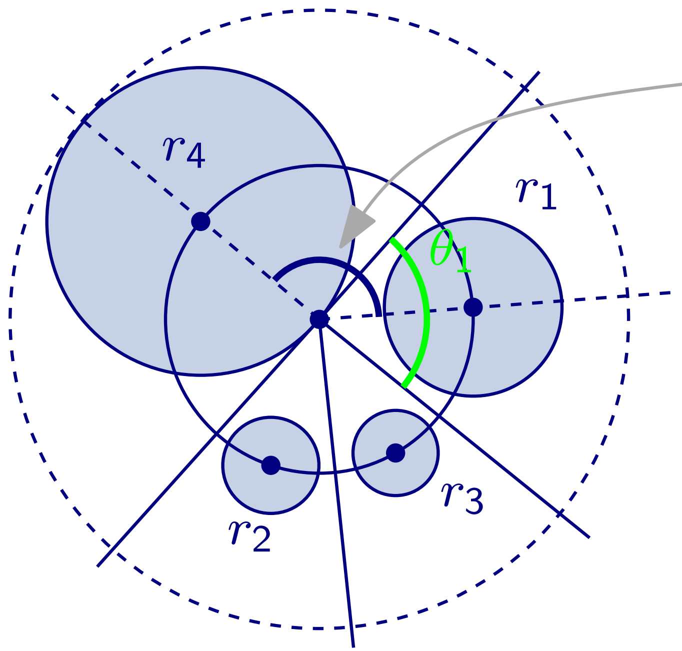
- $\frac{\theta_{\sigma_i} + \theta_{\sigma_{i+1}}}{2}$ - angle between two edges
- $AngResl_\sigma = \min_{1 \leq i \leq n}\{\frac{\theta_{\sigma_i} + \theta_{\sigma_{i+1}}}{2}\}$

Baloon Layout. Algorithm by Lin & Yen.



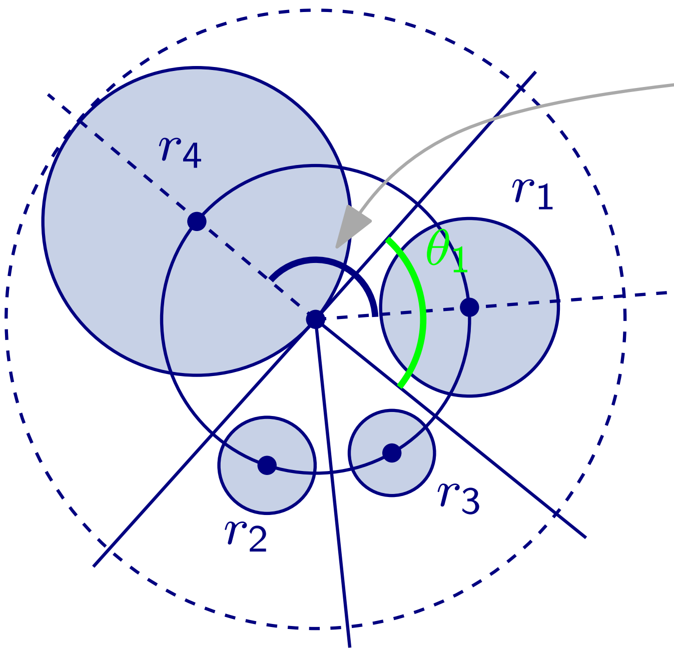
- $\frac{\theta_{\sigma_i} + \theta_{\sigma_{i+1}}}{2}$ - angle between two edges
- $AngResl_\sigma = \min_{1 \leq i \leq n}\{\frac{\theta_{\sigma_i} + \theta_{\sigma_{i+1}}}{2}\}$
- Permute the circles (σ) so that the $AngleResl_\sigma$ is minimied.

Baloon Layout. Algorithm by Lin & Yen.



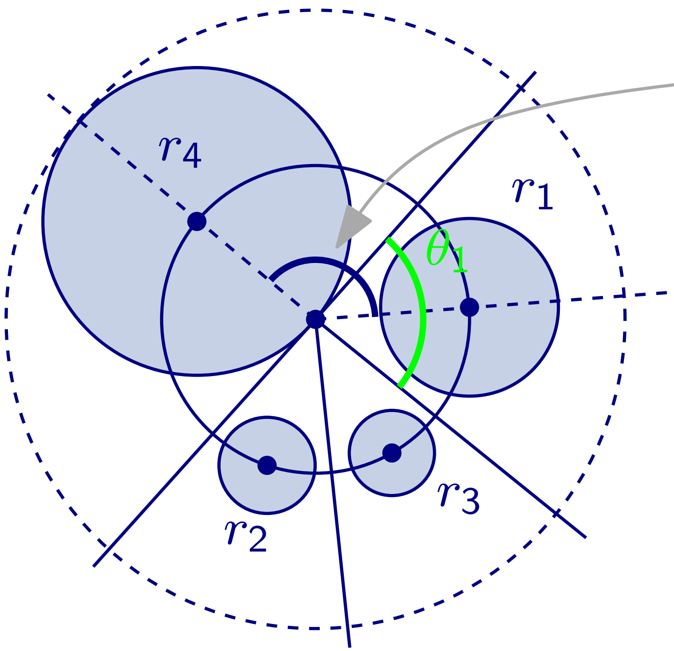
- $\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}$ - angle between two edges
- $AngResl_\sigma = \min_{1\leq i\leq n}\{\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}\}$
- Permute the circles (σ) so that the $AngleResl_\sigma$ is minimied.
- Let $m_1, m_2, \ldots, m_k, mid, M_k, M_{k-1}, \ldots, M_2, M_1$ be the angles θ in the increasing ordering, i.e. m_i (M_i) is i-th minimum (maximum), mid-unique medium, in case of odd number of circles.

Baloon Layout. Algorithm by Lin & Yen.



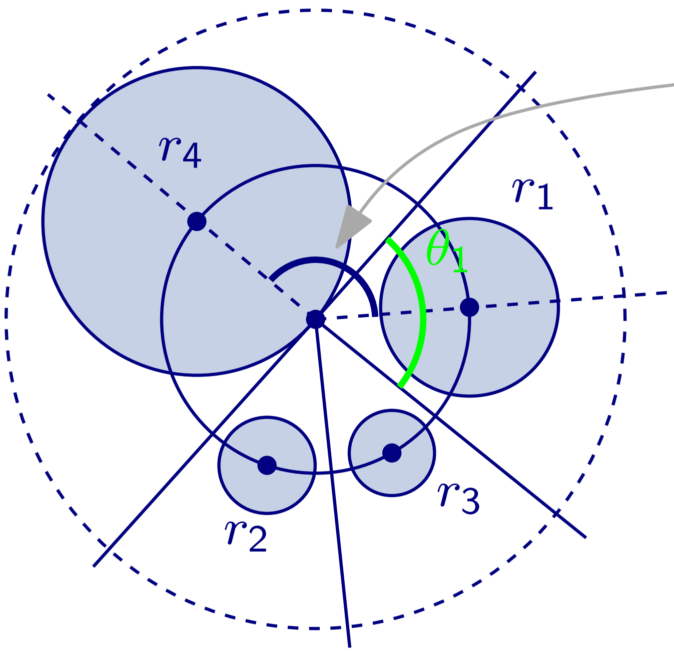
- $\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}$ - angle between two edges
- $AngResl_\sigma = \min_{1\leq i\leq n}\{\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}\}$
- Permute the circles (σ) so that the $AngleResl_\sigma$ is minimied.
- Let $m_1, m_2, \ldots, m_k, mid, M_k, M_{k-1}, \ldots, M_2, M_1$ be the angles θ in the increasing ordering, i.e. m_i (M_i) is i-th minimum (maximum), mid-unique medium, in case of odd number of circles.
- Let $\sigma = \{M_1, m_2, M_3, m_4, \ldots, M_{k-1}, m_k, mid, M_k, m_{k-1}, \ldots, M_4, m_3, M_2, m_1\}$. We show that σ gives minimum angle resolution.

Baloon Layout. Algorithm by Lin & Yen.



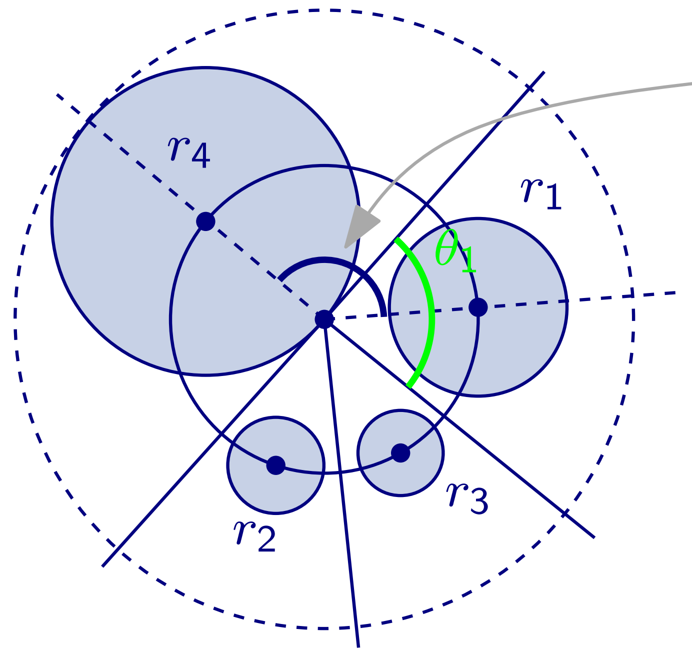
- $\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}$ - angle between two edges
- $AngResl_\sigma = \min_{1\leq i\leq n}\{\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}\}$
- Permute the circles (σ) so that the $AngleResl_\sigma$ is minimied.
- Let $m_1, m_2, \ldots, m_k, mid, M_k, M_{k-1}, \ldots, M_2, M_1$ be the angles θ in the increasing ordering, i.e. m_i (M_i) is i-th minimum (maximum), mid-unique medium, in case of odd number of circles.
- Let $\sigma = \{M_1, m_2, M_3, m_4, \ldots, M_{k-1}, m_k, mid, M_k, m_{k-1}, \ldots, M_4, m_3, M_2, m_1\}$. We show that σ gives minimum angle resolution.
- Let $\alpha_{ij} = \frac{M_i+m_j}{2}$. Angles $\frac{mid+m_k}{2}, \alpha_{(i-1)i}, \frac{M_k+mid}{2}, \alpha_{i(i-1)}$ are in σ.

Baloon Layout. Algorithm by Lin & Yen.



- $\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}$ - angle between two edges
- $AngResl_\sigma = \min_{1\leq i\leq n}\{\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}\}$
- Permute the circles (σ) so that the $AngleResl_\sigma$ is minimied.
- Let $m_1, m_2, \ldots, m_k, mid, M_k, M_{k-1}, \ldots, M_2, M_1$ be the angles θ in the increasing ordering, i.e. m_i (M_i) is i-th minimum (maximum), mid-unique medium, in case of odd number of circles.
- Let $\sigma = \{M_1, m_2, M_3, m_4, \ldots, M_{k-1}, m_k, mid, M_k, m_{k-1}, \ldots, M_4, m_3, M_2, m_1\}$. We show that σ gives minimum angle resolution.
- Let $\alpha_{ij} = \frac{M_i+m_j}{2}$. Angles $\frac{mid+m_k}{2}, \alpha_{(i-1)i}, \frac{M_k+mid}{2}, \alpha_{i(i-1)}$ are in σ.
- Relations among α_{ij}:

Baloon Layout. Algorithm by Lin & Yen.



- $\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}$ - angle between two edges
- $AngResl_\sigma = \min_{1\leq i\leq n}\{\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}\}$
- Permute the circles (σ) so that the $AngleResl_\sigma$ is minimied.
- Let $m_1, m_2, \ldots, m_k, mid, M_k, M_{k-1}, \ldots, M_2, M_1$ be the angles θ in the increasing ordering, i.e. m_i (M_i) is i-th minimum (maximum), mid-unique medium, in case of odd number of circles.
- Let $\sigma = \{M_1, m_2, M_3, m_4, \ldots, M_{k-1}, m_k, mid, M_k, m_{k-1}, \ldots, M_4, m_3, M_2, m_1\}$. We show that σ gives minimum angle resolution.
- Let $\alpha_{ij} = \frac{M_i+m_j}{2}$. Angles $\frac{mid+m_k}{2}, \alpha_{(i-1)i}, \frac{M_k+mid}{2}, \alpha_{i(i-1)}$ are in σ.
- Relations among α_{ij}:

$$\alpha_{12} > \alpha_{32} < \alpha_{34} > \cdots > \alpha_{j(j-1)} < \cdots > \alpha_{k(k-1)} < \frac{M_k+mid}{2} > \frac{mid+m_k}{2} < \alpha_{(k-1)k} > \cdots > \alpha_{43} < \alpha_{23} > \alpha_{21} < \alpha_{12}.$$

Baloon Layout. Algorithm by Lin & Yen.

- Recall $\alpha_{ij} = \frac{M_i + m_j}{2}$
- Relations among α_{ij}:

$\alpha_{12} > \alpha_{32} < \alpha_{34} > \cdots > \alpha_{j(j-1)} < \cdots > \alpha_{k(k-1)} < \frac{M_k + mid}{2} > \frac{mid + m_k}{2} < \alpha_{(k-1)k} > \cdots > \alpha_{43} < \alpha_{23} > \alpha_{21} < \alpha_{12}$.

- Smallest angle in σ is either: $\frac{mid + m_k}{2}$ or $\alpha_{j(j-1)}$, while the size of the biggest angle is either $\frac{M_k + mid}{2}$ or $\alpha_{(l-1)l}$, $j, l \in \{2, \ldots, k\}$.

Baloon Layout. Algorithm by Lin & Yen.

- Recall $\alpha_{ij} = \frac{M_i + m_j}{2}$
- Relations among α_{ij}:

$\alpha_{12} > \alpha_{32} < \alpha_{34} > \cdots > \alpha_{j(j-1)} < \cdots > \alpha_{k(k-1)} < \frac{M_k + mid}{2} > \frac{mid + m_k}{2} < \alpha_{(k-1)k} > \cdots > \alpha_{43} < \alpha_{23} > \alpha_{21} < \alpha_{12}$.

- Smallest angle in σ is either: $\frac{mid + m_k}{2}$ or $\alpha_{j(j-1)}$, while the size of the biggest angle is either $\frac{M_k + mid}{2}$ or $\alpha_{(l-1)l}$, $j, l \in \{2, \ldots, k\}$.
- Assume the min angle of σ is $\alpha_{i,i-1} = \frac{M_i + m_{i-1}}{2}$.

Baloon Layout. Algorithm by Lin & Yen.

- Recall $\alpha_{ij} = \frac{M_i + m_j}{2}$
- Relations among α_{ij}:

$\alpha_{12} > \alpha_{32} < \alpha_{34} > \cdots > \alpha_{j(j-1)} < \cdots > \alpha_{k(k-1)} < \frac{M_k + mid}{2} > \frac{mid + m_k}{2} < \alpha_{(k-1)k} > \cdots > \alpha_{43} < \alpha_{23} > \alpha_{21} < \alpha_{12}$.

- Smallest angle in σ is either: $\frac{mid + m_k}{2}$ or $\alpha_{j(j-1)}$, while the size of the biggest angle is either $\frac{M_k + mid}{2}$ or $\alpha_{(l-1)l}$, $j, l \in \{2, \ldots, k\}$.
- Assume the min angle of σ is $\alpha_{i,i-1} = \frac{M_i + m_{i-1}}{2}$.
- Let δ be a permutation with maximum angle resolution

Baloon Layout. Algorithm by Lin & Yen.

- Recall $\alpha_{ij} = \frac{M_i + m_j}{2}$
- Relations among α_{ij}:

$\alpha_{12} > \alpha_{32} < \alpha_{34} > \cdots > \alpha_{j(j-1)} < \cdots > \alpha_{k(k-1)} < \frac{M_k + mid}{2} > \frac{mid + m_k}{2} < \alpha_{(k-1)k} > \cdots > \alpha_{43} < \alpha_{23} > \alpha_{21} < \alpha_{12}$.

- Smallest angle in σ is either: $\frac{mid + m_k}{2}$ or $\alpha_{j(j-1)}$, while the size of the biggest angle is either $\frac{M_k + mid}{2}$ or $\alpha_{(l-1)l}$, $j, l \in \{2, \ldots, k\}$.
- Assume the min angle of σ is $\alpha_{i,i-1} = \frac{M_i + m_{i-1}}{2}$.
- Let δ be a permutation with maximum angle resolution
- If M_i and m_{i-1} neighbor in δ then $optAngResl = \alpha_{i,i-1}$ (???)

Baloon Layout. Algorithm by Lin & Yen.

- Recall $\alpha_{ij} = \frac{M_i + m_j}{2}$
- Relations among α_{ij}:

$\alpha_{12} > \alpha_{32} < \alpha_{34} > \cdots > \alpha_{j(j-1)} < \cdots > \alpha_{k(k-1)} < \frac{M_k + mid}{2} > \frac{mid + m_k}{2} < \alpha_{(k-1)k} > \cdots > \alpha_{43} < \alpha_{23} > \alpha_{21} < \alpha_{12}$.

- Smallest angle in σ is either: $\frac{mid + m_k}{2}$ or $\alpha_{j(j-1)}$, while the size of the biggest angle is either $\frac{M_k + mid}{2}$ or $\alpha_{(l-1)l}$, $j, l \in \{2, \ldots, k\}$.
- Assume the min angle of σ is $\alpha_{i,i-1} = \frac{M_i + m_{i-1}}{2}$.
- Let δ be a permutation with maximum angle resolution
- If M_i and m_{i-1} neighbor in δ then $optAngResl = \alpha_{i,i-1}$ (???)
- If they do not, let x, y be the neighbors of m_{i-1} in δ, then:

$$\underbrace{m_1 < \ldots}_{i-2} < m_{i-1} < \ldots < M_i < \underbrace{\ldots < x < \ldots < y < M_1}_{i-1}$$

Baloon Layout. Algorithm by Lin & Yen.

- Thus, M_i and m_{i-1} neighbor in δ and therefore $AngRes_\sigma = AngRes_\delta$, i.e. σ maximizes the size of the smallest angle.

Baloon Layout. Algorithm by Lin & Yen.

- Thus, M_i and m_{i-1} neighbor in δ and therefore $AngRes_\sigma = AngRes_\delta$, i.e. σ maximizes the size of the smallest angle.
- Similarly, we can show that σ minimizes the lagest angle

Baloon Layout. Algorithm by Lin & Yen.

- Thus, M_i and m_{i-1} neighbor in δ and therefore $AngRes_\sigma = AngRes_\delta$, i.e. σ maximizes the size of the smallest angle.

- Similarly, we can show that σ minimizes the lagest angle

- Recall that: $AspRatio_\sigma = \frac{\max_{1 \leq i \leq n}\{\frac{\theta_{\sigma_i} + \theta_{\sigma_{i+1}}}{2}\}}{\min_{1 \leq i \leq n}\{\frac{\theta_{\sigma_i} + \theta_{\sigma_{i+1}}}{2}\}}$

Baloon Layout. Algorithm by Lin & Yen.

- Thus, M_i and m_{i-1} neighbor in δ and therefore $AngRes_\sigma = AngRes_\delta$, i.e. σ maximizes the size of the smallest angle.
- Similarly, we can show that σ minimizes the lagest angle
- Recall that: $AspRatio_\sigma = \frac{\max_{1 \le i \le n}\{\frac{\theta_{\sigma_i} + \theta_{\sigma_{i+1}}}{2}\}}{\min_{1 \le i \le n}\{\frac{\theta_{\sigma_i} + \theta_{\sigma_{i+1}}}{2}\}}$
- The radii and therefore the angles are independent on each level

Baloon Layout. Algorithm by Lin & Yen.

- Thus, M_i and m_{i-1} neighbor in δ and therefore $AngRes_\sigma = AngRes_\delta$, i.e. σ maximizes the size of the smallest angle.
- Similarly, we can show that σ minimizes the lagest angle
- Recall that: $AspRatio_\sigma = \frac{\max_{1\le i\le n}\{\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}\}}{\min_{1\le i\le n}\{\frac{\theta_{\sigma_i}+\theta_{\sigma_{i+1}}}{2}\}}$
- The radii and therefore the angles are independent on each level
- Therefore, if we apply σ at each level, we obtain an optimal aspect ratio. □