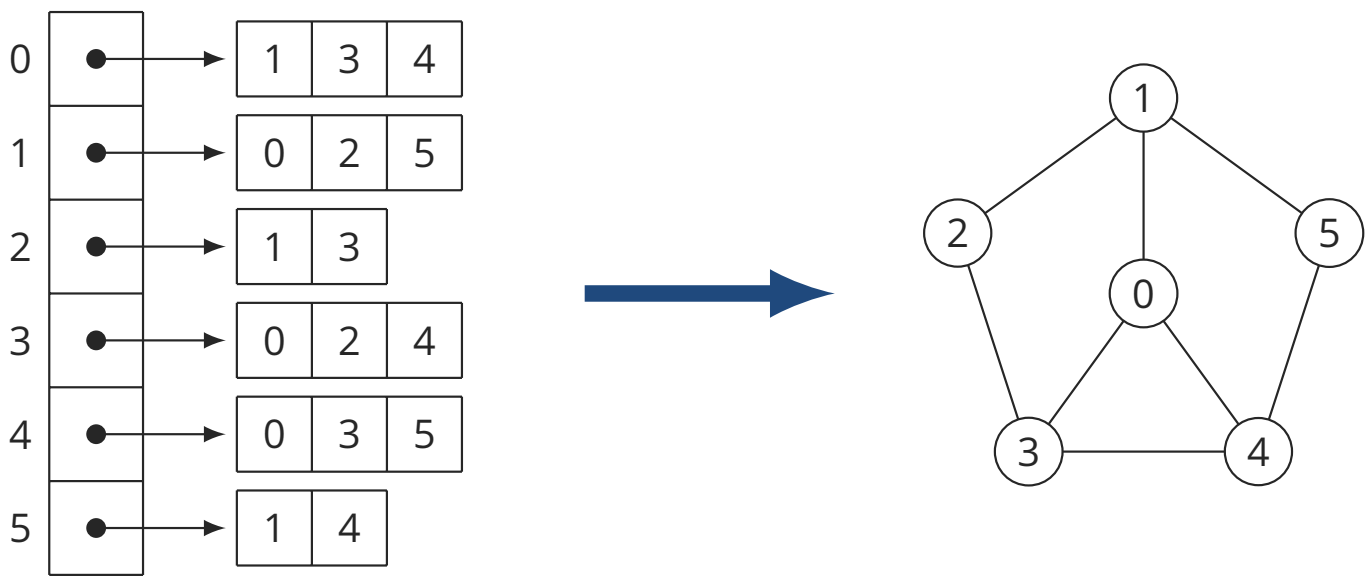
Graph Drawing

Embedding

Embedding

For a given graph G, an *embedding* (into $\mathbb{R}^2$) assigns each vertex a coordinate and each edge a (not necessarily straight) line connecting the corresponding coordinates.



Drawing Problem

Graph Drawing Problem

For a given graph G, find an embedding Γ such that Γ satisfies a certain set of properties, i. e., Γ is a "nice" drawing of G.

Possible properties

- Minimum number of edge crossings
- Small area
- Straight or short edges
- Good representation of graph structure
- ...

Drawing (Rooted) Trees

Why Starting with Trees

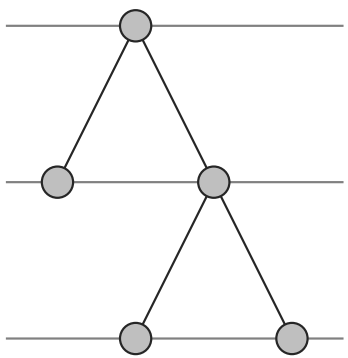
Approaching Complex Problems

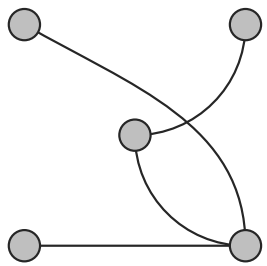
- Start with the easiest non-trivial variant.
- You do not encounter all complexity of the general problem.
- Usually, you learn something about the problem.
- You might be able to generalise your solution.

Drawing Rooted Binary Trees

Aesthetic Requirements

- Straight non-crossing edges.
- The layout displays the hierarchical structure of the tree, i. e., the y-coordinate of a node is given by its level.

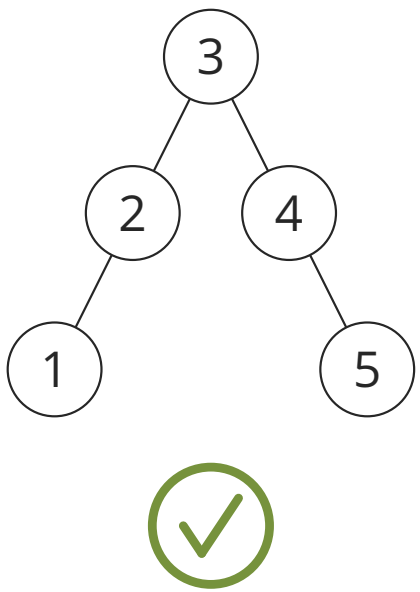


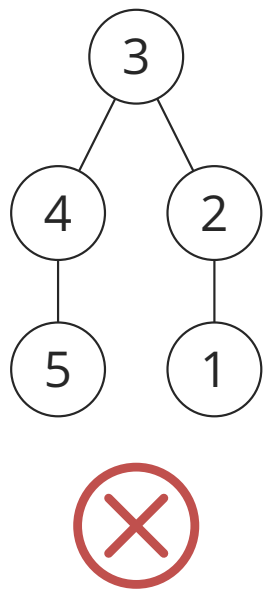


Drawing Rooted Binary Trees

Aesthetic Requirements

- The order of the children of a node is displayed in the drawing. If it is a search tree, maintain left and right ordering, i. e., left child is left of parent and right child is right of parent.

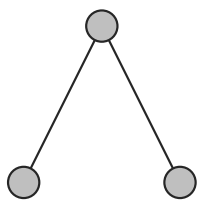


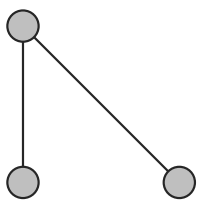


Drawing Rooted Binary Trees

Aesthetic Requirements

- A parent node is centred above its children. (Exception: only one child in a binary tree)

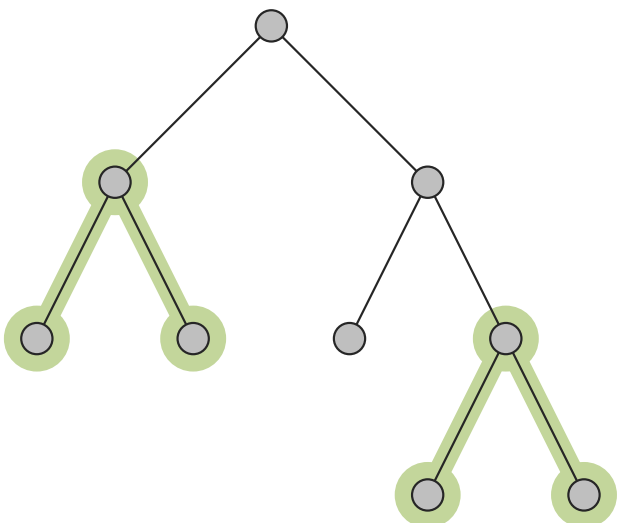


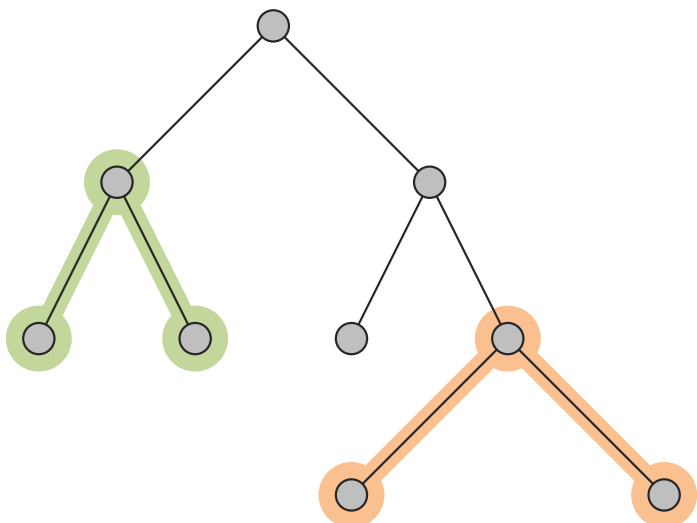


Drawing Rooted Binary Trees

Aesthetic Requirements

- The drawing of a subtree does not depend on its position in the tree, i. e., equal subtrees are drawn identically up to translation and symmetry.

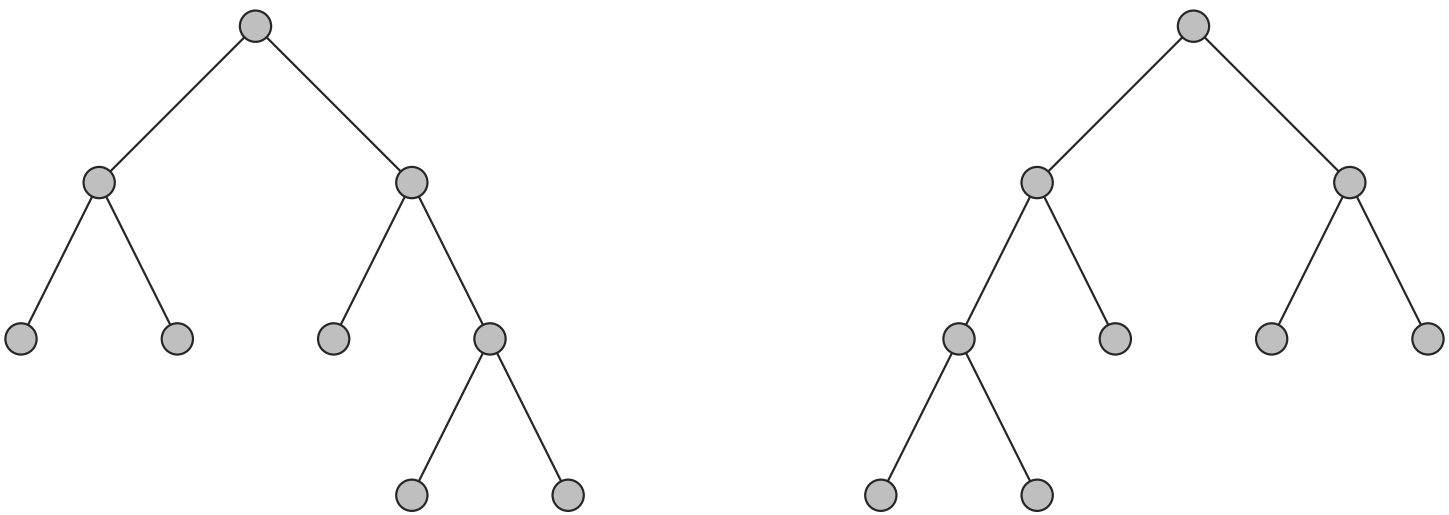




Drawing Rooted Binary Trees

Aesthetic Requirements

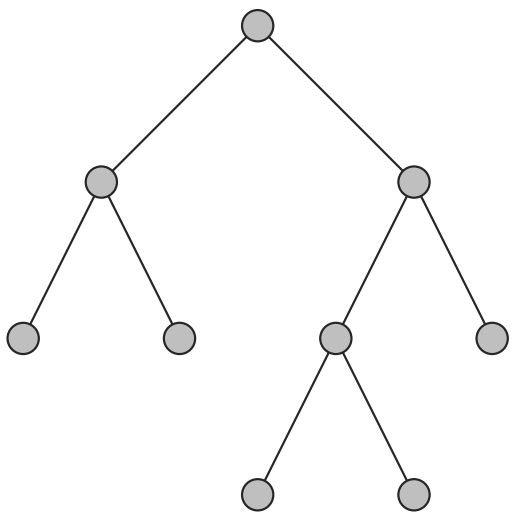
- The algorithm works symmetrically, i. e., the drawing of the reflection of a tree is the reflected drawing of the original tree.
  (more interesting for non-binary trees)

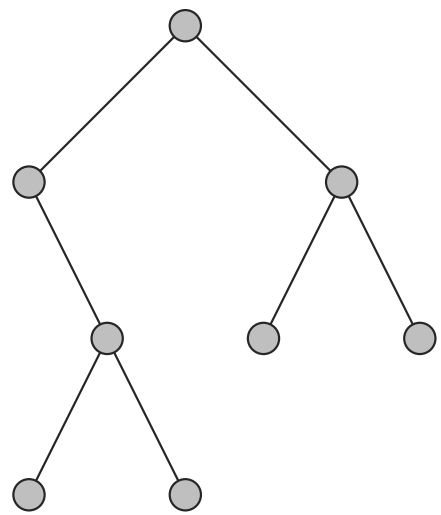


Reingold-Tilford Algorithm for Binary Trees

General Idea

- Draw the left and right subtree recursively.

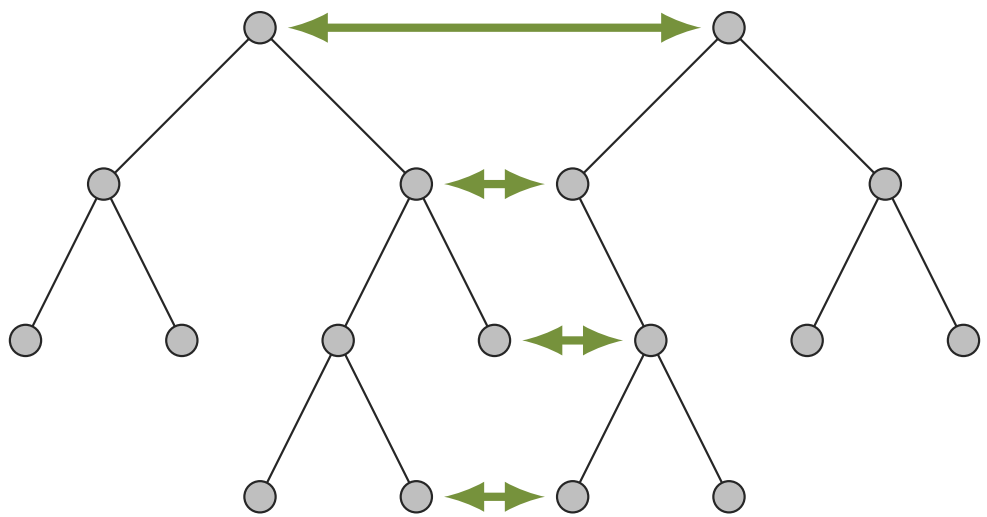




Reingold-Tilford Algorithm for Binary Trees

General Idea

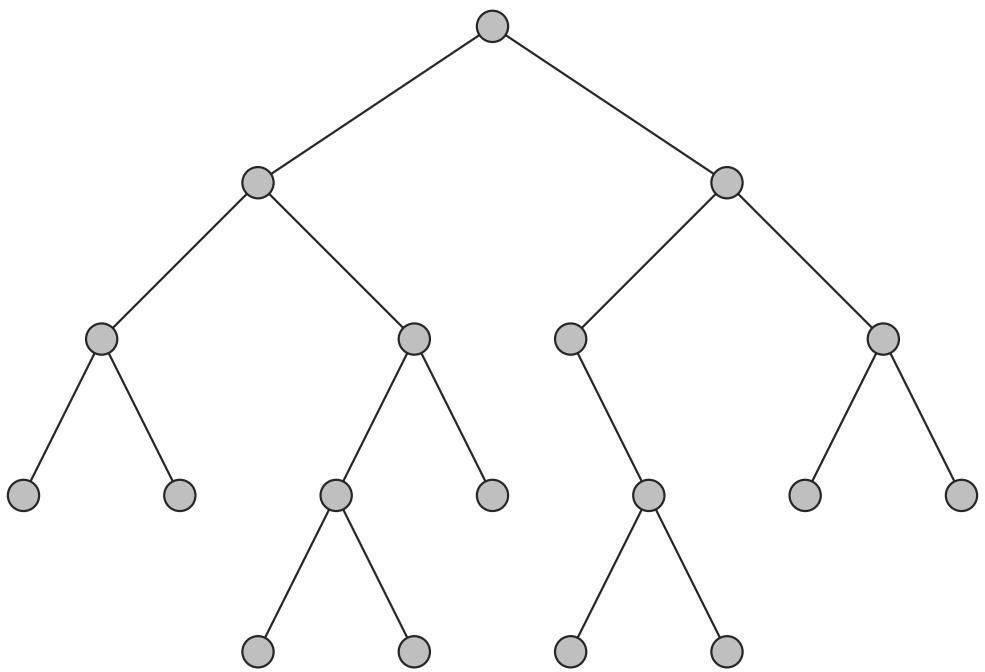
- Draw the left and right subtree recursively.
- Shift subtrees such that there is enough but minimal space between the subtrees.



Reingold-Tilford Algorithm for Binary Trees

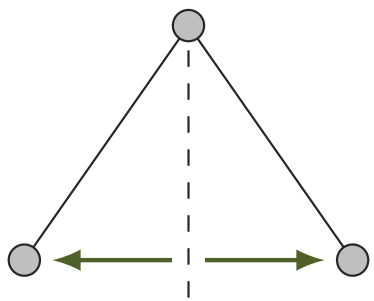
General Idea

- Draw the left and right subtree recursively.
- Shift subtrees such that there is enough but minimal space between the subtrees.
- Place the root in the middle between the roots of the subtrees.



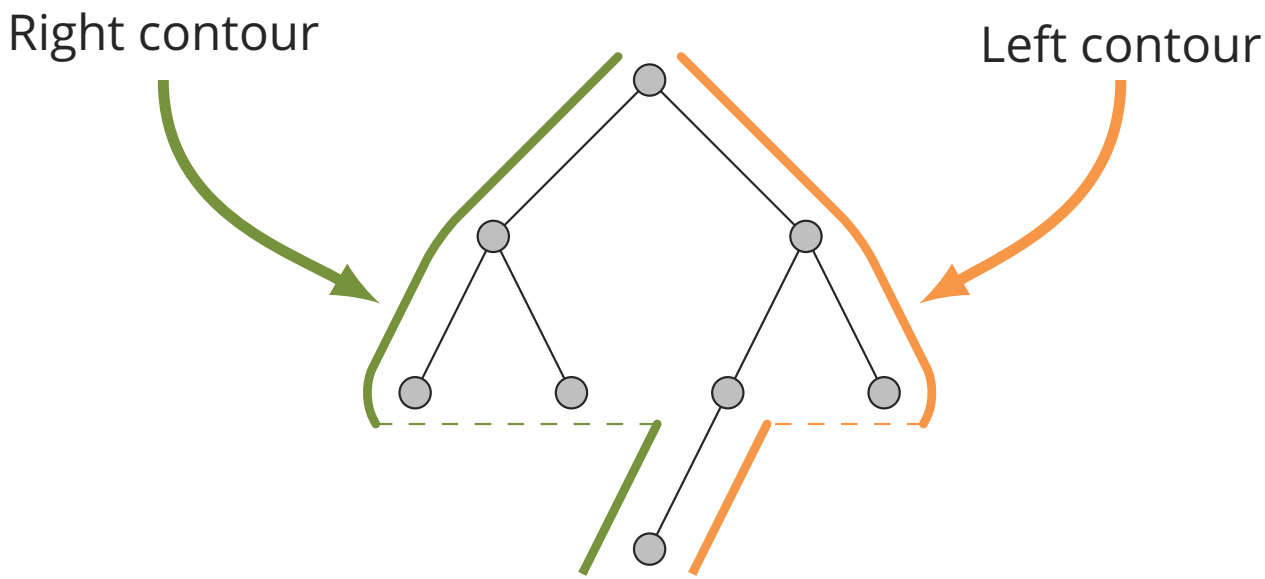
Shifting Trees

- Each node knows its x-coordinate relative to its parent.



Shifting Trees

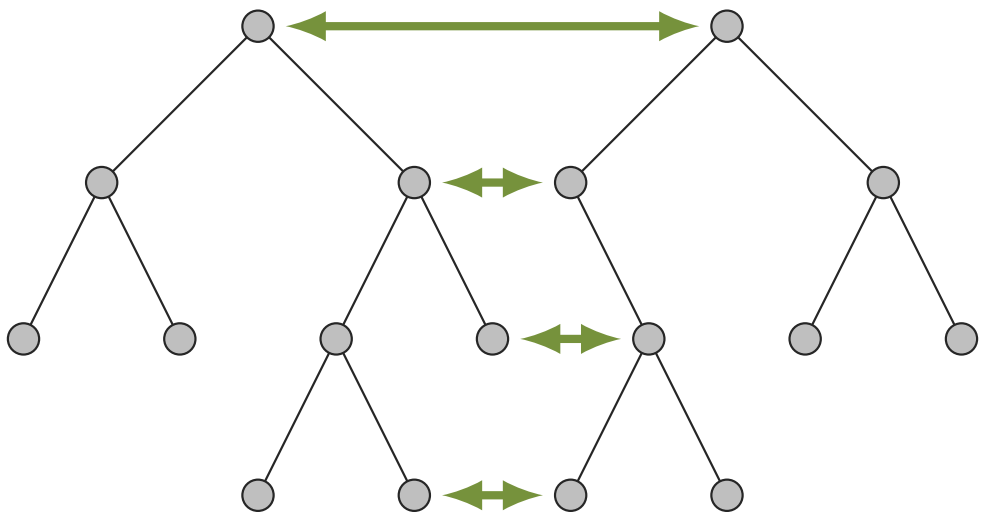
- Left (or right) *contour* of a subtree: The sequence of leftmost (or rightmost) nodes in each level. Can be implemented as list.



Reingold-Tilford Algorithm for Binary Trees

Shifting Trees

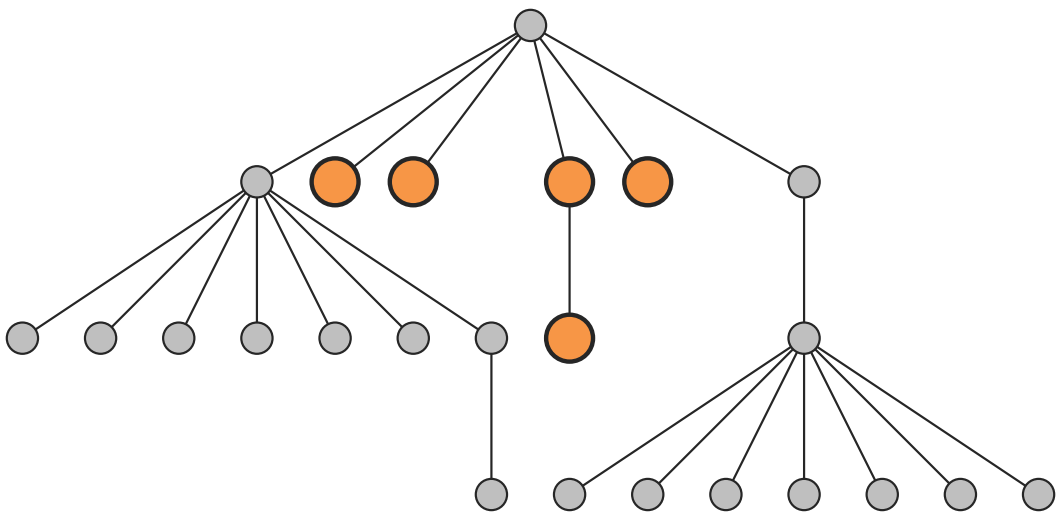
- Iterate top down over left contour of right subtree and right contour of left subtree. Compare coordinates of vertices and shift subtree if necessary.



Walker's Algorithm for Non-Binary Trees

Drawing Non-Binary Rooted Trees

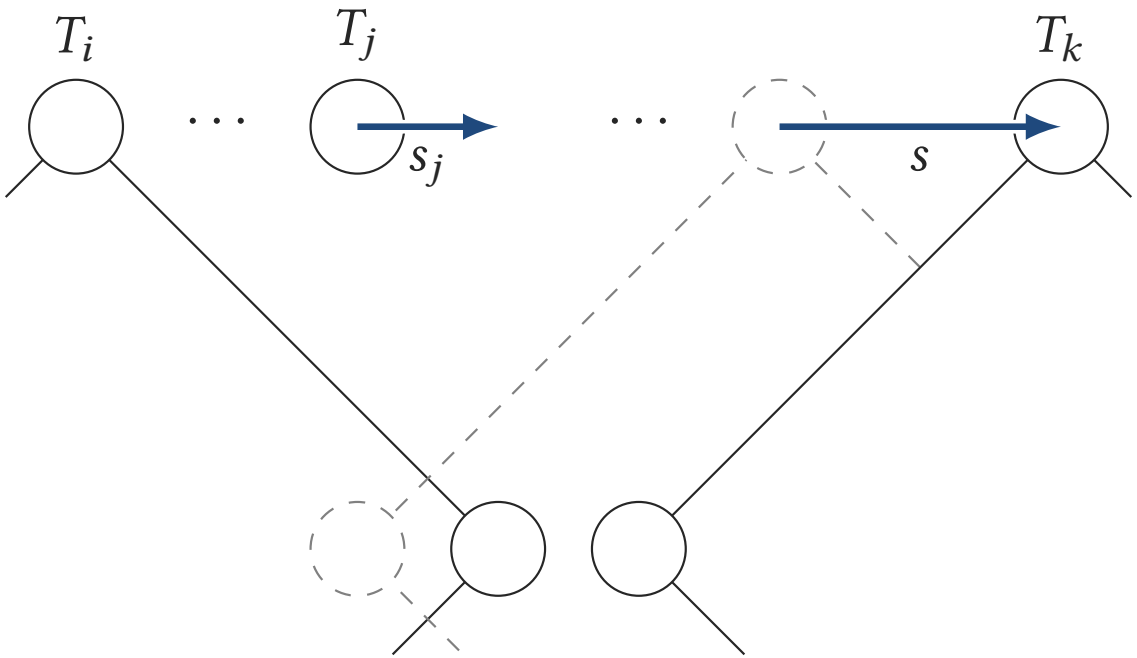
- Same approach as Reingold and Tilford: Draw subtrees recursively, shift them to avoid overlapping
- Problem: small subtrees between large subtrees



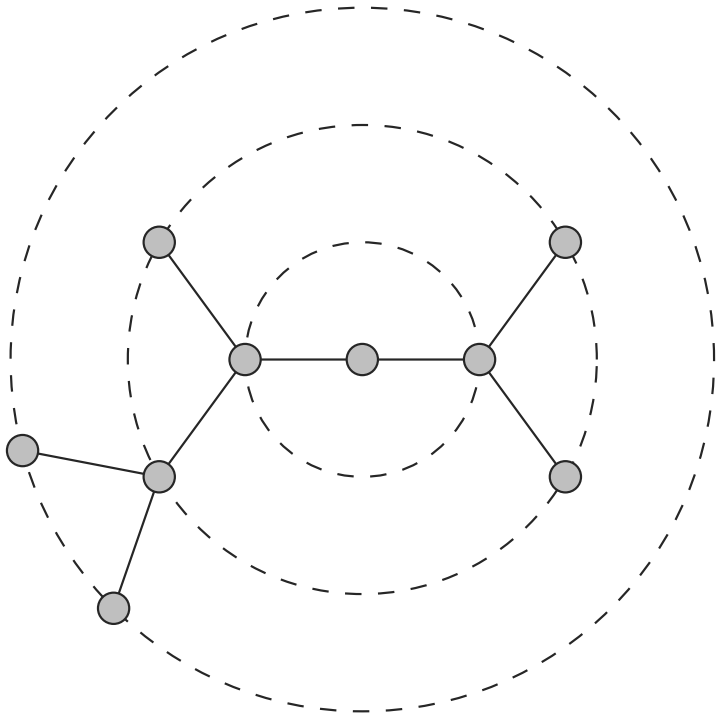
Walker's Algorithm for General Trees

Walker's Idea

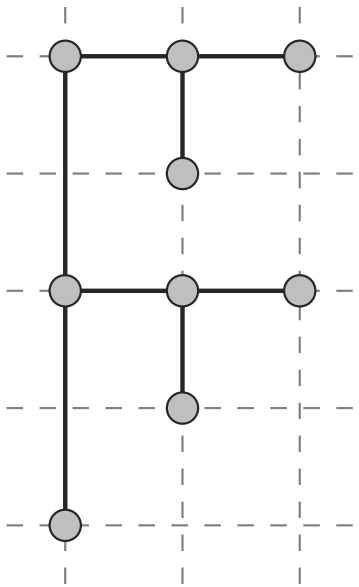
- Spread small subtrees evenly between large subtrees.
- If T_k is shifted by s because of T_i, shift all subtrees T_j between T_i and T_k by some fraction s_j of s



Other Approaches for Trees

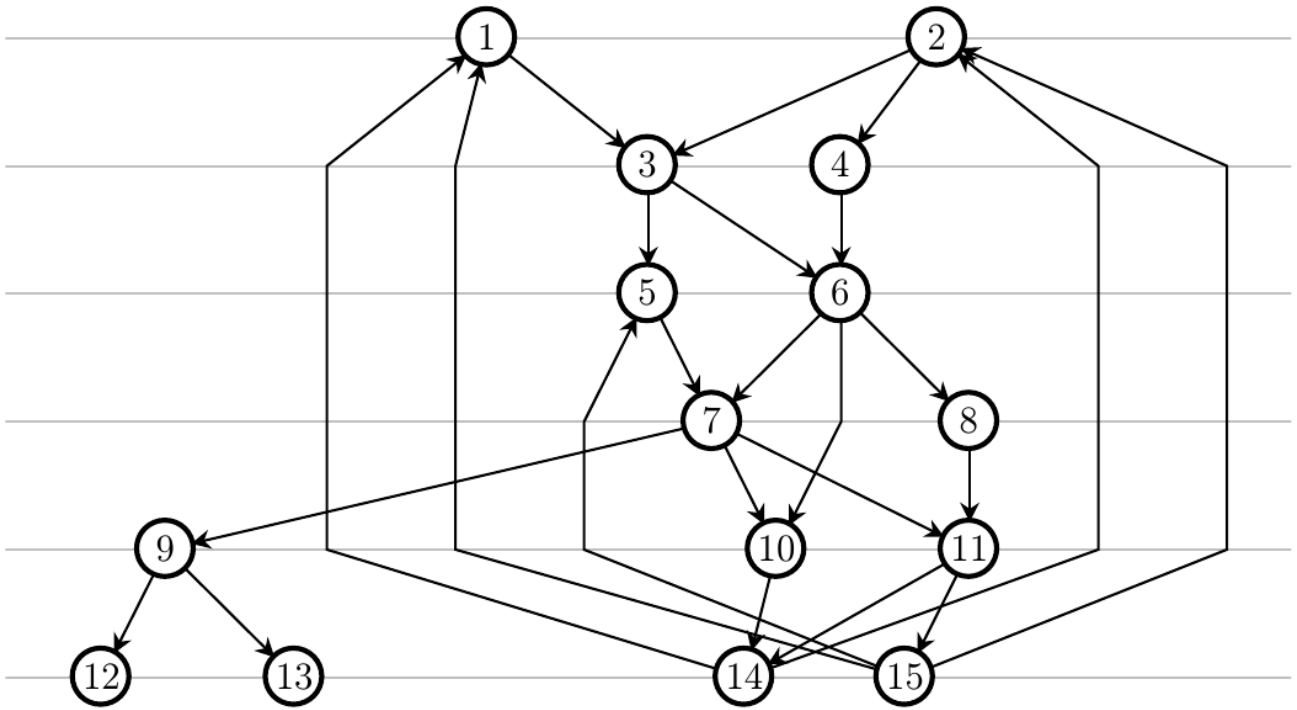


Other Approaches for Trees



Hierarchical Drawings

Hierarchical Drawings



Hierarchical Drawings

Hierarchical Drawings = Drawing Directed Acyclic Graphs

Requirements

- Edges point in a uniform direction (usually down)
- Polyline edges (straight between adjacent layers)
- Possibly short edges
- Possibly low number of edge crossings
- Possibly straight edges
- Possibly low width

Hierarchical Drawings

Basic approach

1. *Layer Assignment.* Assign each vertex to a layer, i. e., a y-coordinate. Try to minimise width or depth.

2. *Minimise Crossings.* Determine order of vertices within a layer. Long edges are replaced dummy vertices such that edges are only between two layers.

3. *Assign x-coordinates.* Parent should be close to children, ideally centred above. Straighten long edges, i. e., try to align dummy edges of same edge below each other.

Layer Assignment

Using Topological Order

- Assign all sinks to layer i.
- “Remove” sinks from graph.
- Repeat for $i + 1$ until graph is empty.

Properties

- Runs in linear time.
- Gives minimal height.
- No control over width.

Precedence Constrained Scheduling Problem

Given

- A set $S = \{J_1, \ldots, J_n\}$ of jobs (All jobs require one time unit.)
- A partial order $\prec$ on S (If $J_i \prec J_j$, J_i has to be completed before J_j can start.)
- A number k of processors

Problem

- Find a valid scheduling for S with minimal height.

Theorem

There is (probably) no polynomial time algorithm to solve the Precedence Constrained Scheduling Problem.

In 1972, Coffman and Graham gave an $\mathcal{O}(n^2)$ time 2-approximation algorithm, i. e., the generated hight is at most twice the optimal height.

Minimise Crossings

Minimise Crossings

- Determine order of vertices within a layer to minimise number of crossings.
- Exact x-coordinates have no influence on number of crossings.
- We can ignore edge directions.

Theorem Garey and Johnson 1983, Eades and Wormald 1994

For two given layers, there is (probably) no polynomial time algorithm to minimize the number of crossings, even if the order of vertices in one layer is fixed.

There are many heuristics to minimise the number of crossings.

Force Based Drawings

Force Based Drawings

General Idea

- Vertices are equally charged particles repelling each other. Force decreases (quadratically) with distance.
- Edges are springs. If length is larger than 1, vertices are pulled towards each other. If length is shorter than 1, vertices are pushed away from each other. Force increases (or decreases) linear with distance.

Algorithm

- Compute total force for each vertex.
- Move each vertex according to force.
- Repeat until stable position is found, i. e., force at each vertex is 0.