Visualization of Graphs

Lecture 1b:
Drawing Trees and Series-Parallel Graphs

Part I:
Layered Drawings

Alexander Wolff
(Rooted) Trees
(Rooted) Trees

**Leaf**: Vertex of degree 1
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated root
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated root

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated root

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root
(Rooted) Trees

**Leaf**: Vertex of degree 1

**Rooted tree**: tree with designated **root**

**Ancestor**: Vertex on path to root

**Parent**: Neighbor on path to root

**Successor**: Vertex on path away from root

**Child**: Neighbor not on path to root

**Depth**: Length of path to root
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated root

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

\[
\text{parent}(u) \quad \text{children}(u) \\
\text{successors}(u) \\
\text{ancestors}(u) \\
\text{depth}(u) = 3 \\
\text{height}(G) = 5
\]
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)
(Rooted) Trees

**Leaf**: Vertex of degree 1

**Rooted tree**: tree with designated root

**Ancestor**: Vertex on path to root

**Parent**: Neighbor on path to root

**Successor**: Vertex on path away from root

**Child**: Neighbor not on path to root

**Depth**: Length of path to root

**Height**: Maximum depth of a leaf

**Binary Tree**: At most two children per vertex (left / right child)
(Rooted) Trees

**Leaf:** Vertex of degree 1  
**Rooted tree:** tree with designated **root**  
**Ancestor:** Vertex on path to root  
**Parent:** Neighbor on path to root  
**Successor:** Vertex on path away from root  
**Child:** Neighbor not on path to root

**Depth:** Length of path to root  
**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:
(Rooted) Trees

**Leaf**: Vertex of degree 1

**Rooted tree**: tree with designated **root**

**Ancestor**: Vertex on path to root

**Parent**: Neighbor on path to root

**Successor**: Vertex on path away from root

**Child**: Neighbor not on path to root

**Depth**: Length of path to root

**Height**: Maximum depth of a leaf

**Binary Tree**: At most two children per vertex (left / right child)

3 traversals:

- **preorder**: node – left – right

\[
\text{depth}(u) = 3 \\
\text{height}(G) = 5
\]
(Rooted) Trees

**Leaf**: Vertex of degree 1

**Rooted tree**: tree with designated root

**Ancestor**: Vertex on path to root

**Parent**: Neighbor on path to root

**Successor**: Vertex on path away from root

**Child**: Neighbor not on path to root

**Depth**: Length of path to root

**Height**: Maximum depth of a leaf

**Binary Tree**: At most two children per vertex (left / right child)

3 traversals:

- **preorder**: node – left – right

![Diagram of a rooted tree with labeled parts and traversal paths](image)
(Rooted) Trees

**Leaf**: Vertex of degree 1

**Rooted tree**: tree with designated **root**

**Ancestor**: Vertex on path to root

**Parent**: Neighbor on path to root

**Successor**: Vertex on path away from root

**Child**: Neighbor not on path to root

**Depth**: Length of path to root

**Height**: Maximum depth of a leaf

**Binary Tree**: At most two children per vertex (left / right child)

3 traversals:

1. **preorder**
   
   node – left – right

   
   
   depth\( (u) = 3 \)

   height\( (G) = 5 \)
(Rooted) Trees

**Leaf**: Vertex of degree 1

**Rooted tree**: tree with designated root

**Ancestor**: Vertex on path to root

**Parent**: Neighbor on path to root

**Successor**: Vertex on path away from root

**Child**: Neighbor not on path to root

**Depth**: Length of path to root

**Height**: Maximum depth of a leaf

**Binary Tree**: At most two children per vertex (left / right child)

3 traversals:

- **preorder**: node – left – right

![Diagram of a rooted tree with labels for parent, children, ancestors, successors, and traversal paths.](attachment:image.png)

- `depth(u) = 3`
- `height(G) = 5`
(Rooted) Trees

Leaf: Vertex of degree 1
Rooted tree: tree with designated root
Ancestor: Vertex on path to root
Parent: Neighbor on path to root
Successor: Vertex on path away from root
Child: Neighbor not on path to root
Depth: Length of path to root
Height: Maximum depth of a leaf
Binary Tree: At most two children per vertex (left / right child)

3 traversals:

preorder

node – left – right

\[ \text{depth}(u) = 3 \]
\[ \text{height}(G) = 5 \]
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder**
  - node – left – right

\[
\text{depth}(u) = 3 \\
\text{height}(G) = 5
\]
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated root

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- preorder: node – left – right

```
         u
       /   \
      /     \        depth(u) = 3
     /       \       height(G) = 5
   /         \      root
 child(u)   succ(u)  ancestors(u)
```

children(u), parent(u), successors(u), ancestors(u)
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated root

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder**
  
  node – left – right
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder:** node – left – right
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- preorder: node – left – right

![Diagram of a tree with labeled vertices and paths]

- `depth(u) = 3`
- `height(G) = 5`
(Rooted) Trees

Leaf: Vertex of degree 1
Rooted tree: tree with designated root
Ancestor: Vertex on path to root
Parent: Neighbor on path to root
Successor: Vertex on path away from root
Child: Neighbor not on path to root
Depth: Length of path to root
Height: Maximum depth of a leaf
Binary Tree: At most two children per vertex (left / right child)

3 traversals:

preorder: node – left – right
inorder: left – node – right
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder**: node – left – right
- **inorder**: left – node – right
- **3 traversals:**
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder**
  
  node – left – right

- **inorder**
  
  left – node – right

- **depth**

  depth(u) = 3

- **height**

  height(G) = 5
(Rooted) Trees

**Leaf**: Vertex of degree 1

**Rooted tree**: tree with designated root

**Ancestor**: Vertex on path to root

**Parent**: Neighbor on path to root

**Successor**: Vertex on path away from root

**Child**: Neighbor not on path to root

**Depth**: Length of path to root

**Height**: Maximum depth of a leaf

**Binary Tree**: At most two children per vertex (left / right child)

3 traversals:

- **preorder**: node – left – right
- **inorder**: left – node – right
- **successors**: vertex on path away from root
- **ancestors**: vertex on path to root
- **parent**: neighbor on path to root
- **children**: neighbor not on path to root
- **depth**: length of path to root
- **height**: maximum depth of a leaf

$\text{depth}(u) = 3$

$\text{height}(G) = 5$
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated root

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder:** node – left – right
- **inorder:** left – node – right
- **successors(u)**
- **parent(u)**
- **ancestors(u)**
- **children(u)**

\[ \text{depth}(u) = 3 \]
\[ \text{height}(G') = 5 \]
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder**
  - node – left – right

- **inorder**
  - left – node – right

\[ \text{depth}(u) = 3 \]
\[ \text{height}(G) = 5 \]
(Rooted) Trees

Leaf: Vertex of degree 1
Rooted tree: tree with designated root
Ancestor: Vertex on path to root
Parent: Neighbor on path to root
Successor: Vertex on path away from root
Child: Neighbor not on path to root
Depth: Length of path to root
Height: Maximum depth of a leaf
Binary Tree: At most two children per vertex (left / right child)

3 traversals:

- preorder: node – left – right
- inorder: left – node – right

\[ \text{depth}(u) = 3 \]
\[ \text{height}(G) = 5 \]
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder:** node – left – right
- **inorder:** left – node – right
- **successors(u)**
(Rooted) Trees

Leaf: Vertex of degree 1

Rooted tree: tree with designated root

Ancestor: Vertex on path to root

Parent: Neighbor on path to root

Successor: Vertex on path away from root

Child: Neighbor not on path to root

Depth: Length of path to root

Height: Maximum depth of a leaf

Binary Tree: At most two children per vertex (left / right child)

3 traversals:

- preorder: node – left – right
- inorder: left – node – right

```
parent(u)  
children(u)  
ancestors(u)  
successors(u)  

depth(u) = 3  
height(G) = 5
```
(Rooted) Trees

**Leaf**: Vertex of degree 1

**Rooted tree**: tree with designated root

**Ancestor**: Vertex on path to root

**Parent**: Neighbor on path to root

**Successor**: Vertex on path away from root

**Child**: Neighbor not on path to root

**Depth**: Length of path to root

**Height**: Maximum depth of a leaf

**Binary Tree**: At most two children per vertex (left / right child)

3 traversals:

- **preorder**: node – left – right
- **inorder**: left – node – right
- **depth**(u) = 3
- height(G) = 5
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated root

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder:** node – left – right
- **inorder:** left – node – right
- **postorder:** left – right – node

\[
\begin{align*}
\text{parent}(u) & \quad \text{children}(u) \\
\text{successors}(u) & \quad \text{ancestors}(u) \\
\text{depth}(u) & = 3 \\
\text{height}(G) & = 5
\end{align*}
\]
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated root

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder**
  - node – left – right

- **inorder**
  - left – node – right

- **postorder**
  - left – right – node
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder**: node – left – right
- **inorder**: left – node – right
- **postorder**: left – right – node
(Rooted) Trees

Leaf: Vertex of degree 1
Rooted tree: tree with designated root
Ancestor: Vertex on path to root
Parent: Neighbor on path to root
Successor: Vertex on path away from root
Child: Neighbor not on path to root
Depth: Length of path to root
Height: Maximum depth of a leaf
Binary Tree: At most two children per vertex (left / right child)

3 traversals:

- preorder: node – left – right
- inorder: left – node – right
- postorder: left – right – node
(Rooted) Trees

**Leaf**: Vertex of degree 1

**Rooted tree**: tree with designated **root**

**Ancestor**: Vertex on path to root

**Parent**: Neighbor on path to root

**Successor**: Vertex on path away from root

**Child**: Neighbor not on path to root

**Depth**: Length of path to root

**Height**: Maximum depth of a leaf

**Binary Tree**: At most two children per vertex (left / right child)

3 traversals:

- **preorder**: node – left – right
- **inorder**: left – node – right
- **postorder**: left – right – node

\[
\begin{align*}
\text{depth}(u) &= 3 \\
\text{height}(G) &= 5
\end{align*}
\]
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated root

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **Preorder:** node – left – right
- **Inorder:** left – node – right
- **Postorder:** left – right – node

\[ \text{parent}(u), \text{children}(u), \text{successors}(u), \text{ancestors}(u), \text{depth}(u) = 3, \text{height}(G') = 5 \]
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder**
  - node – left – right

- **inorder**
  - left – node – right

- **postorder**
  - left – right – node
(Rooted) Trees

**Leaf**: Vertex of degree 1

**Rooted tree**: tree with designated **root**

**Ancestor**: Vertex on path to root

**Parent**: Neighbor on path to root

**Successor**: Vertex on path away from root

**Child**: Neighbor not on path to root

**Depth**: Length of path to root

**Height**: Maximum depth of a leaf

**Binary Tree**: At most two children per vertex (left / right child)

3 traversals:

- **preorder**: node – left – right
- **inorder**: left – node – right
- **postorder**: left – right – node

\[
\begin{align*}
\text{parent}(u) & \quad \text{successors}(u) \\
\text{root} & \quad \text{children}(u) \\
\text{depth}(u) & = 3 \\
\text{height}(G) & = 5 \\
\text{ancestors}(u) & \\
\end{align*}
\]
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated root

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder**
  - node – left – right

- **inorder**
  - left – node – right

- **postorder**
  - left – right – node
(Rooted) Trees

**Leaf:** Vertex of degree 1

**Rooted tree:** tree with designated **root**

**Ancestor:** Vertex on path to root

**Parent:** Neighbor on path to root

**Successor:** Vertex on path away from root

**Child:** Neighbor not on path to root

**Depth:** Length of path to root

**Height:** Maximum depth of a leaf

**Binary Tree:** At most two children per vertex (left / right child)

3 traversals:

- **preorder**
  - node – left – right

- **inorder**
  - left – node – right

- **postorder**
  - left – right – node

\[ \text{depth}(u) = 3 \]

\[ \text{height}(G) = 5 \]
First Grid Layout of Binary Trees

1. Choose $y$-coordinates:
First Grid Layout of Binary Trees

1. Choose $y$-coordinates:

2. Choose $x$-coordinates:
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:
First Grid Layout of Binary Trees

1. Choose $y$-coordinates:  $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

   - **preorder**
   - **inorder**
   - **postorder**
First Grid Layout of Binary Trees

1. Choose y-coordinates: \( y(u) = \text{depth}(u) \)

2. Choose x-coordinates:

- **preorder**
- **inorder**
- **postorder**
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

- **preorder**
- **inorder**
- **postorder**
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

- preorder
- inorder
- postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:
First Grid Layout of Binary Trees

1. Choose \( y \)-coordinates: \( y(u) = \text{depth}(u) \)

2. Choose \( x \)-coordinates:

preorder

inorder

postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

- **preorder**
- **inorder**
- **postorder**
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

   - preorder
   - inorder
   - postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

   - preorder
   - inorder
   - postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

- preorder
- inorder
- postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:
   - preorder
   - inorder
   - postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:
First Grid Layout of Binary Trees

1. Choose \( y \)-coordinates: \( y(u) = \text{depth}(u) \)

2. Choose \( x \)-coordinates:

   - preorder
   - inorder
   - postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

   - preorder
   - inorder
   - postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:
First Grid Layout of Binary Trees

1. Choose \( y \)-coordinates: \( y(u) = \text{depth}(u) \)

2. Choose \( x \)-coordinates:

   - **preorder**
   - **inorder**
   - **postorder**
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

- preorder
- inorder
- postorder
First Grid Layout of Binary Trees

1. Choose \( y \)-coordinates: \( y(u) = \text{depth}(u) \)

2. Choose \( x \)-coordinates:

- **Preorder**
- **Inorder**
- **Postorder**
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

- preorder
- inorder
- postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

- preorder
- inorder
- postorder
1. Choose y-coordinates: \( y(u) = \text{depth}(u) \)

2. Choose x-coordinates:
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

preorder   inorder   postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

   - preorder
   - inorder
   - postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

   - preorder
   - inorder
   - postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

- **preorder**
- **inorder**
- **postorder**
First Grid Layout of Binary Trees

1. Choose \( y \)-coordinates: \( y(u) = \text{depth}(u) \)

2. Choose \( x \)-coordinates:

   - **Preorder**
   - **Inorder**
   - **Postorder**
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: \[ y(u) = \text{depth}(u) \]

2. Choose $x$-coordinates:

preorder

inorder

postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

- **preorder**
- **inorder**
- **postorder**
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:

- preorder
- inorder
- postorder
First Grid Layout of Binary Trees

1. Choose $y$-coordinates: $y(u) = \text{depth}(u)$

2. Choose $x$-coordinates:
First Grid Layout of Binary Trees

1. Choose y-coordinates: \( y(u) = \text{depth}(u) \)

2. Choose x-coordinates:

- Preorder
- Inorder
- Postorder
Layered Drawings – Applications

Decision tree for outcome prediction after traumatic brain injury

Source: Nature Reviews Neurology
Layered Drawings – Applications

Aloisius Gaultier 1821

Family tree of LOTR elves and half-elves
What are properties of the layout?
Layered Drawings – Drawing Style

- What are properties of the layout?
- What are the drawing conventions?
Layered Drawings – Drawing Style

- What are properties of the layout?
- What are the drawing conventions?
- What are aesthetics to optimize?
Layered Drawings – Drawing Style

- What are properties of the layout?
- What are the drawing conventions?
- What are aesthetics to optimize?
Layered Drawings – Drawing Style

- What are properties of the layout?
- What are the drawing conventions?
- What are aesthetics to optimize?

**Drawing conventions**
- Vertices lie on layers and have integer coordinates
Layered Drawings – Drawing Style

- What are properties of the layout?
- What are the drawing conventions?
- What are aesthetics to optimize?

**Drawing conventions**
- Vertices lie on layers and have integer coordinates
- Parent centered above children
Layered Drawings – Drawing Style

What are properties of the layout?
What are the drawing conventions?
What are aesthetics to optimize?

Drawing conventions
- Vertices lie on layers and have integer coordinates
- Parent centered above children
- Edges are straight-line segments
Layered Drawings – Drawing Style

What are properties of the layout?
What are the drawing conventions?
What are aesthetics to optimize?

Drawing conventions
- Vertices lie on layers and have integer coordinates
- Parent centered above children
- Edges are straight-line segments
- Isomorphic subtrees have identical drawings
Layered Drawings – Drawing Style

- What are properties of the layout?
- What are the drawing conventions?
- What are aesthetics to optimize?

**Drawing conventions**
- Vertices lie on layers and have integer coordinates
- Parent centered above children
- Edges are straight-line segments
- Isomorphic subtrees have identical drawings
Layered Drawings – Drawing Style

What are properties of the layout?
What are the drawing conventions?
What are aesthetics to optimize?

Drawing conventions
- Vertices lie on layers and have integer coordinates
- Parent centered above children
- Edges are straight-line segments
- Isomorphic subtrees have identical drawings
Layered Drawings – Drawing Style

- What are properties of the layout?
- What are the drawing conventions?
- What are aesthetics to optimize?

**Drawing conventions**
- Vertices lie on layers and have integer coordinates
- Parent centered above children
- Edges are straight-line segments
- Isomorphic subtrees have identical drawings

**Drawing aesthetics**
Layered Drawings – Drawing Style

■ What are properties of the layout?
■ What are the drawing conventions?
■ What are aesthetics to optimize?

Drawing conventions
■ Vertices lie on layers and have integer coordinates
■ Parent centered above children
■ Edges are straight-line segments
■ Isomorphic subtrees have identical drawings

Drawing aesthetics
■ Area
Layered Drawings – Drawing Style

- What are properties of the layout?
- What are the drawing conventions?
- What are aesthetics to optimize?

**Drawing conventions**
- Vertices lie on layers and have integer coordinates
- Parent centered above children
- Edges are straight-line segments
- Isomorphic subtrees have identical drawings

**Drawing aesthetics**
- Area
- Symmetries
Layered Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** A layered drawing of $T$
Layered Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** A layered drawing of $T$

**Base case:**

**Divide:**

**Conquer:**
Layered Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** A layered drawing of $T$

**Base case:** A single vertex

**Divide:**

**Conquer:**
Layered Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** A layered drawing of $T$

**Base case:** A single vertex

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

**Conquer:**
Layered Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** A layered drawing of $T$

**Base case:** A single vertex

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

**Conquer:**
Layered Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** A layered drawing of $T$

**Base case:** A single vertex

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

**Conquer:**
Layered Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** A layered drawing of $T$

**Base case:** A single vertex

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

**Conquer:**
Layered Drawings – Algorithm

**Input:** A binary tree $T$
**Output:** A layered drawing of $T$

**Base case:** A single vertex

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

**Conquer:**
Layered Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** A layered drawing of $T$

**Base case:** A single vertex

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

**Conquer:**

Some agreed distance
Layered Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** A layered drawing of $T$

**Base case:** A single vertex

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

**Conquer:**

- some agreed distance
- parent centered wrt to children
Layered Drawings – Algorithm

**Input:** A binary tree $T$
**Output:** A layered drawing of $T$

**Base case:** A single vertex

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

**Conquer:**

- some agreed distance
- parent centered wrt to children

sometimes 3 apart for grid drawing!
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child

Phase 2 – preorder traversal:
- Compute x- and y-coordinates

Graph showing vertex traversal and coordinate markers.
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex \( u \) (below \( v \)) store left and right contour of subtree \( T(u) \)
- Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute \( x \)- and \( y \)-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex \( u \) (below \( v \)) store left and right contour of subtree \( T(u) \)
- Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
- Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
- Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
- Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex \( u \) (below \( v \)) store left and right contour of subtree \( T(u) \)
- Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute \( x \)- and \( y \)-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of sub-tree $T(u)$
- Contour is linked list of vertex coordinates/offsets

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex \( u \) (below \( v \)) store left and right contour of subtree \( T(u) \)
- Contour is linked list of vertex coordinates/offsets
- Find \( d_v = \text{min. horiz. distance between } v_l \text{ and } v_r \)

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of sub-tree $T(u)$
- Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
- Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Layered Drawings – Algorithm Details

**Phase 1 – postorder traversal:**
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
- Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

**Phase 2 – preorder traversal:**
- Compute $x$- and $y$-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
- Contour is linked list of vertex coordinates/offsets
- Find $d_v = \min$ horiz. distance between $v_l$ and $v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:

■ For each vertex compute horizontal displacement of left and right child

■ At vertex $u$ (below $v$) store left and right contour of sub-tree $T(u)$

■ Contour is linked list of vertex coordinates/offsets

■ Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:

■ Compute $x$- and $y$-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- \( x\text{-offset}(v_l) = -\left\lceil \frac{d_v}{2} \right\rceil, x\text{-offset}(v_r) = \left\lceil \frac{d_v}{2} \right\rceil \)
- At vertex \( u \) (below \( v \)) store left and right contour of sub-tree \( T(u) \)
- Contour is linked list of vertex coordinates/offsets
- Find \( d_v = \text{min. horiz. distance between } v_l \text{ and } v_r \)

Phase 2 – preorder traversal:
- Compute \( x \)- and \( y \)-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- \( x\text{-offset}(v_l) = -\lceil \frac{d_v}{2} \rceil, \ x\text{-offset}(v_r) = \lceil \frac{d_v}{2} \rceil \)
- At vertex \( u \) (below \( v \)) store left and right contour of sub-tree \( T(u) \)
- Contour is linked list of vertex coordinates/offsets
- Find \( d_v = \text{min. horiz. distance between } v_l \text{ and } v_r \)

Phase 2 – preorder traversal:
- Compute x- and y-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- $x$-offset($v_l$) = $-\lceil \frac{d_v}{2} \rceil$, $x$-offset($v_r$) = $\lceil \frac{d_v}{2} \rceil$
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
- Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- \( x\text{-offset}(v_l) = -\left\lceil \frac{d_v}{2} \right\rceil, x\text{-offset}(v_r) = \left\lceil \frac{d_v}{2} \right\rceil \)
- At vertex \( u \) (below \( v \)) store left and right contour of sub-tree \( T(u) \)
- Contour is linked list of vertex coordinates/offsets
- Find \( d_v = \text{min. horiz. distance between } v_l \text{ and } v_r \)

Phase 2 – preorder traversal:
- Compute \( x- \) and \( y \)-coordinates

Runtime?
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- $x$-offset($v_l$) = $-\lceil \frac{d_v}{2} \rceil$, $x$-offset($v_r$) = $\lceil \frac{d_v}{2} \rceil$
- At vertex $u$ (below $v$) store left and right contour of sub-tree $T(u)$
- Contour is linked list of vertex coordinates/offsets
- Find $d_v$ = min. horiz. distance between $v_l$ and $v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates

Runtime?
- How often do we have to walk along a contour?
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- $x\text{-offset}(v_l) = -\lceil \frac{d_v}{2} \rceil$, $x\text{-offset}(v_r) = \lceil \frac{d_v}{2} \rceil$
- At vertex $u$ (below $v$) store left and right contour of sub-tree $T(u)$
- Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates

Runtime?
- How often do we have to walk along a contour?
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:

■ For each vertex compute horizontal displacement of left and right child

■ \( x\text{-offset}(v_l) = -\lfloor \frac{d_v}{2} \rfloor, \ x\text{-offset}(v_r) = \lceil \frac{d_v}{2} \rceil \)

■ At vertex \( u \) (below \( v \)) store left and right contour of sub-tree \( T(u) \)

■ Contour is linked list of vertex coordinates/offsets

■ Find \( d_v = \text{min. horiz. distance between } v_l \text{ and } v_r \)

Phase 2 – preorder traversal:

■ Compute x- and y-coordinates

Runtime?

■ How often do we have to walk along a contour?
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- $x$-offset($v_l$) = $-\left\lceil \frac{d_v}{2} \right\rceil$, $x$-offset($v_r$) = $\left\lceil \frac{d_v}{2} \right\rceil$
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
- Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates

Runtime?
- How often do we have to walk along a contour?
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- \( x\text{-offset}(v_l) = -\left\lfloor \frac{d_v}{2} \right\rfloor, x\text{-offset}(v_r) = \left\lceil \frac{d_v}{2} \right\rceil \)
- At vertex \( u \) (below \( v \)) store left and right contour of sub-tree \( T(u) \)
- Contour is linked list of vertex coordinates/offsets
- Find \( d_v = \) min. horiz. distance between \( v_l \) and \( v_r \)

Phase 2 – preorder traversal:
- Compute x- and y-coordinates

Runtime?
- How often do we have to walk along a contour?
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- $\text{x-offset}(v_l) = -\lceil \frac{d_v}{2} \rceil$, $\text{x-offset}(v_r) = \lceil \frac{d_v}{2} \rceil$
- At vertex $u$ (below $v$) store left and right contour of subtree $T(u)$
- Contour is linked list of vertex coordinates/offsets
- Find $d_v = \text{min. horiz. distance between } v_l \text{ and } v_r$

Phase 2 – preorder traversal:
- Compute x- and y-coordinates

Runtime?
- How often do we have to walk along a contour?
Layered Drawings – Algorithm Details

**Phase 1 – postorder traversal:**
- For each vertex compute horizontal displacement of left and right child
- \( x\text{-offset}(v_l) = -\left\lfloor \frac{d_v}{2} \right\rfloor, \ x\text{-offset}(v_r) = \left\lceil \frac{d_v}{2} \right\rceil \)
- At vertex \( u \) (below \( v \)) store left and right contour of sub-tree \( T(u) \)
- Contour is linked list of vertex coordinates/offsets
- Find \( d_v = \text{min. horiz. distance between } v_l \text{ and } v_r \)

**Phase 2 – preorder traversal:**
- Compute \( x \)- and \( y \)-coordinates

**Runtime?**
- How often do we have to walk along a contour?
Layered Drawings – Algorithm Details

Phase 1 – postorder traversal:
- For each vertex compute horizontal displacement of left and right child
- $x$-offset($v_l$) = $-\left\lceil \frac{d_v}{2} \right\rceil$, $x$-offset($v_r$) = $\left\lceil \frac{d_v}{2} \right\rceil$
- At vertex $u$ (below $v$) store left and right contour of sub-tree $T(u)$
- Contour is linked list of vertex coordinates/offsets
- Find $d_v$ = min. horiz. distance between $v_l$ and $v_r$

Phase 2 – preorder traversal:
- Compute $x$- and $y$-coordinates

Runtime?
- How often do we have to walk along a contour?
- Less than $n = \#$ vertices times!
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: $y$-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$—but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- Horizontal and vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection
Layered Drawings – Result

Theorem. [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\operatorname{depth}(v)$
- Horizontal and vertical distances are at least 1
- Each vertex is centred with respect to its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]

Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]

Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:
- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
Layered Drawings – Result

**Theorem.** [Reingold & Tilford ’81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ -
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]

Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
**Theorem.** [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: $y$-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
**Theorem.** [Reingold & Tilford '81]

Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]

Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: $y$-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!

**NP-hard**
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]

Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation

**NP-hard**
Layered Drawings – Result

Theorem. [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation

NP-hard
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation

**NP-hard**
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]

Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection

**NP-hard**
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]

Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection
Layered Drawings – Result

Theorem. [Reingold & Tilford ’81]
Let $T$ be a rooted binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection

NP-hard
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]

Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection

\[
\text{NP-hard}
\]
Theorem. [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection

**NP-hard**
Layered Drawings – Result

Theorem. [Reingold & Tilford ’81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!

- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection

NP-hard
Layered Drawings – Result

Theorem. [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection

NP-hard
Layered Drawings – Result

Theorem. [Reingold & Tilford ’81]
Let $T$ be a rooted binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection

NP-hard
Layered Drawings – Result

Theorem. [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection

NP-hard
Layered Drawings – Result

Theorem. [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: $y$-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection

NP-hard
Layered Drawings – Result

Theorem. [Reingold & Tilford '81]
Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: y-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection

NP-hard
Layered Drawings – Result

**Theorem.** [Reingold & Tilford '81]

Let $T$ be a binary tree with $n$ vertices. We can construct a drawing $\Gamma$ of $T$ in $O(n)$ time, such that:

- $\Gamma$ is planar, straight-line and strictly downward
- $\Gamma$ is layered: $y$-coordinate of vertex $v$ is $-\text{depth}(v)$
- Horizontal and Vertical distances are at least 1
- Each vertex is centred w.r.t. its children
- Area of $\Gamma$ is in $O(n^2)$ – but not optimal!
- Simply isomorphic subtrees have congruent drawings, up to translation
- Axially isomorphic subtrees have congruent drawings, up to translation and reflection
Visualization of Graphs

Lecture 1b:
Drawing Trees and Series-Parallel Graphs

Part II:
HV-Drawings
HV-Drawings – Drawing Style

Applications

- Cons cell diagram in LISP
HV-Drawings – Drawing Style

Applications

- Cons cell diagram in LISP
- Cons (constructs) are memory objects that hold two values or pointers to values
HV-Drawings – Drawing Style

Applications

- Cons cell diagram in LISP
- Cons (constructs) are memory objects that hold two values or pointers to values

Source: after gajon.org/trees-linked-lists-common-lisp/
HV-Drawings – Drawing Style

Applications

- Cons cell diagram in LISP
- Cons (constructs) are memory objects that hold two values or pointers to values

Drawing conventions

- Children are vertically or horizontally aligned with their parent
- The bounding boxes of the subtrees of the children are disjoint
- Edges are strictly down- or rightwards

Drawing aesthetics

- Height, width, area

Source: after gajon.org/trees-linked-lists-common-lisp/
HV-Drawings – Drawing Style

Applications

- Cons cell diagram in LISP
- **Cons** (constructs) are memory objects that hold two values or pointers to values

```
1 -- 3 -- 11 /
|      |     |
5 -- 10 --- 12 /
|     |      |
1 -- 12 /

4 -- 6 -- 7 -- 8 /
```

Drawing conventions
- Children are vertically or horizontally aligned with their parent

Drawing aesthetics
- Height, width, area

Source: after gajon.org/trees-linked-lists-common-lisp/
HV-Drawings – Drawing Style

Applications

- Cons cell diagram in LISP
- Cons (constructs) are memory objects that hold two values or pointers to values

Drawing conventions

- Children are vertically or horizontally aligned with their parent
- The bounding boxes of the subtrees of the children are disjoint

Drawing aesthetics

Source: after gajon.org/trees-linked-lists-common-lisp/
HV-Drawings – Drawing Style

Applications
- Cons cell diagram in LISP
- *Cons* (constructs) are memory objects that hold two values or pointers to values

![Diagram of a cons cell](Source: after gajon.org/trees-linked-lists-common-lisp/)

Drawing conventions
- Children are vertically or horizontally aligned with their parent
- The bounding boxes of the subtrees of the children are disjoint

Drawing aesthetics
- Height, width, area
**HV-Drawings – Drawing Style**

**Applications**
- Cons cell diagram in LISP
- *Cons* (constructs) are memory objects that hold two values or pointers to values

```
   1---2---3
     /    /
    5---10---11
   1---2---3
     /    /
    9---12
```

**Drawing conventions**
- Children are vertically or horizontally aligned with their parent
- The bounding boxes of the subtrees of the children are disjoint

**Drawing aesthetics**
- Height, width, area

*Source:* after gajon.org/trees-linked-lists-common-lisp/
HV-Drawings – Drawing Style

Applications

- Cons cell diagram in LISP
- Cons (constructs) are memory objects that hold two values or pointers to values

```
1  3  /  10  11 /
5  /  9  12 /
6  7  8 /
```

Drawing conventions

- Children are vertically or horizontally aligned with their parent
- The bounding boxes of the subtrees of the children are disjoint

Drawing aesthetics

Source: after gajon.org/trees-linked-lists-common-lisp/
HV-Drawings – Drawing Style

Applications

- Cons cell diagram in LISP
- Cons (constructs) are memory objects that hold two values or pointers to values

Drawing conventions

- Children are vertically or horizontally aligned with their parent
- The bounding boxes of the subtrees of the children are disjoint
- Edges are strictly down- or rightwards

Drawing aesthetics

Source: after gajon.org/trees-linked-lists-common-lisp/
HV-Drawings – Drawing Style

Applications
- Cons cell diagram in LISP
- *Cons* (constructs) are memory objects that hold two values or pointers to values

`Source: after gajon.org/trees-linked-lists-common-lisp/`

Drawing conventions
- Children are vertically or horizontally aligned with their parent
- The bounding boxes of the subtrees of the children are disjoint
- Edges are strictly down- or rightwards

Drawing aesthetics
- Height, width, area
HV-Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** An HV-drawing of $T$
HV-Drawings – Algorithm

Input: A binary tree $T$
Output: An HV-drawing of $T$

Base case:  

HV-Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** An HV-drawing of $T$

**Base case:**

**Divide:** Recursively apply the algorithm to draw the left and right subtrees
HV-Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** An HV-drawing of $T$

**Base case:**

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

**Conquer:**
HV-Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** An HV-drawing of $T$

**Base case:** 

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

**Conquer:** horizontal combination
HV-Drawings – Algorithm

**Input:** A binary tree $T$

**Output:** An HV-drawing of $T$

**Base case:**

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

**Conquer:**

- horizontal combination
- vertical combination
HV-Drawings – Right-Heavy HV-Layout

Right-heavy approach
- Always apply horizontal combination
HV-Drawings – Right-Heavy HV-Layout

**Right-heavy approach**

- Always apply horizontal combination
- Place the larger subtree to the right
HV-Drawings – Right-Heavy HV-Layout

Right-heavy approach

- Always apply horizontal combination
- Place the larger subtree to the right

← This can change the embedding!
HV-Drawings – Right-Heavy HV-Layout

**Right-heavy approach**

- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

← This can change the embedding!
HV-Drawings – Right-Heavy HV-Layout

**Right-heavy approach**

- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

← This can change the embedding!
HV-Drawings – Right-Heavy HV-Layout

Right-heavy approach

- Always apply horizontal combination
- Place the larger subtree to the right
  
  Size of subtree := number of vertices

← This can change the embedding!
HV-Drawings – Right-Heavy HV-Layout

**Right-heavy approach**

- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

← This can change the embedding!
HV-Drawings – Right-Heavy HV-Layout

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right
  Size of subtree := number of vertices

← This can change the embedding!
HV-Drawings – Right-Heavy HV-Layout

Right-heavy approach

- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

← This can change the embedding!
Right-heavy approach

- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

← This can change the embedding!
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has

- width at most $n - 1$ and
- height at most $\log n$.

**Right-heavy approach**

- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

← This can change the embedding!
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$.

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

← This can change the embedding!
Lemma. Let \( T \) be a binary tree. The drawing constructed by the right-heavy approach has

- width at most \( n - 1 \) and
- height at most \( \log n \).
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$. 

**Right-heavy approach**
- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

← This can change the embedding!
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has

- width at most $n - 1$ and
- height at most $\log n$.

Right-heavy approach

- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

This can change the embedding!
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$. 

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has

- width at most $n - 1$ and
- height at most $\log n$.
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$. 

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right
  Size of subtree := number of vertices

This can change the embedding!
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$.

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

This can change the embedding!
**Lemma.** Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has

- width at most $n - 1$ and
- height at most $\log n$. 

**Right-heavy approach**

- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

This can change the embedding!
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has
- width at most $n - 1$ and
- height at most $\log n$. 

Right-heavy approach
- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

This can change the embedding!
Lemma. Let $T$ be a binary tree. The drawing constructed by the right-heavy approach has

- width at most $n - 1$ and
- height at most $\log n$.

Right-heavy approach

- Always apply horizontal combination
- Place the larger subtree to the right

Size of subtree := number of vertices

How to implement this in linear time?

This can change the embedding!
Theorem.
Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing (planar, orthogonal, strictly right-/downward)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem.
Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing
  (planar, orthogonal, strictly right-/downward)
- Width is at most $n-1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem.
Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:
\begin{itemize}
  \item $\Gamma$ is an HV-drawing
    (planar, orthogonal, strictly right-/downward)
  \item Width is at most $n - 1$
\end{itemize}
Theorem.
Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing (planar, orthogonal, strictly right-/downward)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem.
Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing (planar, orthogonal, strictly right-/downward)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem.
Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing (planar, orthogonal, strictly right-/downward)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem.

Let \( T \) be a binary tree with \( n \) vertices. The right-heavy algorithm constructs in \( O(n) \) time a drawing \( \Gamma \) of \( T \) s.t.:

- \( \Gamma \) is an HV-drawing
  (planar, orthogonal, strictly right-/downward)
- Width is at most \( n - 1 \)
- Height is at most \( \log n \)
- Area is in \( O(n \log n) \)
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem. Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing (planar, orthogonal, strictly right-/downward)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem. Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing (planar, orthogonal, strictly right-/downward)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
**Theorem.** Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing (planar, orthogonal, strictly right-/downward)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem. Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing (planar, orthogonal, strictly right-/downward)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem.

Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing (planar, orthogonal, strictly right-/downward)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem. Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing (planar, orthogonal, strictly right-/downward)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation
Theorem. Let $T$ be a binary tree with $n$ vertices. The right-heavy algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is an HV-drawing (planar, orthogonal, strictly right-/downward)
- Width is at most $n - 1$
- Height is at most $\log n$
- Area is in $O(n \log n)$
- Simply and axially isomorphic subtrees have congruent drawings up to translation

General rooted tree

Optimal area?
Not with divide & conquer approach, but can be computed with Dynamic Programming.
Visualization of Graphs

Lecture 1b:
Drawing Trees and Series-Parallel Graphs

Part III:
Radial Layouts
Phylogenetic tree by Colicelli, ScienceSignaling, 2004
Radial Layouts – Applications

Flare Visualization Toolkit code structure
by Heer, Bostock and Ogievetsky, 2010

Greek Myth Family
by Ribecca, 2011
Radial Layouts – Drawing Style

Drawing conventions

Drawing aesthetics
Radial Layouts – Drawing Style

**Drawing conventions**
- Vertices lie on circular layers according to their depth

**Drawing aesthetics**
Radial Layouts – Drawing Style

**Drawing conventions**
- Vertices lie on circular layers according to their depth
- Drawing is planar

**Drawing aesthetics**
Radial Layouts – Drawing Style

**Drawing conventions**
- Vertices lie on circular layers according to their depth
- Drawing is planar

**Drawing aesthetics**
- Distribution of the vertices
Radial Layouts – Drawing Style

**Drawing conventions**
- Vertices lie on circular layers according to their depth
- Drawing is planar

**Drawing aesthetics**
- Distribution of the vertices

How can an algorithm optimize the distribution of the vertices?
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:
Radial Layouts – Algorithm Attempt

Idea
- Reserve area corresponding to size $\ell(u)$ of $T(u)$:

\[
\tau_u = \ell(u) - 1
\]

- Place $u$ in middle of area

![Diagram](image.png)
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:

$$\tau_u = \ell(u) - 1$$

- Place $u$ in middle of area $v$

![Tree Diagram]

$$\ell(u)$$

$u$

$v$

![Tree Diagram]
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:

$$\tau_u = \ell(u) - 1$$

- Place $u$ in middle of area

![Diagram of a tree with nodes labeled 1 and branches indicating the structure.](diagram.png)
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:

$$
\tau_u = \ell(u) - 1
$$

- Place $u$ in middle of area $v$
Radial Layouts – Algorithm Attempt

Idea
- Reserve area corresponding to size $\ell(u)$ of $T(u)$:
  \[
  \tau_u = \ell(u) - 1
  \]
- Place $u$ in middle of area.
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:

\[
\tau_u = \ell(u) - 1
\]

- Place $u$ in middle of area
Radial Layouts – Algorithm Attempt

Idea
- Reserve area corresponding to size $\ell(u)$ of $T(u)$:

$$\tau_u = \ell(u) - 1$$
- Place $u$ in middle of area $v$.

\[ \begin{array}{c}
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
\end{array} \]

\[ \begin{array}{c}
\text{Idea} \\
\text{Reserve area corresponding to size } \ell(u) \text{ of } T(u): \\
\tau_u = \ell(u) - 1 \\
\text{Place } u \text{ in middle of area } v \\
\end{array} \]
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:

$$\tau_u = \ell(u) - 1$$

Place $u$ in middle of area $\ell(u)$.
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:

$$\tau_u = \frac{\ell(u)}{\ell(v) - 1}$$
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:

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

- Place $u$ in middle of area

Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:

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

- Place $u$ in middle of area
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:
  \[ \tau_u = \frac{\ell(u)}{\ell(v) - 1} \]

- Place $u$ in middle of area
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:
  $$\tau_u = \frac{\ell(u)}{\ell(v) - 1}$$

- Place $u$ in middle of area
Radial Layouts – Algorithm Attempt

**Idea**

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:

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

- Place $u$ in middle of area
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:
  \[ \tau_u = \frac{\ell(u)}{\ell(v) - 1} \]

- Place $u$ in middle of area
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:
  \[
  \tau_u = \frac{\ell(u)}{\ell(v) - 1}
  \]

- Place $u$ in middle of area
Radial Layouts – Algorithm Attempt

Idea

■ Reserve area corresponding to size $\ell(u)$ of $T(u)$:

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

■ Place $u$ in middle of area
Radial Layouts – Algorithm Attempt

Idea

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:
  \[ \tau_u = \frac{\ell(u)}{\ell(v) - 1} \]

- Place $u$ in middle of area
Radial Layouts – Algorithm Attempt

**Idea**

- Reserve area corresponding to size $\ell(u)$ of $T(u)$:
  \[ \tau_u = \frac{\ell(u)}{\ell(v) - 1} \]

- Place $u$ in middle of area
Radial Layouts – How To Avoid Crossings
Radial Layouts – How To Avoid Crossings
Radial Layouts – How To Avoid Crossings
Radial Layouts – How To Avoid Crossings
Radial Layouts – How To Avoid Crossings
Radial Layouts – How To Avoid Crossings

\( \tau_u \) – angle of the wedge corresponding to vertex \( u \)
Radial Layouts – How To Avoid Crossings

\( \tau_u \) – angle of the wedge corresponding to vertex \( u \)
Radial Layouts – How To Avoid Crossings

- $\tau_u$ – angle of the wedge corresponding to vertex $u$
- $\ell(u)$ – number of nodes in the subtree rooted at $u$
Radial Layouts – How To Avoid Crossings

- $\tau_u$ – angle of the wedge corresponding to vertex $u$
- $\ell(u)$ – number of nodes in the subtree rooted at $u$
- $\rho_i$ – radius of layer $i$
Radial Layouts – How To Avoid Crossings

- $\tau_u$ – angle of the wedge corresponding to vertex $u$
- $\ell(u)$ – number of nodes in the subtree rooted at $u$
- $\rho_i$ – radius of layer $i$
Radial Layouts – How To Avoid Crossings

- $\tau_u$ – angle of the wedge corresponding to vertex $u$
- $\ell(u)$ – number of nodes in the subtree rooted at $u$
- $\rho_i$ – radius of layer $i$
Radial Layouts – How To Avoid Crossings

- $\tau_u$ – angle of the wedge corresponding to vertex $u$
- $\ell(u)$ – number of nodes in the subtree rooted at $u$
- $\rho_i$ – radius of layer $i$
- $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$
Radial Layouts – How To Avoid Crossings

- $\tau_u$ – angle of the wedge corresponding to vertex $u$
- $\ell(u)$ – number of nodes in the subtree rooted at $u$
- $\rho_i$ – radius of layer $i$
- $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$
- $\tau_u = \min \left\{ \frac{\ell(u)}{\ell(v)-1}, 2 \arccos \frac{\rho_i}{\rho_{i+1}} \right\}$
Radial Layouts – How To Avoid Crossings

- $\tau_u$ – angle of the wedge corresponding to vertex $u$
- $\ell(u)$ – number of nodes in the subtree rooted at $u$
- $\rho_i$ – radius of layer $i$
- $\cos \frac{\tau_u}{2} = \frac{\rho_i}{\rho_{i+1}}$
- $\tau_u = \min \left\{ \frac{\ell(u)}{\ell(v)-1}, 2 \arccos \frac{\rho_i}{\rho_{i+1}} \right\}$
- Alternative:
  \[
  \alpha_{\text{min}} = \alpha_u - \arccos \frac{\rho_i}{\rho_{i+1}} \\
  \alpha_{\text{max}} = \alpha_u + \arccos \frac{\rho_i}{\rho_{i+1}}
  \]
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
    postorder($r$)
    preorder($r$, 0, 0, $2\pi$)
    return $(d_v, \alpha_v)_{v \in V(T)}$
    // vertex positions in polar coordinates
Radial Layouts – Pseudocode

RadialTreeLayout(tree \( T \), root \( r \in T \), radii \( \rho_1 < \cdots < \rho_k \))

begin

postorder(\( r \))

preorder(\( r, 0, 0, 2\pi \))

return \((d_v, \alpha_v)_{v \in V(T)}\)

\(/ / \text{vertex positions in polar coordinates}\)

postorder(vertex \( v \))

\( \ell(v) \leftarrow 1 \)

foreach child \( w \) of \( v \) do

\( \text{calculate the size of the subtree recursively} \)
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
    \[postorder(r)\]
    \[preorder(r, 0, 0, 2\pi)\]
    return \((d_v, \alpha_v)_{v \in V(T)}\)
\end

// vertex positions in polar coordinates

postorder(vertex $v$)
\[
\ell(v) \leftarrow 1
\]
\begin{footnotesize}
foreach child $w$ of $v$ do
\endfootnotesize
    \[postorder(w)\]
\[
\ell(v) \leftarrow \ell(v) + \ell(w)
\]
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
  $\text{postorder}(r)$
  $\text{preorder}(r, 0, 0, 2\pi)$
  return $(d_v,\alpha_v)_{v \in V(T)}$
// vertex positions in polar coordinates

$\text{postorder}(\text{vertex } v)$
\begin{align*}
  \ell(v) & \leftarrow 1 \\
  \text{foreach child } w \text{ of } v \text{ do} \\
  & \quad \text{postorder}(w) \\
  & \quad \ell(v) \leftarrow \ell(v) + \ell(w)
\end{align*}

$\text{preorder}(\text{vertex } v, t, \alpha_{\text{min}}, \alpha_{\text{max}})$
\begin{align*}
  d_v & \leftarrow \rho_t \\
  \alpha_v & \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2 \\
  \text{if } t > 0 \text{ then} \\
  & \quad \alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \rho_t/\rho_t+1\} \\
  & \quad \alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \rho_t/\rho_t+1\} \\
  \text{left} & \leftarrow \alpha_{\text{min}} \\
  \text{foreach child } w \text{ of } v \text{ do} \\
  & \quad \text{right} \leftarrow \text{left} + \ell(w) \\
  & \quad \ell(v) - 1 \cdot (\alpha_{\text{max}} - \alpha_{\text{min}}) \\
  & \quad \text{preorder}(w, t + 1, \text{left}, \text{right}) \\
  \text{left} & \leftarrow \text{right}
\end{align*}
Radial Layouts – Pseudocode

RadialTreeLayout(tree T, root r ∈ T, radii ρ₁ < · · · < ρₖ)
begin
postorder(r)
preorder(r, 0, 0, 2π)
return (dv, αv)v∈V(T)
/ / vertex positions in polar coordinates
postorder(vertex v)
ℓ(v) ←1
foreach child w of v do
postorder(w)
ℓ(v) ←ℓ(v) + ℓ(w)

preorder(vertex v, t, αmin, αmax)

dv ←ρt
αv ←(αmin + αmax)/2
if t> 0 then
αmin ←max{αmin,αv−arccos ρt
ρt+1
}
αmax ... ←αmin
foreach child w of v do
right ←left + ℓ(w)
ℓ(v)−1 ·(αmax −αmin)
preorder(w,t + 1,left,right)
left ←right
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
  postorder($r$)
  preorder($r, 0, 0, 2\pi$)
  return $(d_v, \alpha_v)_{v \in V(T)}$
// vertex positions in polar coordinates

postorder(vertex $v$)

\[
\ell(v) \leftarrow 1
\]

foreach child $w$ of $v$ do

  postorder($w$)

\[
\ell(v) \leftarrow \ell(v) + \ell(w)
\]

preorder(vertex $v$, $t$, $\alpha_{\text{min}}$, $\alpha_{\text{max}}$)

\[
d_v \leftarrow \rho_t
\]

\[
\alpha_v \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2
\]
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
    postorder($r$)
    preorder($r$, 0, 0, $2\pi$)
    return $(d_v, \alpha_v)_{v \in V(T)}$
// vertex positions in polar coordinates

postorder(vertex $v$)
\begin{align*}
    \ell(v) & \leftarrow 1 \\
    \text{foreach child } w \text{ of } v \text{ do} \\
    & \quad \text{postorder}(w) \\
    & \quad \ell(v) \leftarrow \ell(v) + \ell(w)
\end{align*}

preorder(vertex $v$, $t$, $\alpha_{\min}$, $\alpha_{\max}$)
\begin{align*}
    d_v & \leftarrow \rho_t \\
    \alpha_v & \leftarrow (\alpha_{\min} + \alpha_{\max})/2
\end{align*}
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
  postorder($r$)
  preorder($r, 0, 0, 2\pi$)
  return $(d_v, \alpha_v)_{v \in V(T)}$
end // vertex positions in polar coordinates

postorder(vertex $v$)
begin
  $\ell(v) \leftarrow 1$
  foreach child $w$ of $v$ do
    postorder($w$)
    $\ell(v) \leftarrow \ell(v) + \ell(w)$
end

preorder(vertex $v, t, \alpha_{\min}, \alpha_{\max}$)
begin
  $d_v \leftarrow \rho_t$
  $\alpha_v \leftarrow (\alpha_{\min} + \alpha_{\max})/2$
end //output
Radial Layouts – Pseudocode

RadialTreeLayout(tree \( T \), root \( r \in T \), radii \( \rho_1 < \cdots < \rho_k \))
begin
\begin{align*}
\text{postorder}(r) \\
\text{preorder}(r, 0, 0, 2\pi) \\
\text{return } (d_v, \alpha_v)_{v \in V(T)} \\
\end{align*}
\hfill // vertex positions in polar coordinates
end

postorder(vertex \( v \))
\begin{align*}
\ell(v) &\leftarrow 1 \\
\text{foreach child } w \text{ of } v \text{ do} \\
&\quad \text{postorder}(w) \\
&\quad \ell(v) \leftarrow \ell(v) + \ell(w)
\end{align*}

preorder(vertex \( v \), \( t \), \( \alpha_{\text{min}}, \alpha_{\text{max}} \))
\begin{align*}
&\quad d_v \leftarrow \rho_t \\
&\quad \alpha_v \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2 \\
\text{if } t > 0 \text{ then} \\
&\quad \alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \frac{\rho_t}{\rho_t+1}\} \\
&\quad \alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \frac{\rho_t}{\rho_t+1}\} \\
&\quad \leftarrow \text{left} + \ell(w) \\
&\quad \ell(v) - 1 \cdot (\alpha_{\text{max}} - \alpha_{\text{min}}) \\
&\quad \text{preorder}(w, t + 1, \text{left}, \text{right}) \\
&\quad \text{left} \leftarrow \text{right}
\end{align*}
\hfill //output
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
    postorder($r$)
    preorder($r, 0, 0, 2\pi$)
    return $(d_v, \alpha_v)_{v \in V(T)}$
    // vertex positions in polar coordinates
postorder(vertex $v$)
    $\ell(v) \leftarrow 1$
    foreach child $w$ of $v$ do
        postorder($w$)
        $\ell(v) \leftarrow \ell(v) + \ell(w)$

preorder(vertex $v$, $t$, $\alpha_{\text{min}}$, $\alpha_{\text{max}}$)
begin
    $d_v \leftarrow \rho_t$
    $\alpha_v \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2$
    //output
    if $t > 0$ then
        $\alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$
        $\alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$
        $\ell(v) \leftarrow 1 \cdot (\alpha_{\text{max}} - \alpha_{\text{min}})$
        foreach child $w$ of $v$ do
            right $\leftarrow$ left $+ \ell(w)$
            $\ell(v) - 1 \cdot (\alpha_{\text{max}} - \alpha_{\text{min}})$
            preorder($w$, $t + 1$, left, right)
            left $\leftarrow$ right
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
  postorder($r$)
  preorder($r$, 0, 0, $2\pi$)
return $(d_v, \alpha_v)_{v \in V(T)}$
// vertex positions in polar coordinates
postorder(vertex $v$)
  $\ell(v) \leftarrow 1$
  foreach child $w$ of $v$ do
    postorder($w$)
    $\ell(v) \leftarrow \ell(v) + \ell(w)$
preorder(vertex $v$, $t$, $\alpha_{\min}$, $\alpha_{\max}$)
begin
  $d_v \leftarrow \rho_t$
  $\alpha_v \leftarrow (\alpha_{\min} + \alpha_{\max})/2$
if $t > 0$ then
  $\alpha_{\min} \leftarrow \max\{\alpha_{\min}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$
  $\alpha_{\max} \leftarrow \min\{\alpha_{\max}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$
left $\leftarrow \alpha_{\min}$
foreach child $w$ of $v$ do
  right $\leftarrow$ left $+ \ell(w)$
  $\ell(v) - 1 \cdot (\alpha_{\max} - \alpha_{\min})$
preorder($w$, $t + 1$, left, right)
left $\leftarrow$ right
//output
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
  postorder($r$)
  preorder($r, 0, 0, 2\pi$)
return $(d_v, \alpha_v)_{v \in V(T)}$
// vertex positions in polar coordinates
postorder(vertex $v$)
  $\ell(v) \leftarrow 1$
  foreach child $w$ of $v$ do
    postorder($w$)
    $\ell(v) \leftarrow \ell(v) + \ell(w)$

preorder(vertex $v$, $t$, $\alpha_{\text{min}}$, $\alpha_{\text{max}}$)
begin
  $d_v \leftarrow \rho_t$
  $\alpha_v \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2$
//output
  if $t > 0$ then
    $\alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$
    $\alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$
  left $\leftarrow \alpha_{\text{min}}$
  foreach child $w$ of $v$ do
  right $\leftarrow$ left $+$ $\ell(w)$
  $\ell(v) \leftarrow 1 \cdot (\alpha_{\text{max}} - \alpha_{\text{min}})$
  preorder($w$, $t$ $+$ $1$, left, right)
  left $\leftarrow$ right
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin

  postorder($r$)
  preorder($r, 0, 0, 2\pi$)

  return $(d_v, \alpha_v)_{v \in V(T)}$
  // vertex positions in polar coordinates

postorder(vertex $v$)
begin

  $\ell(v) \leftarrow 1$

  foreach child $w$ of $v$ do
    postorder($w$)
    $\ell(v) \leftarrow \ell(v) + \ell(w)$
end

preorder(vertex $v$, $t$, $\alpha_{\text{min}}$, $\alpha_{\text{max}}$)$$
\begin{align*}
  d_v & \leftarrow \rho_t \\
  \alpha_v & \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2 \\
  \text{//output}
\end{align*}$

if $t > 0$ then
begin

  $\alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$
  $\alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$

  left $\leftarrow \alpha_{\text{min}}$

  foreach child $w$ of $v$ do

    right $\leftarrow$ left $+$ $\frac{\ell(w)}{\ell(v)-1} \cdot (\alpha_{\text{max}} - \alpha_{\text{min}})$

    preorder($w, t + 1, \text{left, right}$)

  left $\leftarrow$ right

end
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
  postorder($r$)
  preorder($r, 0, 0, 2\pi$)
return $(d_v, \alpha_v)_{v \in V(T)}$
// vertex positions in polar coordinates
postorder(vertex $v$)
\[ \ell(v) \leftarrow 1 \]
foreach child $w$ of $v$ do
  postorder($w$)
  \[ \ell(v) \leftarrow \ell(v) + \ell(w) \]

preorder(vertex $v$, $t$, $\alpha_{\text{min}}$, $\alpha_{\text{max}}$)
\[ d_v \leftarrow \rho_t \]
\[ \alpha_v \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2 \]
//output
if $t > 0$ then
\[ \alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\} \]
\[ \alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\} \]
left $\leftarrow \alpha_{\text{min}}$
foreach child $w$ of $v$ do
  right $\leftarrow$ left + $\frac{\ell(w)}{\ell(v)-1} \cdot (\alpha_{\text{max}} - \alpha_{\text{min}})$
  preorder($w, t + 1, left, right$)
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
postorder($r$)
prenode($r$, 0, 0, $2\pi$)
return $(d_v, \alpha_v)_{v \in V(T)}$
// vertex positions in polar coordinates

postorder(vertex $v$)
\[
\ell(v) \leftarrow 1
\]
foreach child $w$ of $v$ do
postorder($w$)
\[
\ell(v) \leftarrow \ell(v) + \ell(w)
\]

preorder(vertex $v$, $t$, $\alpha_{\min}$, $\alpha_{\max}$)
\[
d_v \leftarrow \rho_t
\]
\[
\alpha_v \leftarrow (\alpha_{\min} + \alpha_{\max})/2
\]
//output
if $t > 0$ then
\[
\alpha_{\min} \leftarrow \max\{\alpha_{\min}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}
\]
\[
\alpha_{\max} \leftarrow \min\{\alpha_{\max}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}
\]
left $\leftarrow \alpha_{\min}$
foreach child $w$ of $v$ do
\[
right \leftarrow \text{left} + \frac{\ell(w)}{\ell(v)-1} \cdot (\alpha_{\max} - \alpha_{\min})
\]
prenode($w$, $t + 1$, left, right)
left $\leftarrow$ right
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin
  postorder($r$)
  preorder($r, 0, 0, 2\pi$)
return $(d_v, \alpha_v)_{v \in V(T)}$ // vertex positions in polar coordinates
postorder(vertex $v$)
  $\ell(v) \leftarrow 1$
  foreach child $w$ of $v$ do
    postorder($w$)
    $\ell(v) \leftarrow \ell(v) + \ell(w)$
preorder(vertex $v$, $t$, $\alpha_{\text{min}}$, $\alpha_{\text{max}}$)
  $d_v \leftarrow \rho_1$
  $\alpha_v \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2$ //output
  if $t > 0$ then
    $\alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \frac{\rho_1}{\rho_1 + 1}\}$
    $\alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \frac{\rho_1}{\rho_1 + 1}\}$
  left $\leftarrow \alpha_{\text{min}}$
  foreach child $w$ of $v$ do
    right $\leftarrow$ left $+ \frac{\ell(w)}{\ell(v) - 1} \cdot (\alpha_{\text{max}} - \alpha_{\text{min}})$
    preorder($w, t + 1, \text{left}, \text{right}$)
    left $\leftarrow$ right

Runtime?
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)

begin
  postorder($r$)
  preorder($r, 0, 0, 2\pi$)
  return $(d_v, \alpha_v)_{v \in V(T)}$

// vertex positions in polar coordinates

postorder(vertex $v$)

$\ell(v) \leftarrow 1$

foreach child $w$ of $v$ do
  postorder($w$)
  $\ell(v) \leftarrow \ell(v) + \ell(w)$

preorder(vertex $v$, $t$, $\alpha_{\text{min}}$, $\alpha_{\text{max}}$)

$d_v \leftarrow \rho_t$

$\alpha_v \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2$

//output

if $t > 0$ then
  $\alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \left(\frac{\rho_t}{\rho_{t+1}}\right)\}$
  $\alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \left(\frac{\rho_t}{\rho_{t+1}}\right)\}$

left $\leftarrow \alpha_{\text{min}}$

foreach child $w$ of $v$ do
  right $\leftarrow$ left $+$ $\frac{\ell(w)}{\ell(v)-1} \cdot (\alpha_{\text{max}} - \alpha_{\text{min}})$
  preorder($w, t + 1, \text{left}, \text{right}$)
  left $\leftarrow$ right

Runtime? $\mathcal{O}(n)$
Radial Layouts – Pseudocode

RadialTreeLayout(tree $T$, root $r \in T$, radii $\rho_1 < \cdots < \rho_k$)
begin

\hspace{1cm} postorder($r$)
\hspace{1cm} preorder($r, 0, 0, 2\pi$)
\hspace{1cm} return $(d_v, \alpha_v)_{v \in V(T)}$
\hspace{1cm} // vertex positions in polar coordinates

\hspace{1cm} postorder(vertex $v$)
\hspace{2cm} $\ell(v) \leftarrow 1$
\hspace{2cm} foreach child $w$ of $v$ do
\hspace{3cm} postorder($w$)
\hspace{3cm} $\ell(v) \leftarrow \ell(v) + \ell(w)$

//output

\hspace{1cm} preorder(vertex $v, t, \alpha_{\text{min}}, \alpha_{\text{max}}$)
\hspace{2cm} $d_v \leftarrow \rho_t$
\hspace{2cm} $\alpha_v \leftarrow (\alpha_{\text{min}} + \alpha_{\text{max}})/2$
\hspace{2cm} if $t > 0$ then
\hspace{3cm} $\alpha_{\text{min}} \leftarrow \max\{\alpha_{\text{min}}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\}$
\hspace{3cm} $\alpha_{\text{max}} \leftarrow \min\{\alpha_{\text{max}}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\}$
\hspace{2cm} $left \leftarrow \alpha_{\text{min}}$
\hspace{2cm} foreach child $w$ of $v$ do
\hspace{3cm} $right \leftarrow left + \frac{\ell(w)}{\ell(v)-1} \cdot (\alpha_{\text{max}} - \alpha_{\text{min}})$
\hspace{3cm} preorder($w, t + 1, left, right$)
\hspace{3cm} $left \leftarrow right$

Runtime? $O(n)$
Correctness?
Radial Layouts – Pseudocode

RadialTreeLayout(tree \( T \), root \( r \in T \), radii \( \rho_1 < \cdots < \rho_k \))

\[
\begin{align*}
\text{begin} & \\
\ & \text{postorder}(r) & \\
\ & \text{preorder}(r, 0, 0, 2\pi) & \\
\ & \text{return } (d_v, \alpha_v)_{v \in V(T)} & // \text{vertex positions in polar coordinates}
\end{align*}
\]

postorder(vertex \( v \))

\[
\begin{align*}
\ & \ell(v) \leftarrow 1 & \\
\ & \text{foreach child } w \text{ of } v \text{ do} & \\
\ & \quad \text{postorder}(w) & \\
\ & \quad \ell(v) \leftarrow \ell(v) + \ell(w)
\end{align*}
\]

\[
\begin{align*}
\text{preorder}(\text{vertex } v, t, \alpha_{\min}, \alpha_{\max}) & \\
\ & d_v \leftarrow \rho_t & //\text{output} & \\
\ & \alpha_v \leftarrow (\alpha_{\min} + \alpha_{\max})/2 & \\
\ & \text{if } t > 0 \text{ then} & \\
\ & \quad \alpha_{\min} \leftarrow \max\{\alpha_{\min}, \alpha_v - \arccos \frac{\rho_t}{\rho_{t+1}}\} & \\
\ & \quad \alpha_{\max} \leftarrow \min\{\alpha_{\max}, \alpha_v + \arccos \frac{\rho_t}{\rho_{t+1}}\} & \\
\ & \text{left} \leftarrow \alpha_{\min} & \\
\ & \text{foreach child } w \text{ of } v \text{ do} & \\
\ & \quad \text{right} \leftarrow \text{left} + \ell(w) & \\
\ & \quad \ell(v) - 1 \cdot (\alpha_{\max} - \alpha_{\min}) & \\
\ & \text{preorder}(w, t + 1, \text{left}, \text{right}) & \\
\ & \text{left} \leftarrow \text{right}
\end{align*}
\]

Runtime? \( \mathcal{O}(n) \)

Correctness? \( \checkmark \)
Theorem.
Let $T$ be a tree with $n$ vertices. The RadialTreeLayout algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:

- $\Gamma$ is a radial drawing.
- Vertices lie on circle according to their depth.
- Area is quadratic in $\max\text{degree}(T) \times \text{height}(T)$ (see [GD Ch. 3.1.3] if interested).
Theorem.
Let $T$ be a tree with $n$ vertices. The RadialTreeLayout algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:
- $\Gamma$ is radial drawing

Area quadratic in $\text{max-degree}(T) \times \text{height}(T)$ (see [GD Ch. 3.1.3] if interested)
Theorem.
Let $T$ be a tree with $n$ vertices. The RadialTreeLayout algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:
- $\Gamma$ is radial drawing
- Vertices lie on circle according to their depth
(see [GD Ch. 3.1.3] if interested)
Radial Layouts – Result

Theorem.
Let $T$ be a tree with $n$ vertices. The RadialTreeLayout algorithm constructs in $O(n)$ time a drawing $\Gamma$ of $T$ s.t.:
- $\Gamma$ is radial drawing
- Vertices lie on circle according to their depth
- Area quadratic in $\text{max-degree}(T) \times \text{height}(T)$
  (see [GD Ch. 3.1.3] if interested)
Other tree visualisation styles

Writing Without Words: The project explores methods to visualise the differences in writing styles of different authors.

Similar to balloon layout
Other tree visualisation styles

A phylogenetically organised display of data for all placental mammal species.

Fractal layout
Other tree visualisation styles

A language family tree – in pictures
Other tree visualisation styles
Other tree visualisation styles

treevis.net
Visualization of Graphs

Lecture 1b:
Drawing Trees and Series-Parallel Graphs

Part IV:
Series-Parallel Graphs
Series-Parallel Graphs

A graph $G$ is \textit{series-parallel}, if
Series-Parallel Graphs

A graph $G$ is **series-parallel**, if
- it contains a single (directed) edge $(s, t)$, or
Series-Parallel Graphs

A graph $G$ is **series-parallel**, if

- it contains a single (directed) edge $(s, t)$, or
- it consists of two series-parallel graphs $G_1, G_2$
A graph $G$ is **series-parallel**, if

- it contains a single (directed) edge $(s, t)$, or
- it consists of two series-parallel graphs $G_1$, $G_2$ with sources $s_1$, $s_2$ and sinks $t_1$, $t_2$
Series-Parallel Graphs

A graph $G$ is **series-parallel**, if

- it contains a single (directed) edge $(s, t)$, or
- it consists of two series-parallel graphs $G_1, G_2$ with sources $s_1, s_2$ and sinks $t_1, t_2$ that are combined using one of the following rules:
A graph $G$ is **series-parallel**, if

- it contains a single (directed) edge $(s, t)$, or
- it consists of two series-parallel graphs $G_1$, $G_2$ with sources $s_1$, $s_2$ and sinks $t_1$, $t_2$ that are combined using one of the following rules:

**Series composition**

\[ t_1 = s_2 \]
A graph $G$ is **series-parallel**, if
- it contains a single (directed) edge $(s, t)$, or
- it consists of two series-parallel graphs $G_1$, $G_2$ with sources $s_1$, $s_2$ and sinks $t_1$, $t_2$ that are combined using one of the following rules:

**Series composition**

$G_1$ $G_2$ $t_1 = s_2$ $t_2$

**Parallel composition**

$G_1$ $G_2$ $t_1 = t_2$ $s_1 = s_2$
A graph $G$ is **series-parallel**, if
- it contains a single (directed) edge $(s, t)$, or
- it consists of two series-parallel graphs $G_1$, $G_2$ with sources $s_1$, $s_2$ and sinks $t_1$, $t_2$ that are combined using one of the following rules:

**Series composition**

$G_1$ with sources $s_1$, $s_2$ and sinks $t_1$, $s_2$

$G_2$ with sources $s_1$, $s_2$ and sinks $t_2$

$G_1 \times G_2$ with sources $s_1$, $s_2$ and sinks $t_2$

**Parallel composition**

$G_1$ with sources $s_1$, $s_2$ and sinks $t_1$, $s_2$

$G_2$ with sources $s_1$, $s_2$ and sinks $t_2$

$G_1 \parallel G_2$ with sources $s_1$, $s_2$ and sinks $t_1 = t_2$

Convince yourself that series-parallel graphs are planar!
Series-Parallel Graphs – Decomposition Tree

A decomposition tree of $G$ is a binary tree $T$ with nodes of three types: $S$, $P$ and $Q$: 
A decomposition tree of $G$ is a binary tree $T$ with nodes of three types: $S$, $P$ and $Q$:

- A $Q$-node represents a single edge
Series-Parallel Graphs – Decomposition Tree

A decomposition tree of $G$ is a binary tree $T$ with nodes of three types: $S$, $P$ and $Q$:

- A $Q$-node represents a single edge
- An $S$-node represents a series composition; its children $T_1$ and $T_2$ represent $G_1$ and $G_2$
A decomposition tree of \( G \) is a binary tree \( T \) with nodes of three types: \( S \), \( P \) and \( Q \):

- A \( Q \)-node represents a single edge
- An \( S \)-node represents a series composition; its children \( T_1 \) and \( T_2 \) represent \( G_1 \) and \( G_2 \)
- A \( P \)-node represents a parallel composition; its children \( T_1 \) and \( T_2 \) represent \( G_1 \) and \( G_2 \)
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example

The diagram illustrates a series-parallel graph decomposition. Each node represents a subgraph, and the connections show how these subgraphs are combined in series and parallel configurations.

- **Series** means subgraphs are connected end-to-end.
- **Parallel** means subgraphs are connected side-by-side.

The graph starts with a complex subgraph, which is then broken down into simpler series and parallel configurations, as indicated by the arrows and connections between nodes.
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example

The diagram illustrates the decomposition of a series-parallel graph. Each node represents a simpler graph, and the edges connect these nodes in series or parallel, adhering to the rules of series-parallel graph formation.
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Decomposition Example
Series-Parallel Graphs – Applications

Flowcharts

PERT-Diagrams

(Program Evaluation and Review Technique)
Series-Parallel Graphs – Applications

Flowcharts

Flowcharts

PERT-Diagrams

(Program Evaluation and Review Technique)

Computational complexity:
Series-parallel graphs often admit linear-time algorithms for $\mathcal{NP}$-hard problems, e.g., minimum maximal matching, MIS, Hamiltonian completion
Series-Parallel Graphs – Drawing Style

Drawing conventions

Drawing aesthetics
Series-Parallel Graphs – Drawing Style

Drawing conventions
■ Planarity

Drawing aesthetics
Series-Parallel Graphs – Drawing Style

**Drawing conventions**
- Planarity
- Straight-line edges

**Drawing aesthetics**
Series-Parallel Graphs – Drawing Style

**Drawing conventions**
- Planarity
- Straight-line edges
- Upward

**Drawing aesthetics**
Series-Parallel Graphs – Drawing Style

**Drawing conventions**
- Planarity
- Straight-line edges
- Upward

**Drawing aesthetics**
- Area
Series-Parallel Graphs – Drawing Style

**Drawing conventions**
- Planarity
- Straight-line edges
- Upward

**Drawing aesthetics**
- Area
- Symmetry
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

**Base case:** Q-nodes  

**Divide:** Draw $G_1$ and $G_2$ first
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first

Conquer:
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition

\[
\begin{align*}
\Delta(G) & \quad \Delta(G_1) & \quad \Delta(G_2) \\
\Delta(G_1) & \quad \Delta(G_2) \\
\Delta(G) & \quad s & \quad t
\end{align*}
\]
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree
- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

**Base case:** Q-nodes

**Divide:** Draw $G_1$ and $G_2$ first

**Conquer:**
- S-nodes / series composition
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes  Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition
- P-nodes / parallel composition
Series-Parallel Graphs – Straight-Line Drawings

**Divide & conquer algorithm using the decomposition tree**

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

**Base case:** Q-nodes  
**Divide:** Draw $G_1$ and $G_2$ first

**Conquer:**
- S-nodes / series composition
- P-nodes / parallel composition
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition
- P-nodes / parallel composition
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree
- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes  
Divide: Draw $G_1$ and $G_2$ first

Conquer:
- S-nodes / series composition
- P-nodes / parallel composition
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

**Base case:** Q-nodes  
**Divide:** Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition
- P-nodes / parallel composition

Do you see any problem?
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition
- P-nodes / parallel composition

Divide: Draw $G_1$ and $G_2$ first
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes

Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition
- P-nodes / parallel composition

Divide: Draw $G_1$ and $G_2$ first

Change embedding!
Series-Parallel Graphs – Straight-Line Drawings

Divide & conquer algorithm using the decomposition tree

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

Base case: Q-nodes  
Divide: Draw $G_1$ and $G_2$ first

Conquer:

- S-nodes / series composition
- P-nodes / parallel composition

Divide: Draw $G_1$ and $G_2$ first

change embedding!
Series-Parallel Graphs – Straight-Line Drawings

**Divide & conquer algorithm using the decomposition tree**
- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

**Base case:** Q-nodes

**Divide:** Draw $G_1$ and $G_2$ first

**Conquer:**
- S-nodes / series composition
- P-nodes / parallel composition

---

**Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$**

**Divide:** Draw $G_1$ and $G_2$ first

**Conquer:**
- S-nodes / series composition
- P-nodes / parallel composition

**change embedding!**
Series-Parallel Graphs – Straight-Line Drawings

**Divide & conquer algorithm using the decomposition tree**

- Draw $G$ inside a right-angled isosceles bounding triangle $\Delta(G)$

**Base case:** Q-nodes  

**Divide:** Draw $G_1$ and $G_2$ first

**Conquer:**

- S-nodes / series composition
- P-nodes / parallel composition

change embedding!
Series-Parallel Graphs – Straight-Line Drawings

- What makes parallel composition possible without creating crossings?
Series-Parallel Graphs – Straight-Line Drawings

- What makes parallel composition possible without creating crossings?
What makes parallel composition possible without creating crossings?
Series-Parallel Graphs – Straight-Line Drawings

- What makes parallel composition possible without creating crossings?
Series-Parallel Graphs – Straight-Line Drawings

- What makes parallel composition possible without creating crossings?
Series-Parallel Graphs – Straight-Line Drawings

- What makes parallel composition possible without creating crossings?
Series-Parallel Graphs – Straight-Line Drawings

What makes parallel composition possible without creating crossings?
What makes parallel composition possible without creating crossings?
What makes parallel composition possible without creating crossings?
Series-Parallel Graphs – Straight-Line Drawings

What makes parallel composition possible without creating crossings?
What makes parallel composition possible without creating crossings?
Series-Parallel Graphs – Straight-Line Drawings

What makes parallel composition possible without creating crossings?
What makes parallel composition possible without creating crossings?
Series-Parallel Graphs – Straight-Line Drawings

- What makes parallel composition possible without creating crossings?

Assume the following holds: the only vertex in $\angle(v)$ is $s$
Series-Parallel Graphs – Straight-Line Drawings

- What makes parallel composition possible without creating crossings?

- This condition is preserved during the induction step.

Assume the following holds: the only vertex in angle($v$) is $s$
Series-Parallel Graphs – Straight-Line Drawings

- What makes parallel composition possible without creating crossings?
  
  Assume the following holds: the only vertex in angle($v$) is $s$

- This condition is preserved during the induction step.

**Lemma.**

The drawing produced by the algorithm is planar.
Theorem.
Let $G$ be a series-parallel graph. Then $G$ (with variable embedding) admits a drawing $\Gamma$ that

- is upward planar
- is a straight-line drawing
- has area in $O(n^2)$
- isomorphic components of $G$ have congruent drawings up to translation.

$\Gamma$ can be computed in $O(n)$ time.
**Theorem.**

Let $G$ be a series-parallel graph. Then $G$ (with variable embedding) admits a drawing $\Gamma$ that

- is upward planar and

- Isomorphic components of $G$ have congruent drawings up to translation.

$\Gamma$ can be computed in $O(n)$ time.
Theorem. Let $G$ be a series-parallel graph. Then $G$ (with variable embedding) admits a drawing $\Gamma$ that
- is upward planar and
- a straight-line drawing
- Isomorphic components of $G$ have congruent drawings up to translation.
$\Gamma$ can be computed in $O(n)$ time.
Theorem.
Let $G$ be a series-parallel graph. Then $G$ (with **variable embedding**) admits a drawing $\Gamma$ that
- is upward planar and
- a straight-line drawing
- with area in $O(n^2)$. 

Isomorphic components of $G$ have congruent drawings up to translation.

$\Gamma$ can be computed in $O(n)$ time.
Theorem.
Let $G$ be a series-parallel graph. Then $G$ (with \textbf{variable embedding}) admits a drawing $\Gamma$ that
- is upward planar and
- a straight-line drawing
- with area in $\mathcal{O}(n^2)$.
- Isomorphic components of $G$ have congruent drawings up to translation.

$\Gamma$ can be computed in $\mathcal{O}(n)$ time.
Theorem.
Let $G$ be a series-parallel graph. Then $G$ (with variable embedding) admits a drawing $\Gamma$ that
- is upward planar and
- a straight-line drawing
- with area in $\mathcal{O}(n^2)$.
- Isomorphic components of $G$ have congruent drawings up to translation.
$\Gamma$ can be computed in $\mathcal{O}(n)$ time.
Theorem. [Bertolazzi et al. 94]
There exists a $2^n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]

There exists a $2^n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]
There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]
There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]
There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]
There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]

There exists a $2^n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Series-Parallel Graphs – Fixed Embedding

**Theorem.** [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

---

$$G_0 \quad G_n \quad G_{n+1}$$

$t_0$ $t_n$ $t_{n+1}$

$s_0$ $s_n$ $s_{n+1}$

$s_{n-1}$ $s_{n+1}$

$t_n$ $t_{n+1}$

---

$G_n$
Series-Parallel Graphs – Fixed Embedding

**Theorem.** [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.
Theorem. [Bertolazzi et al. 94]

There exists a \(2n\)-vertex series-parallel graph \(G_n\) such that any upward planar drawing of \(G_n\) that respects the embedding requires \(\Omega(4^n)\) area.
Theorem. [Bertolazzi et al. 94]

There exists a $2^n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

- $2 \cdot \text{Area}(G_n) < \text{Area}(\Pi)$
Theorem. [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

\[ 2 \cdot \text{Area}(G_n) < \text{Area}(\Pi) \]
Theorem. [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

\[ 2 \cdot \text{Area}(G_n) < \text{Area}(\Pi) \]
Theorem. [Bertolazzi et al. 94]

There exists a $2^n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

\[ 2 \cdot \text{Area}(G_n) < \text{Area}(\Pi) \]
Series-Parallel Graphs – Fixed Embedding

**Theorem.** [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

- $2 \cdot \text{Area}(G_n) < \text{Area}(\Pi)$
- $2 \cdot \text{Area}(\Pi) \leq \text{Area}(G_{n+1})$
Series-Parallel Graphs – Fixed Embedding

**Theorem.** [Bertolazzi et al. 94]

There exists a $2n$-vertex series-parallel graph $G_n$ such that any upward planar drawing of $G_n$ that respects the embedding requires $\Omega(4^n)$ area.

- $2 \cdot \text{Area}(G_n) < \text{Area}(\Pi)$
- $2 \cdot \text{Area}(\Pi) \leq \text{Area}(G_{n+1})$

$\Rightarrow 4 \cdot \text{Area}(G_n) \leq \text{Area}(G_{n+1})$
Literature

- [GD, Chapter 3] for divide and conquer methods for rooted trees and series-parallel graphs
- [Reingold, Tilford '81] “Tidier Drawings of Trees” original paper for level-based layout algo
- [Reingold, Supowit '83] “The complexity of drawing trees nicely” linear program and NP-hardness proof for area minimization
- treevis.net – compendium of drawing methods for trees