On Maximizing the Euclidian Distance between Adjacent Vertices in Planar Drawings of Small Area

Benjamin Çoban

Wilhelm-Schickard-Institut für Informatik
Universität Tübingen, Germany

Introduction

- Producing drawings of graphs with geometric constraints is a core topic of Graph Drawing

Introduction

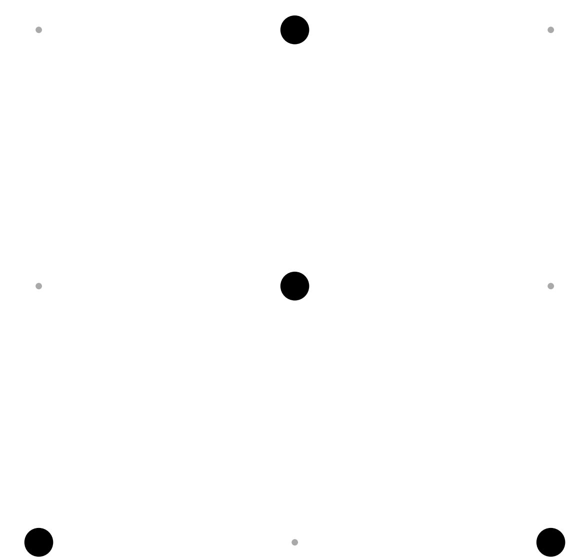
- Producing drawings of graphs with geometric constraints is a core topic of Graph Drawing
- Planar <u>polyline graph drawings</u>

Introduction

- Producing drawings of graphs with geometric constraints is a core topic of Graph Drawing
- Planar polyline graph drawings
  - Vertices placed on points on underlying grid

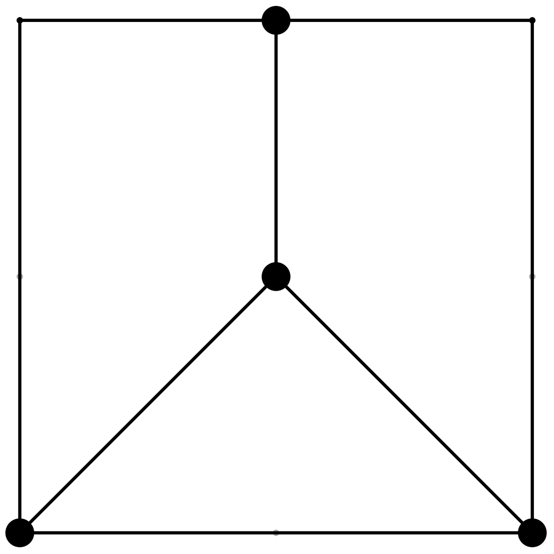
Introduction

- Producing drawings of graphs with geometric constraints is a core topic of Graph Drawing
- Planar <u>polyline graph drawings</u>
  - Vertices placed on points on underlying grid



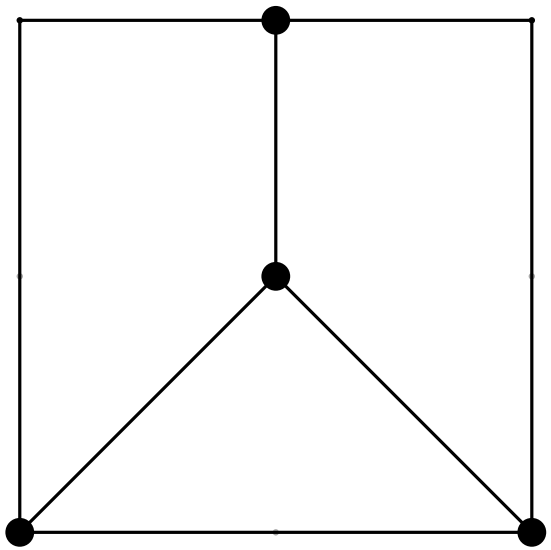
Introduction

- Producing drawings of graphs with geometric constraints is a core topic of Graph Drawing
- Planar polyline graph drawings
  - Vertices placed on points on underlying grid
  - Edges are sequences of straight-line segments



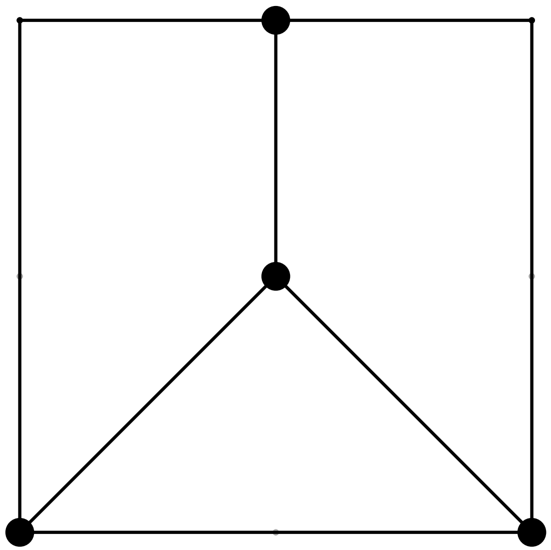
Introduction

- Producing drawings of graphs with geometric constraints is a core topic of Graph Drawing
- Planar polyline graph drawings
  - Vertices placed on points on underlying grid
  - Edges are sequences of straight-line segments
  - Bend points lie on grid points



Introduction

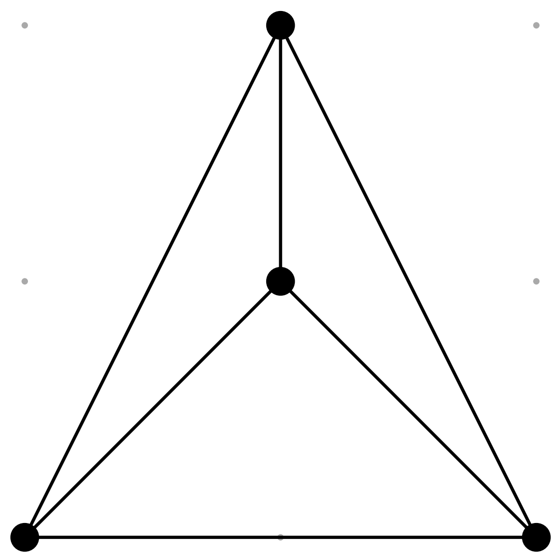
- Producing drawings of graphs with geometric constraints is a core topic of Graph Drawing
- Planar polyline graph drawings
  - Vertices placed on points on underlying grid
  - Edges are sequences of straight-line segments
  - Bend points lie on grid points



- Edge length is the sum of lengths of each line segment

Introduction

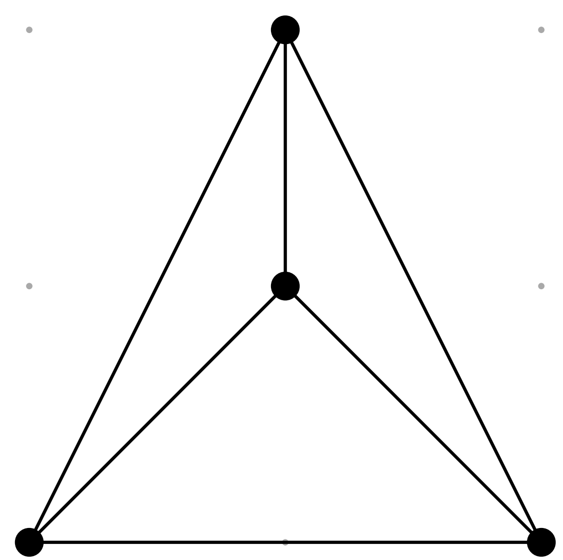
- Producing drawings of graphs with geometric constraints is a core topic of Graph Drawing
- Planar polyline graph drawings
  - Vertices placed on points on underlying grid
  - Edges are sequences of straight-line segments
  - Bend points lie on grid points



- Edge length is the sum of lengths of each line segment
- Straight-line drawing is a polyline drawing without bends

Introduction

- Producing drawings of graphs with geometric constraints is a core topic of Graph Drawing
- Planar polyline graph drawings
  - Vertices placed on points on underlying grid
  - Edges are sequences of straight-line segments
  - Bend points lie on grid points



- Edge length is the sum of lengths of each line segment
- Straight-line drawing is a polyline drawing without bends
- Total grid size determines geometrical properties

Graph Drawing Symposium

- International Symposium of Graph Drawing

Graph Drawing Symposium

- International Symposium of Graph Drawing
  - Main annual event for graph drawings and network visualizations

Graph Drawing Symposium

- International Symposium of Graph Drawing
  - Main annual event for graph drawings and network visualizations
  - *Live Challenge* as part of the contest includes heuristic approaches for geometrical problems of graph drawings on a fixed grid size

Graph Drawing Symposium

- International Symposium of Graph Drawing
  - Main annual event for graph drawings and network visualizations
  - *Live Challenge* as part of the contest includes heuristic approaches for geometrical problems of graph drawings on a fixed grid size
- This work addresses producing drawing algorithms for specific graph classes which will approach the geometrical problems with results described asymptotically

Live Challenge 2021

- Minimize the *edge length* ratio of the *longest* and *shortest* polyline

Live Challenge 2021

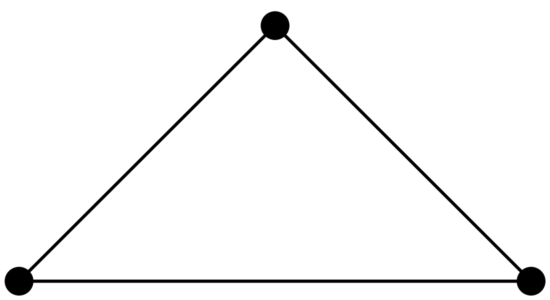
- Minimize the *edge length* ratio of the *longest* and *shortest* polyline
  - Edge length is the sum of lengths of each line segment

Live Challenge 2021

- Minimize the *edge length* ratio of the *longest* and *shortest* polyline
  - Edge length is the sum of lengths of each line segment
  - Ratio $r_{2021} = \frac{l_{\max}}{l_{\min}}$

Live Challenge 2021

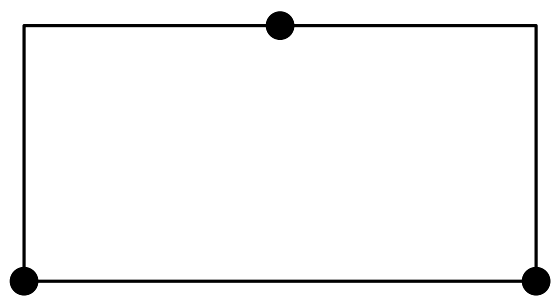
- Minimize the *edge length* ratio of the *longest* and *shortest* polyline
  - Edge length is the sum of lengths of each line segment
  - Ratio $r_{2021} = \frac{l_{\max}}{l_{\min}}$
  - Ratio *optimal*, if $r_{2021} = 1$



$r_{2021} \neq 1$

Live Challenge 2021

- Minimize the *edge length* ratio of the *longest* and *shortest* polyline
  - Edge length is the sum of lengths of each line segment
  - Ratio $r_{2021} = \frac{l_{\max}}{l_{\min}}$
  - Ratio *optimal*, if $r_{2021} = 1$



$r_{2021} = 1$

Live Challenge 2022

- Minimize the ratio of the *longest* polyline and the *shortest Euclidian distance* δ between adjacent vertices

Live Challenge 2022

- Minimize the ratio of the *longest* polyline and the *shortest Euclidian distance* δ between adjacent vertices
  - Topic of this work

Live Challenge 2022

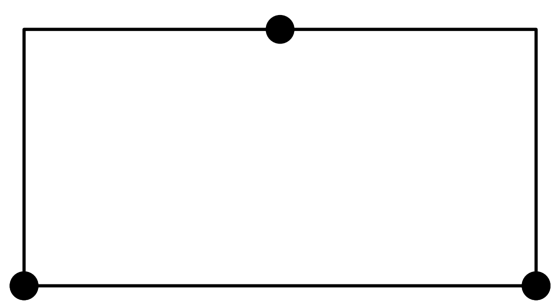
- Minimize the ratio of the *longest* polyline and the *shortest Euclidian distance* δ between adjacent vertices
  - Topic of this work
  - Ratio $r = \frac{l_{\text{max}}}{\delta_{\text{min}}}$

Live Challenge 2022

- Minimize the ratio of the *longest* polyline and the *shortest Euclidian distance* δ between adjacent vertices
  - Topic of this work
  - Ratio $r = \frac{l_{\text{max}}}{\delta_{\text{min}}}$
  - Ratio *optimal*, if $r = 1$

Live Challenge 2022

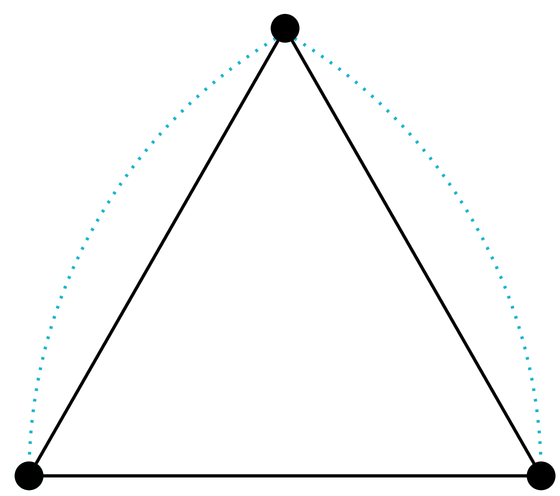
- Minimize the ratio of the *longest* polyline and the *shortest Euclidian distance* δ between adjacent vertices
  - Topic of this work
  - Ratio $r = \frac{l_{\max}}{\delta_{\min}}$
  - Ratio *optimal*, if $r = 1$



$$r = \sqrt{2}$$

Live Challenge 2022

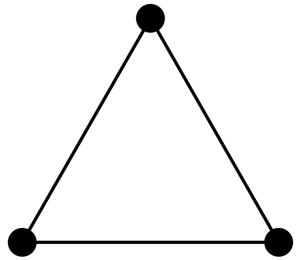
- Minimize the ratio of the *longest* polyline and the *shortest Euclidian distance* δ between adjacent vertices
  - Topic of this work
  - Ratio $r = \frac{l_{\text{max}}}{\delta_{\text{min}}}$
  - Ratio *optimal*, if $r = 1$



$r = 1 + \varepsilon$, ε depends on grid size

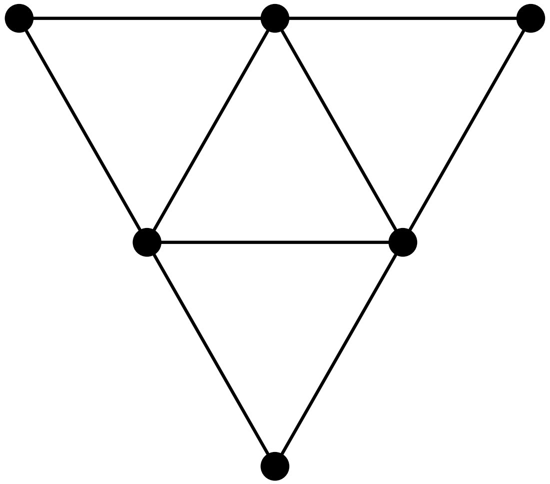
Adjacent and Nested Triangles

- $r = 1 + \varepsilon$



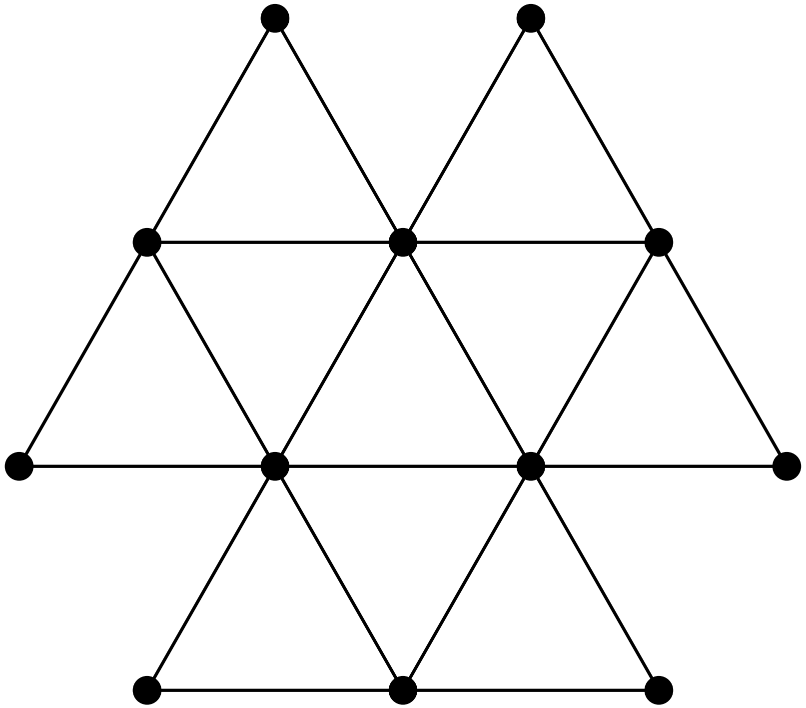
Adjacent and Nested Triangles

- $r = 1 + \varepsilon$



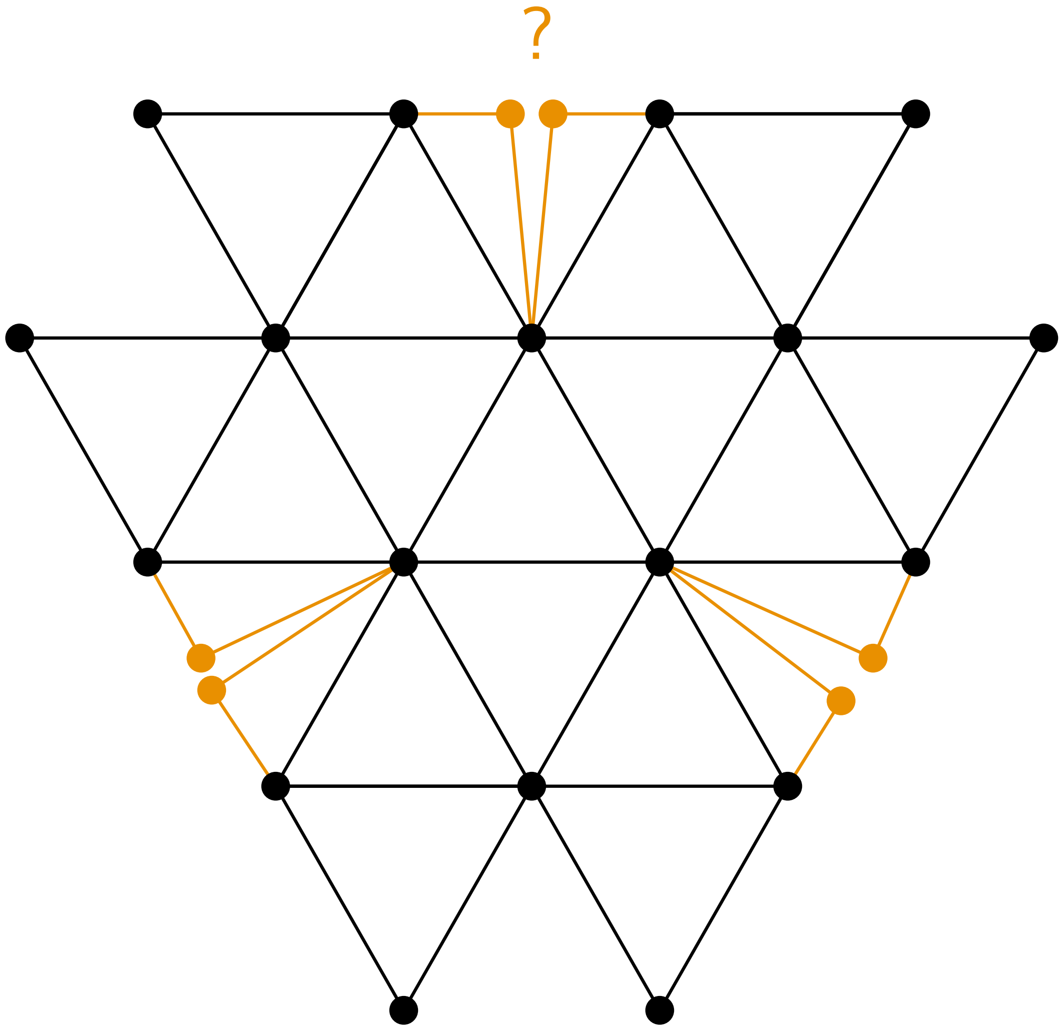
Adjacent and Nested Triangles

- $r = 1 + \varepsilon$



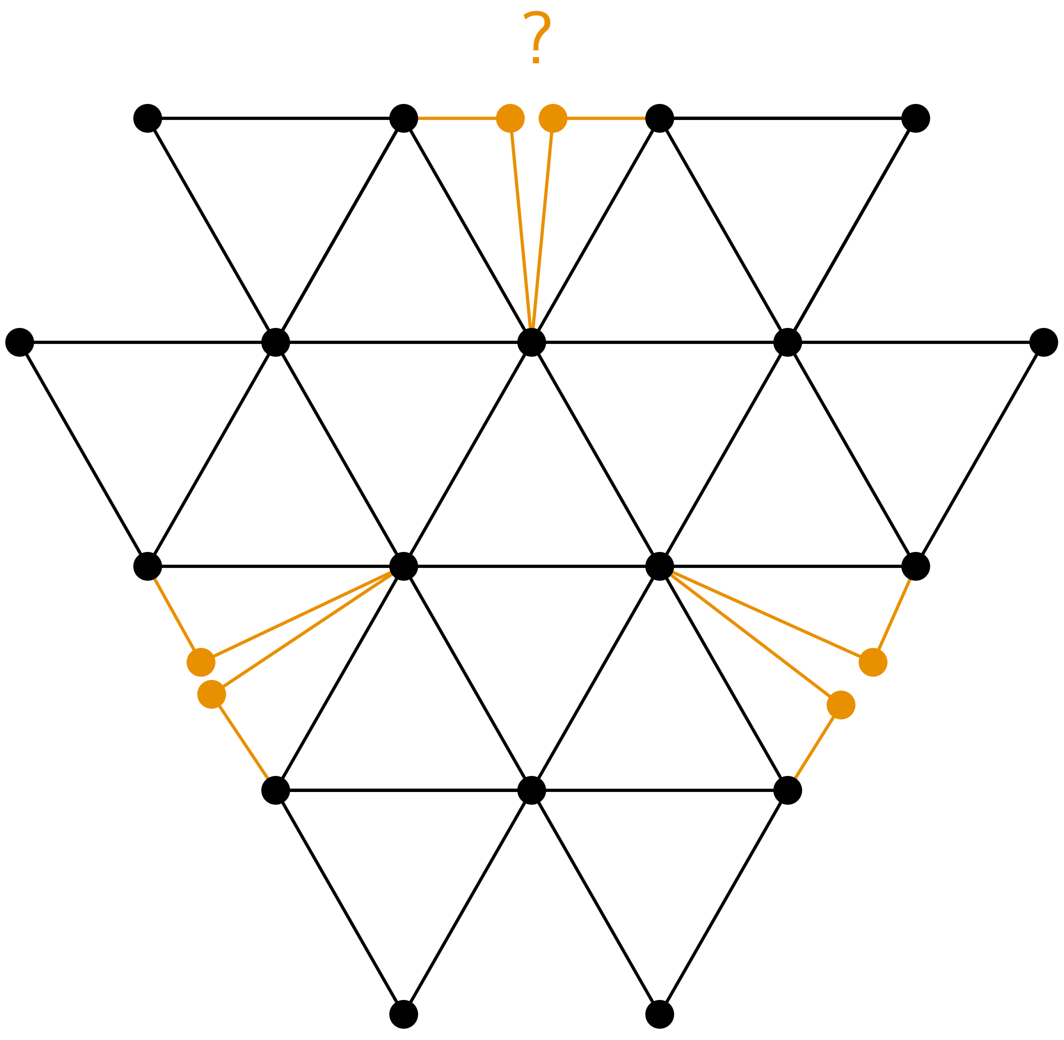
Adjacent and Nested Triangles

- $r = 2 + \varepsilon$

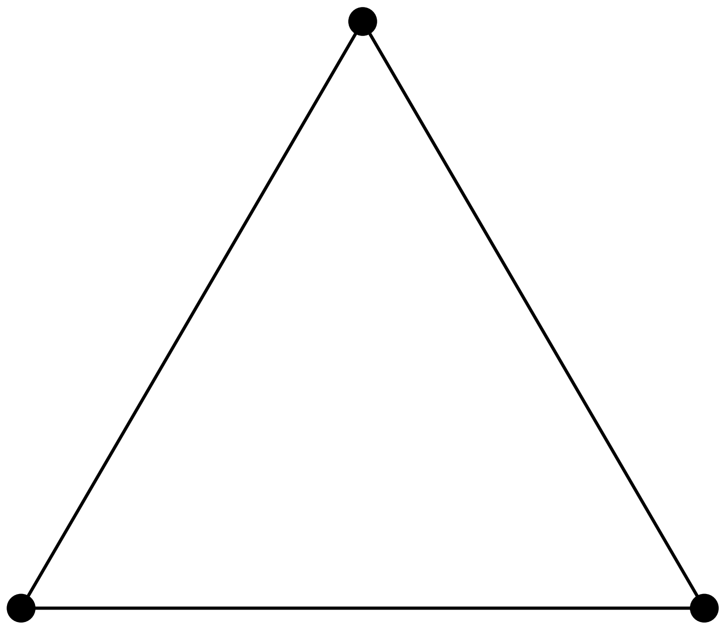


Adjacent and Nested Triangles

- $r = 2 + \varepsilon$

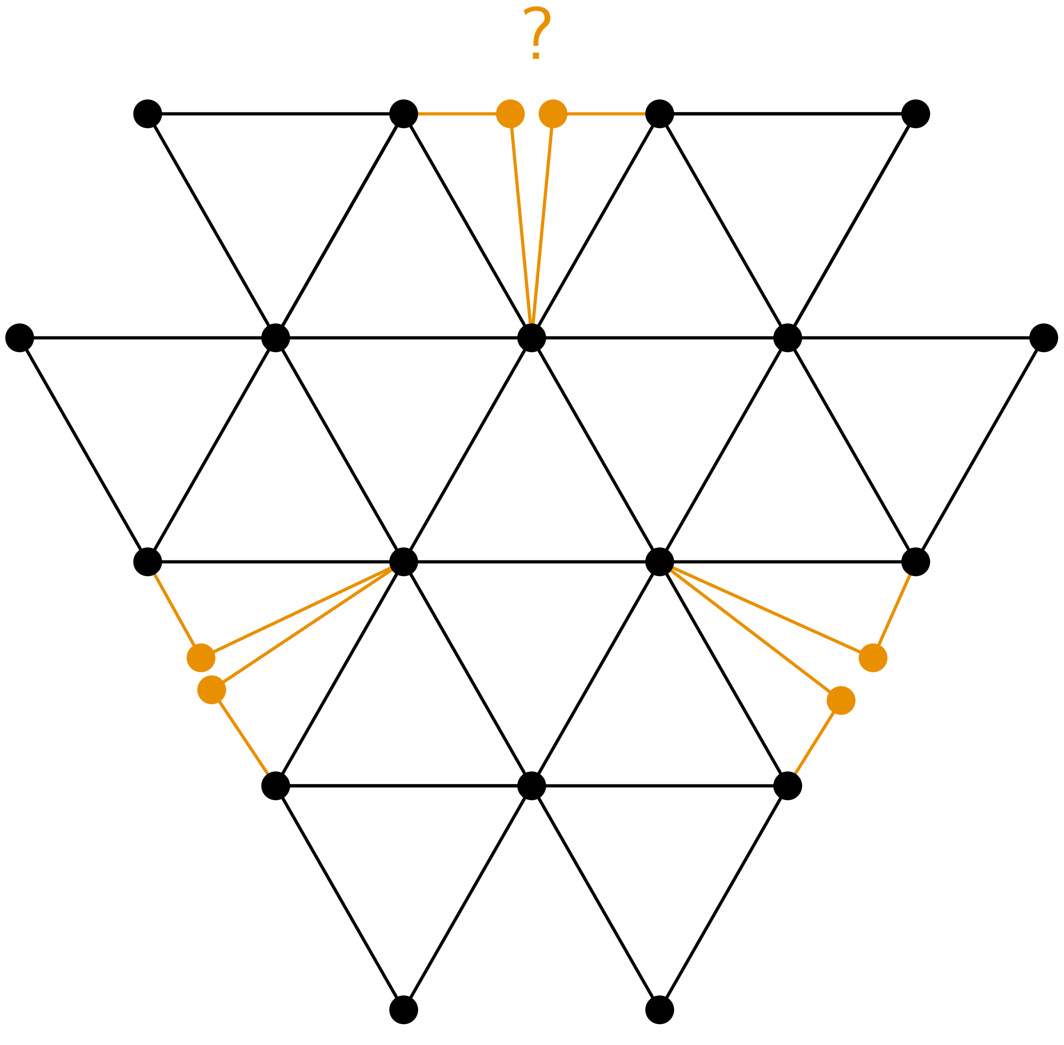


- $r = 1 + \varepsilon$

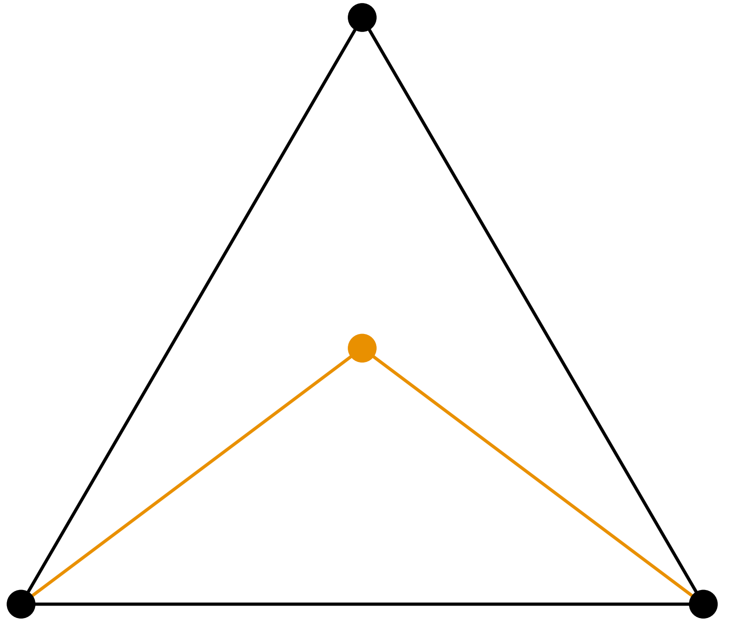


Adjacent and Nested Triangles

- $r = 2 + \varepsilon$

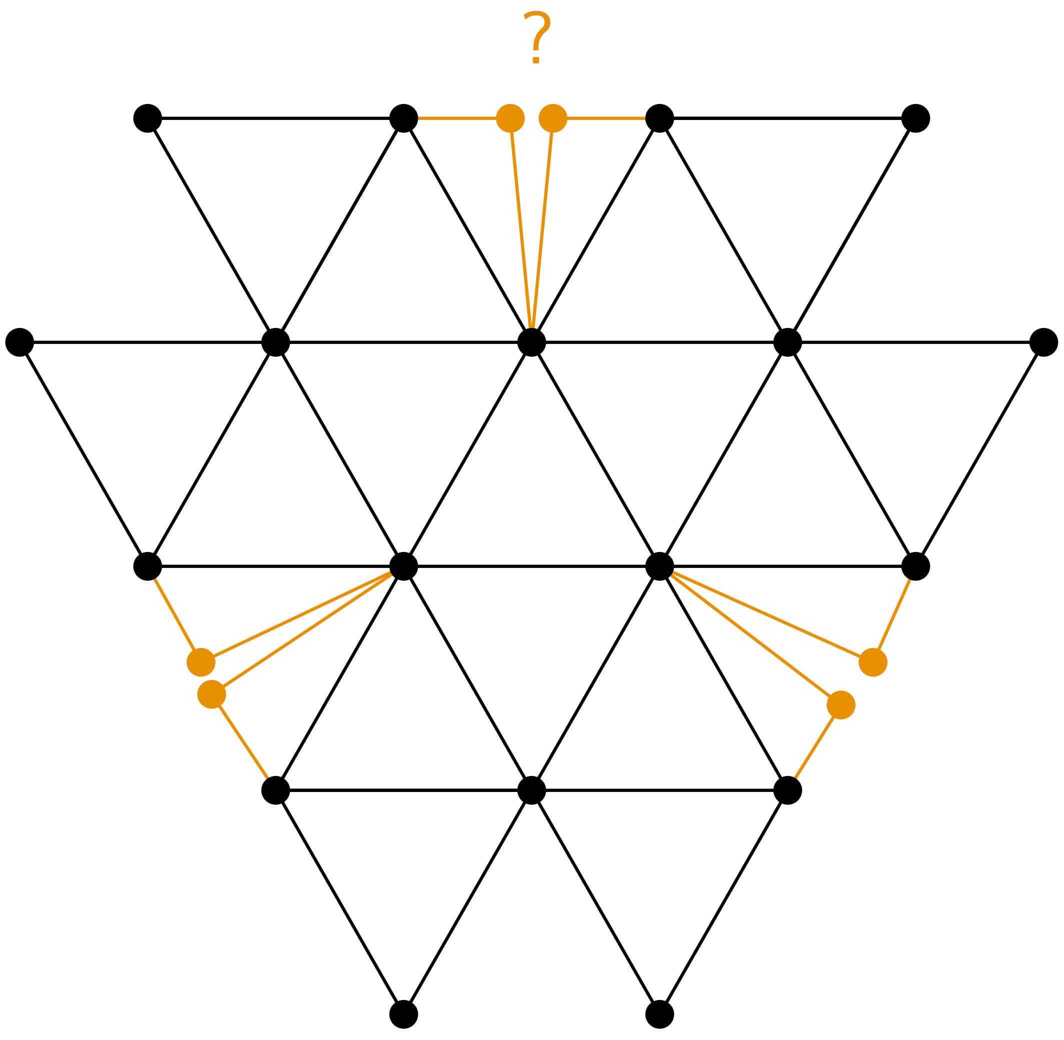


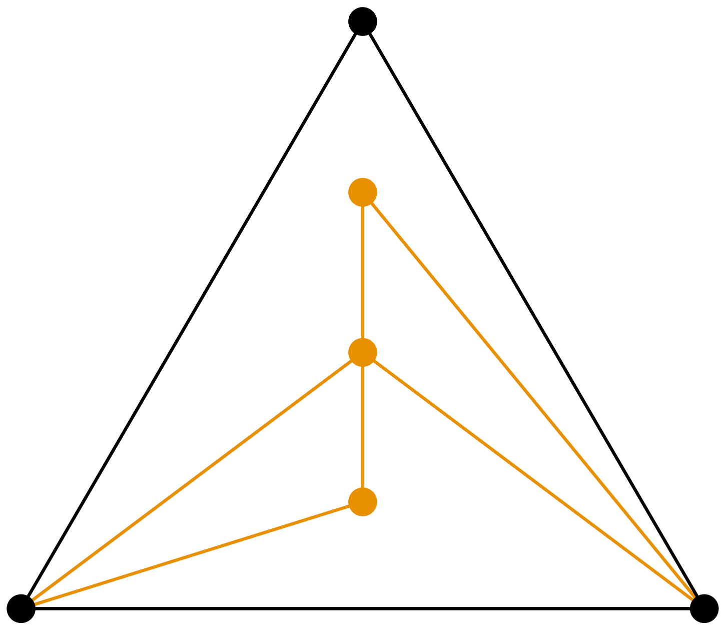
- $r = 2 + \varepsilon$



Adjacent and Nested Triangles

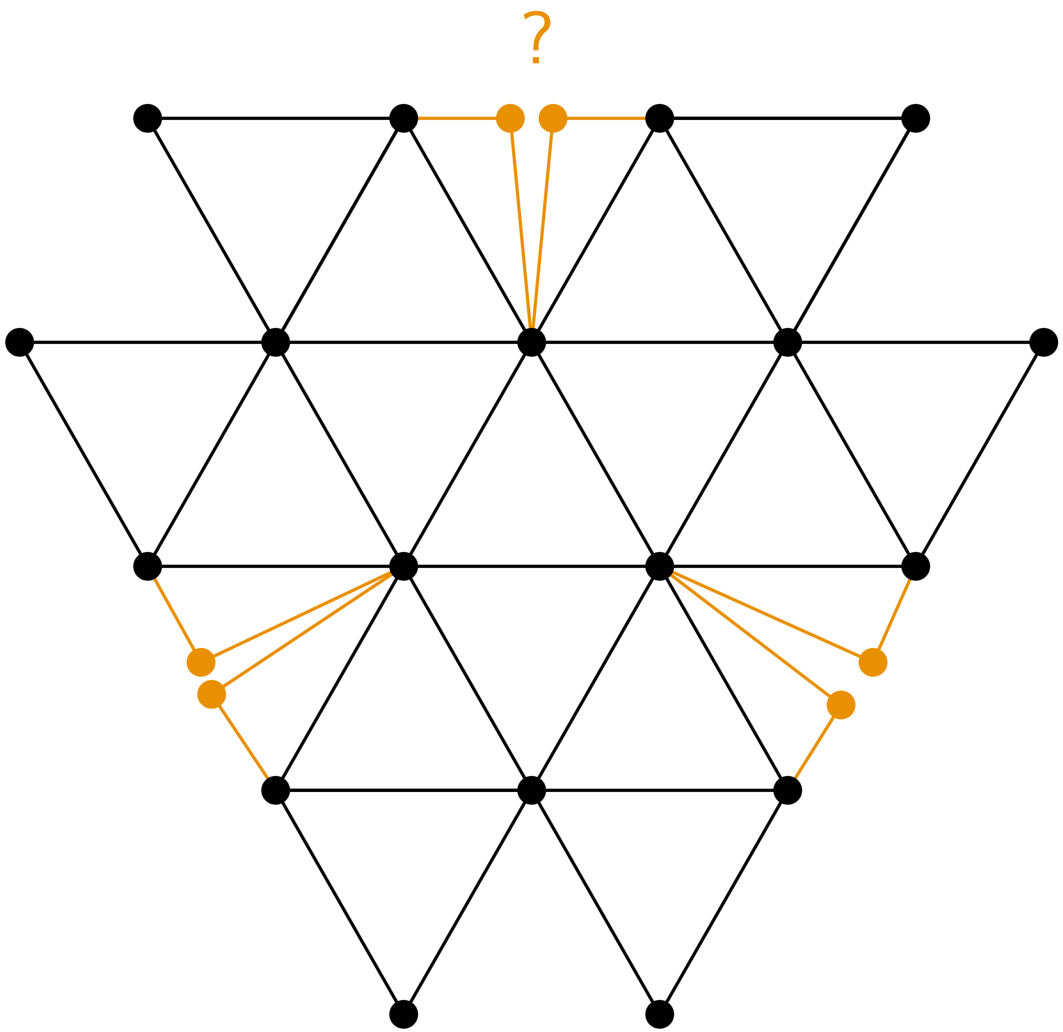
- $r = 2 + \varepsilon$
- $r = 4 + \varepsilon$

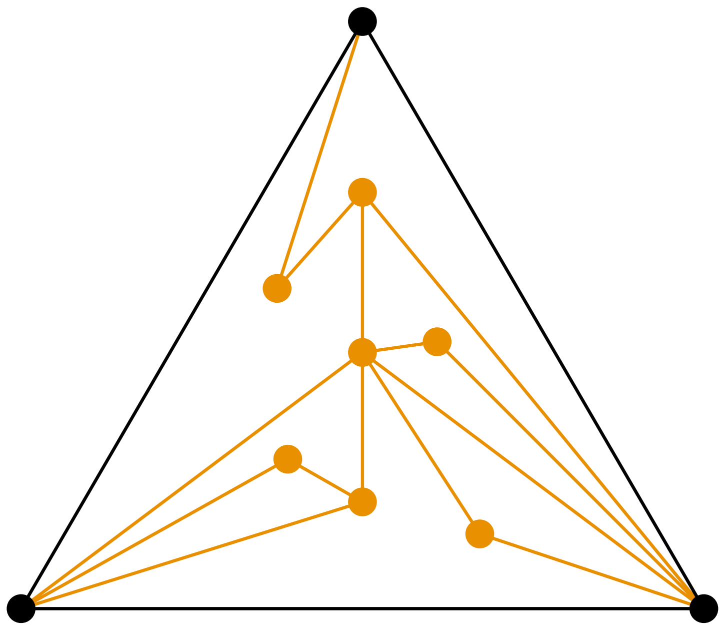




Adjacent and Nested Triangles

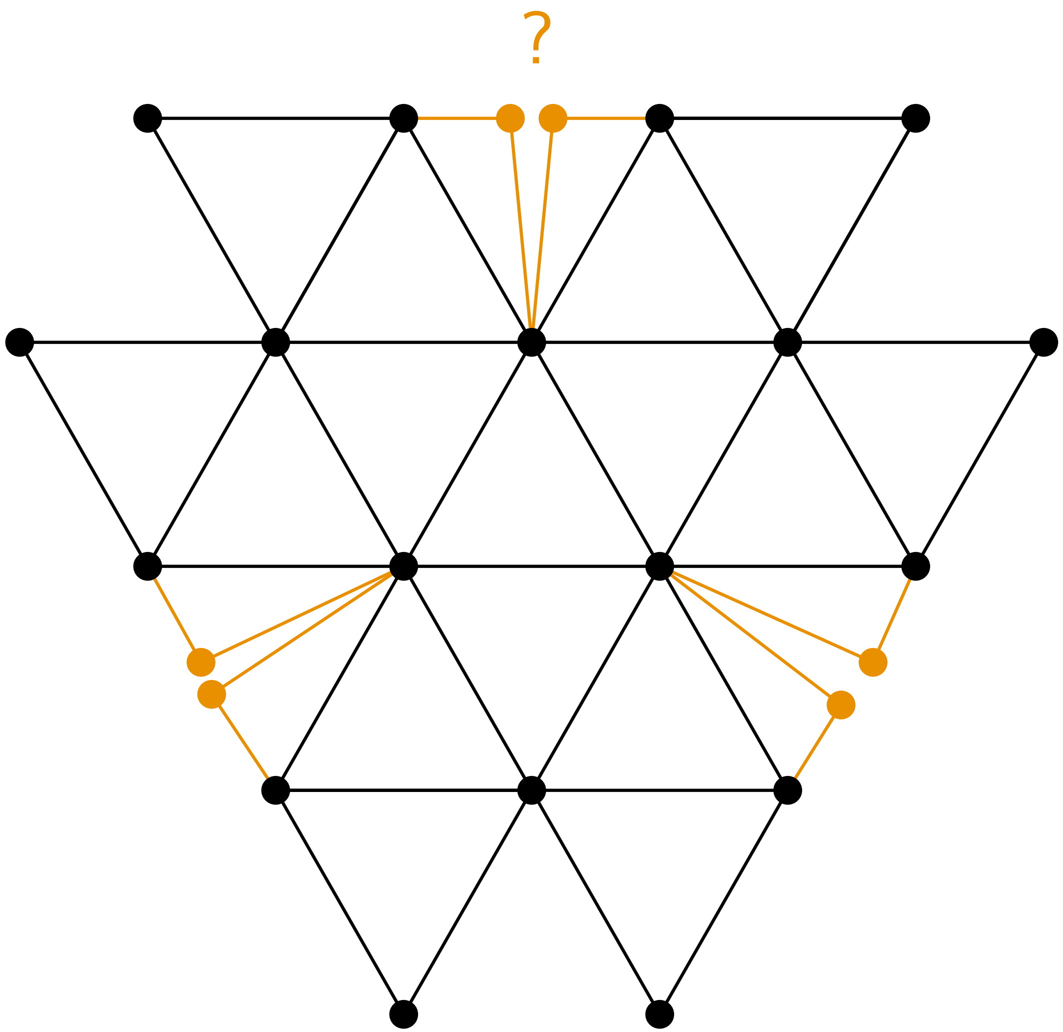
- $r = 2 + \varepsilon$
- $r = 8 + \varepsilon$



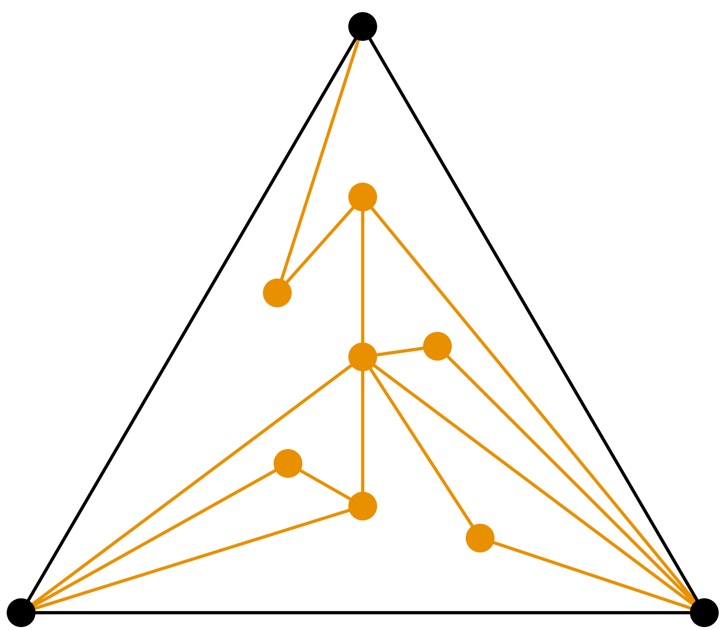


Adjacent and Nested Triangles

- $r = 2 + \varepsilon$
- *outerplanar graphs*

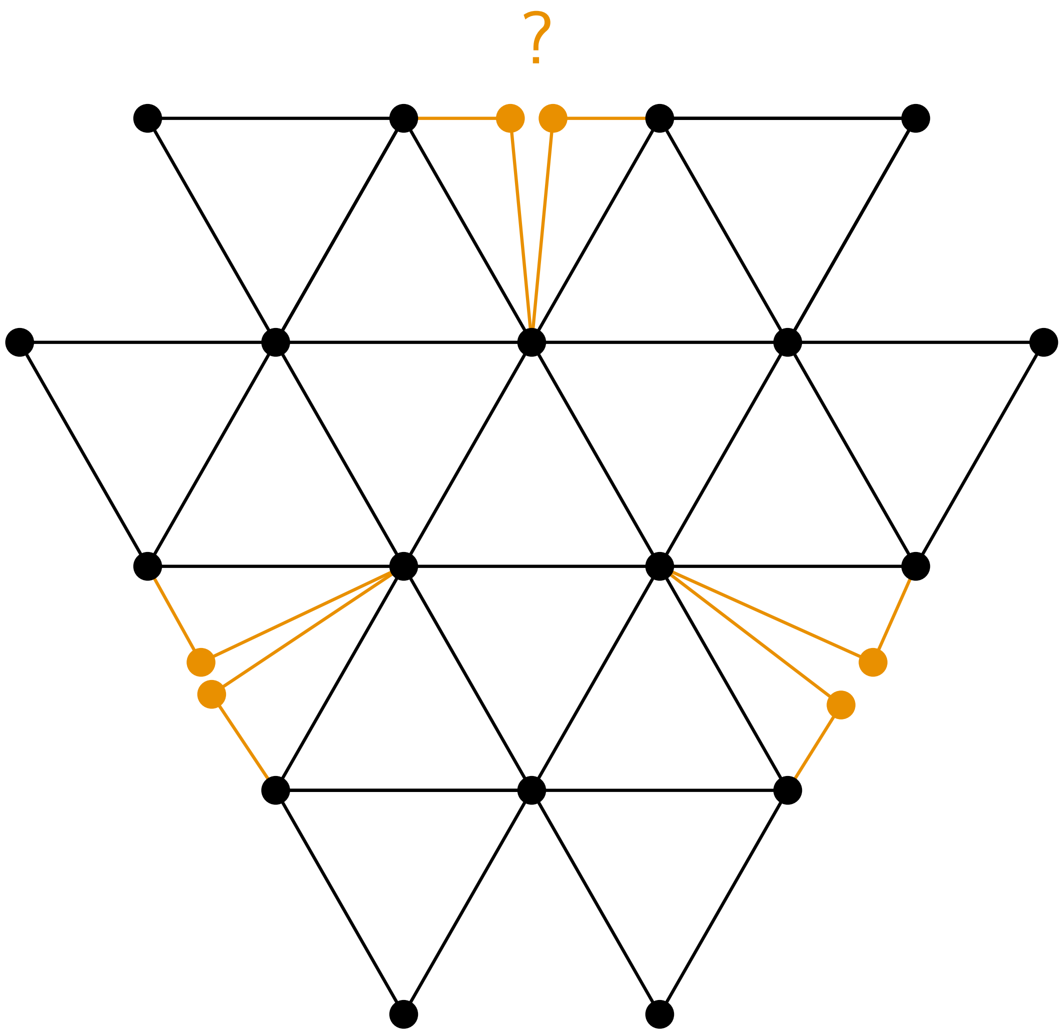


- $r = 8 + \varepsilon$
- series-parallel graphs

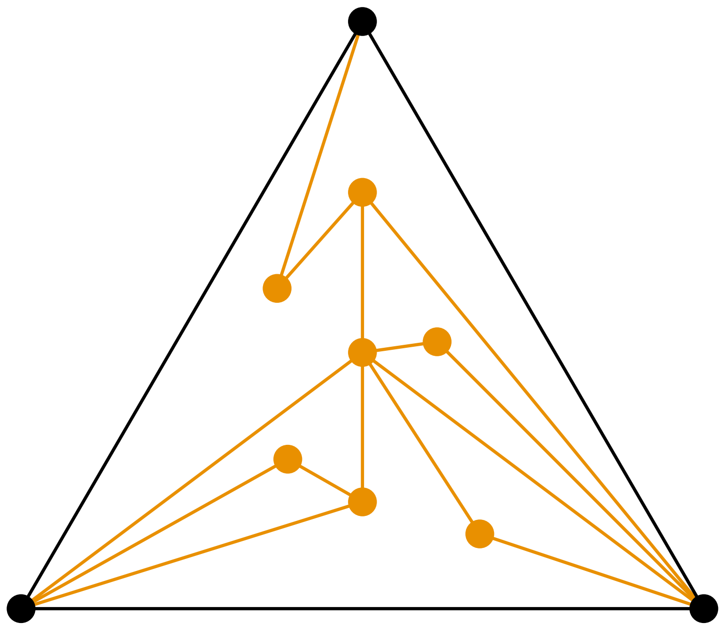


Adjacent and Nested Triangles

- $r = 2 + \varepsilon$
- *outerplanar graphs*



- $r = 8 + \varepsilon$
- series-parallel graphs



...logarithmic?

Outline

- Drawing Algorithm for Trees

Outline

- Drawing Algorithm for Trees
  - Complete k-ary Trees

Outline

- Drawing Algorithm for Trees
  - Complete k-ary Trees
  - General Trees

Outline

- Drawing Algorithm for Trees
  - Complete k-ary Trees
  - General Trees
- Drawing Algorithms for Outerplanar Graphs

Outline

- Drawing Algorithm for Trees
  - Complete k-ary Trees
  - General Trees
- Drawing Algorithms for Outerplanar Graphs
  - Approach I: Using Weak Dual Graph

Outline

- Drawing Algorithm for Trees
  - Complete k-ary Trees
  - General Trees
- Drawing Algorithms for Outerplanar Graphs
  - Approach I: Using Weak Dual Graph
  - Approach II: Using Tree Decomposition

Outline

- Drawing Algorithm for Trees
  - Complete k-ary Trees
  - General Trees
- Drawing Algorithms for Outerplanar Graphs
  - Approach I: Using Weak Dual Graph
  - Approach II: Using Tree Decomposition
- Drawing Algorithm for Series-Parallel Graphs

Outline

- Drawing Algorithm for Trees
  - Complete k-ary Trees
  - General Trees
- Drawing Algorithms for Outerplanar Graphs
  - Approach I: Using Weak Dual Graph
  - Approach II: Using Tree Decomposition
- Drawing Algorithm for Series-Parallel Graphs
- Appendix
  - Planar 3-trees
  - Implementation of Complete k-ary Tree Drawer

Trees

Trees

- connected and acyclic

Trees

- connected and acyclic
- *rooted* trees have a flagged vertex as *root*

Trees

- connected and acyclic
- *rooted* trees have a flagged vertex as *root*
  - any tree can be interpreted as rooted

Trees

- connected and acyclic
- *rooted* trees have a flagged vertex as *root*
  - any tree can be interpreted as rooted
  - path length from any vertex v to *root* defines its *height*

Trees

- connected and acyclic
- *rooted* trees have a flagged vertex as *root*
  - any tree can be interpreted as rooted
  - path length from any vertex v to *root* defines its *height*
  - The height h of a rooted tree is defined by vertex with greatest height

Trees

- connected and acyclic
- *rooted* trees have a flagged vertex as *root*
  - any tree can be interpreted as rooted
  - path length from any vertex v to *root* defines its *height*
  - The height h of a rooted tree is defined by vertex with greatest height
- k-ary tree is a rooted tree with at most k children per inner node

Trees

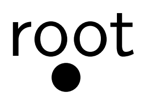
- connected and acyclic
- *rooted* trees have a flagged vertex as *root*
  - any tree can be interpreted as rooted
  - path length from any vertex v to *root* defines its *height*
  - The height h of a rooted tree is defined by vertex with greatest height
- k-ary tree is a rooted tree with at most k children per inner node
- k-ary tree is *complete* if every inner node has k children and all leaves are on the same height

Drawing A Complete k-ary Tree T

- Unidirectional from top to bottom

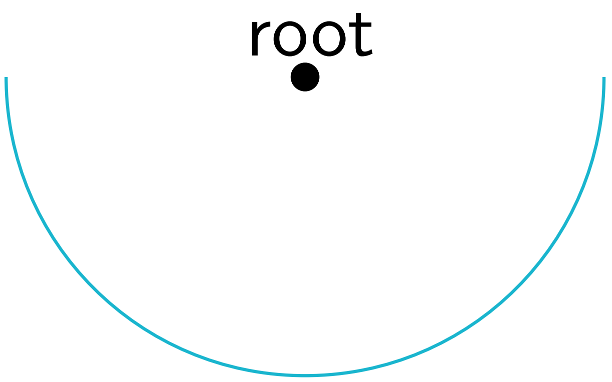
Drawing A Complete k-ary Tree T

- Unidirectional from top to bottom
- Place the root of T on the grid



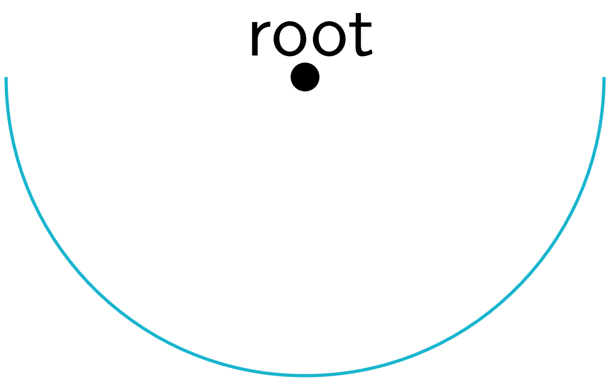
Drawing A Complete k-ary Tree T

- Unidirectional from top to bottom
- Place the root of T on the grid
- Draw circular arc C downwards



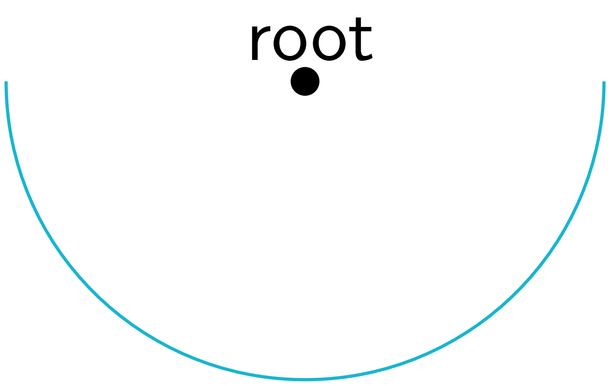
Drawing A Complete k-ary Tree T

- Unidirectional from top to bottom
- Place the root of T on the grid
- Draw circular arc C downwards
  - Radius of n



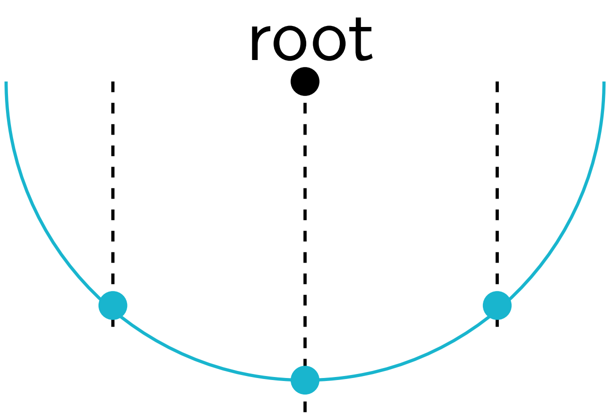
Drawing A Complete k-ary Tree T

- Unidirectional from top to bottom
- Place the root of T on the grid
- Draw circular arc C downwards
  - Radius of n
- Place k children on C equidistantly

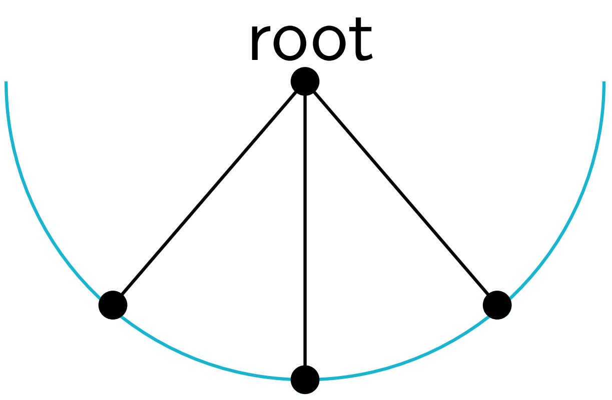


Drawing A Complete k-ary Tree T

- Unidirectional from top to bottom
- Place the root of T on the grid
- Draw circular arc C downwards
  - Radius of n
- Place k children on C equidistantly
  - x value distance: $2 \cdot k^{h-1}$



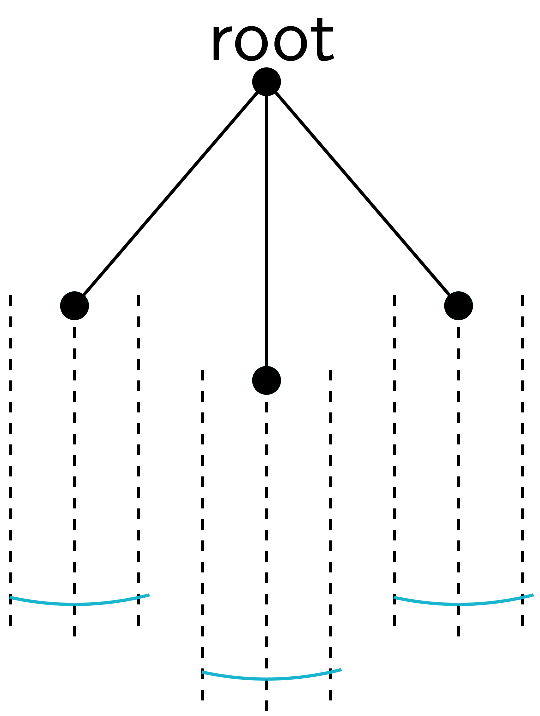
Drawing A Complete k-ary Tree T



- Unidirectional from top to bottom
- Place the root of T on the grid
- Draw circular arc C downwards
  - Radius of n
- Place k children on C equidistantly
  - x value distance: $2 \cdot k^{h-1}$
- Insert straight-lines

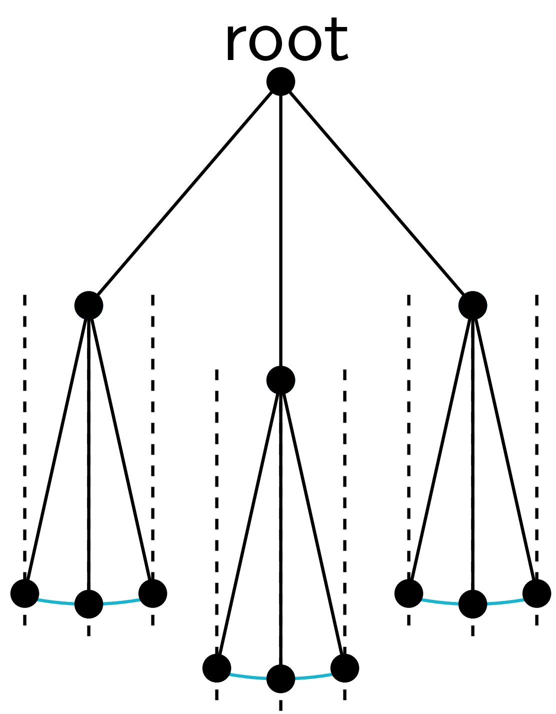
Drawing A Complete k-ary Tree T

- Unidirectional from top to bottom
- Place the root of T on the grid
- Draw circular arc C downwards
  - Radius of n
- Place k children on C equidistantly
  - x value distance: $2 \cdot k^{h-1}$
- Insert straight-lines
- Repeat for all vertices with height $h' < h$
  - x value distance: $2 \cdot k^{h-h'-1}$



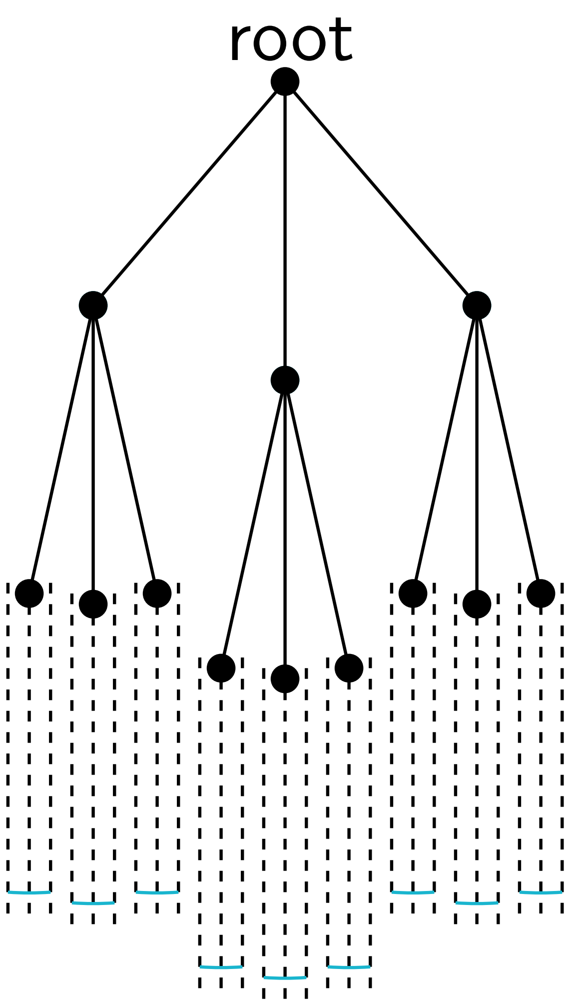
Drawing A Complete k-ary Tree T

- Unidirectional from top to bottom
- Place the root of T on the grid
- Draw circular arc C downwards
  - Radius of n
- Place k children on C equidistantly
  - x value distance: $2 \cdot k^{h-1}$
- Insert straight-lines
- Repeat for all vertices with height $h' < h$
  - x value distance: $2 \cdot k^{h-h'-1}$
  - Radius for circular arcs still n



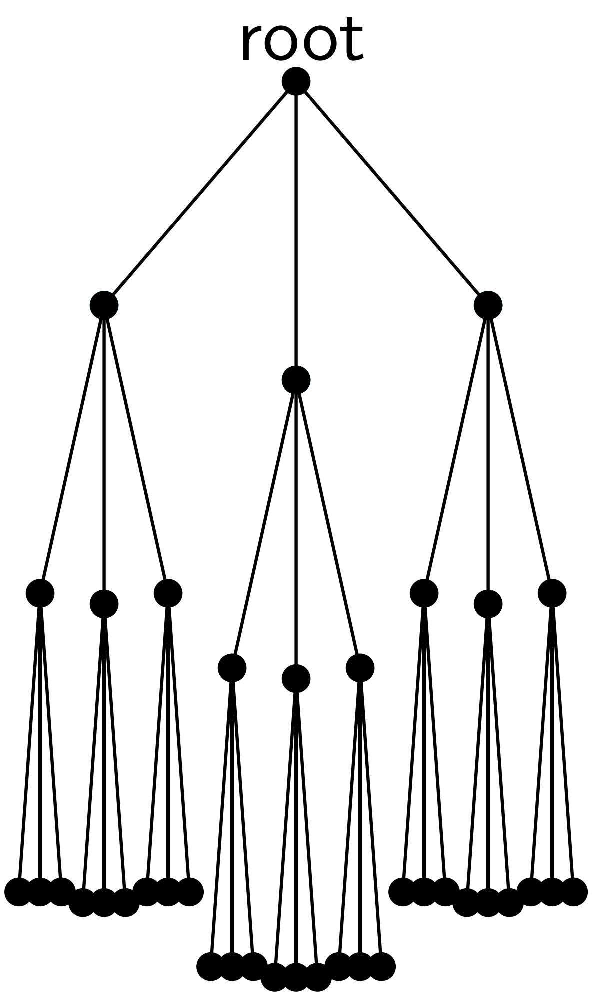
Drawing A Complete k-ary Tree T

- Unidirectional from top to bottom
- Place the root of T on the grid
- Draw circular arc C downwards
  - Radius of n
- Place k children on C equidistantly
  - x value distance: $2 \cdot k^{h-1}$
- Insert straight-lines
- Repeat for all vertices with height $h' < h$
  - x value distance: $2 \cdot k^{h-h'-1}$
  - Radius for circular arcs still n



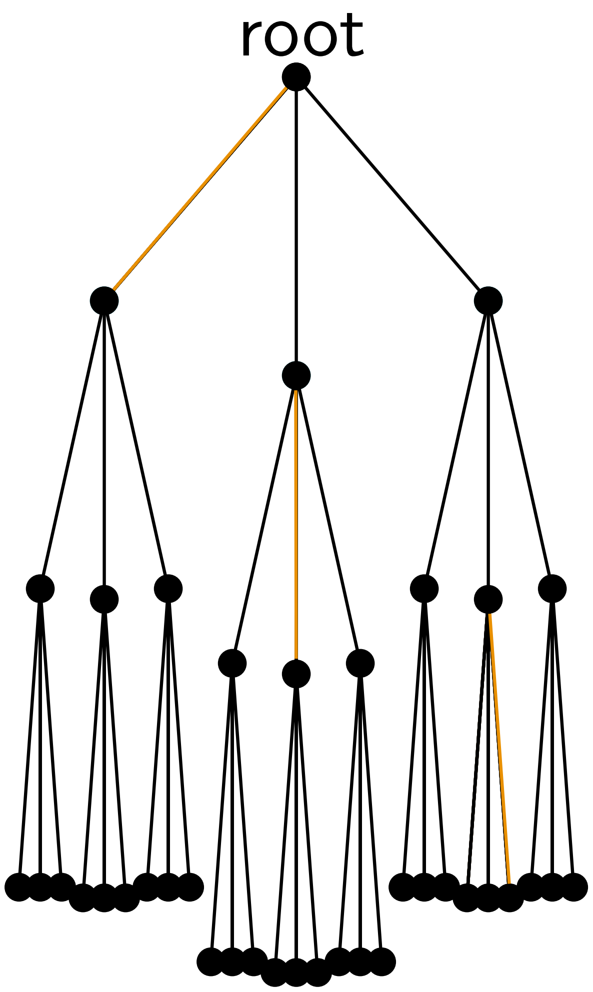
Drawing A Complete k-ary Tree T

- Unidirectional from top to bottom
- Place the root of T on the grid
- Draw circular arc C downwards
  - Radius of n
- Place k children on C equidistantly
  - x value distance: $2 \cdot k^{h-1}$
- Insert straight-lines
- Repeat for all vertices with height $h' < h$
  - x value distance: $2 \cdot k^{h-h'-1}$
  - Radius for circular arcs still n



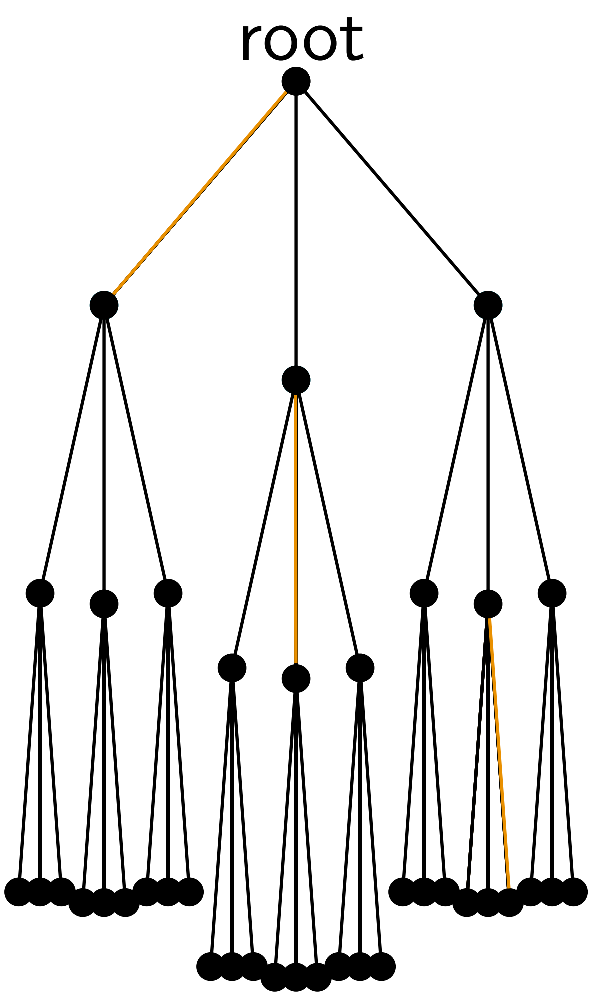
Results: Complete k-ary Trees

- Every straight-line is $\sim n$ long



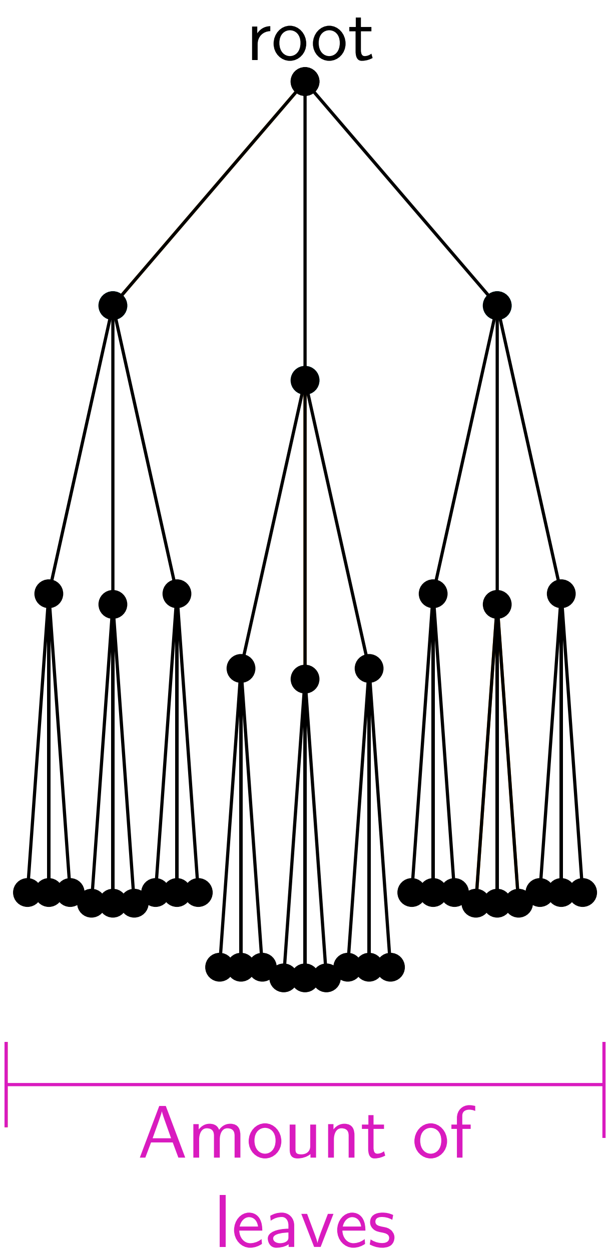
Results: Complete *k*-ary Trees

- Every straight-line is $\sim n$ long
  - Ratio of drawings value $1 + \varepsilon$, $0 \leq \varepsilon < 1$



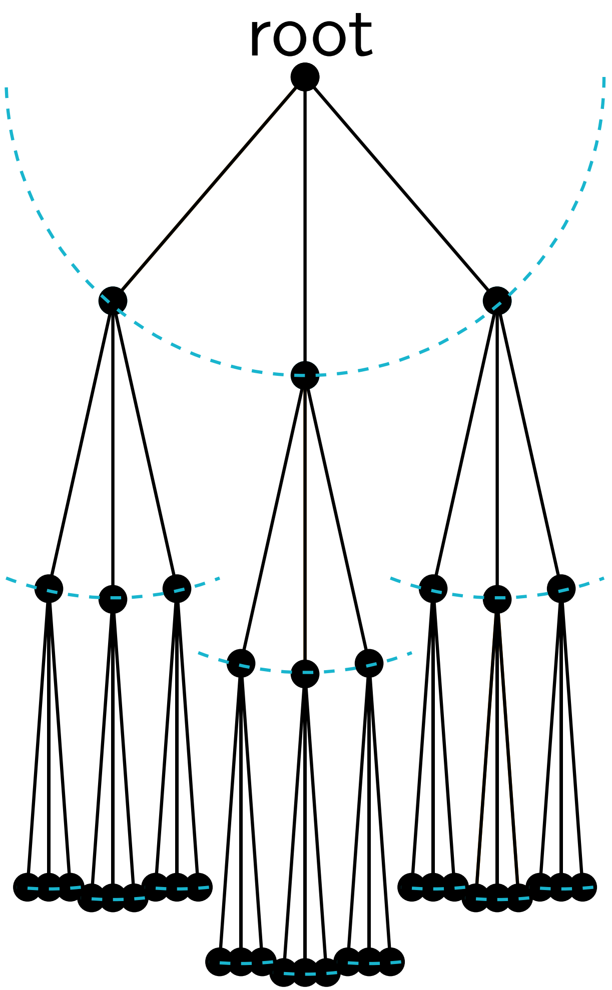
Results: Complete k-ary Trees

- Every straight-line is $\sim n$ long
  - Ratio of drawings value $1 + \varepsilon$, $0 \leq \varepsilon < 1$
- Area width in $\mathcal{O}(n)$



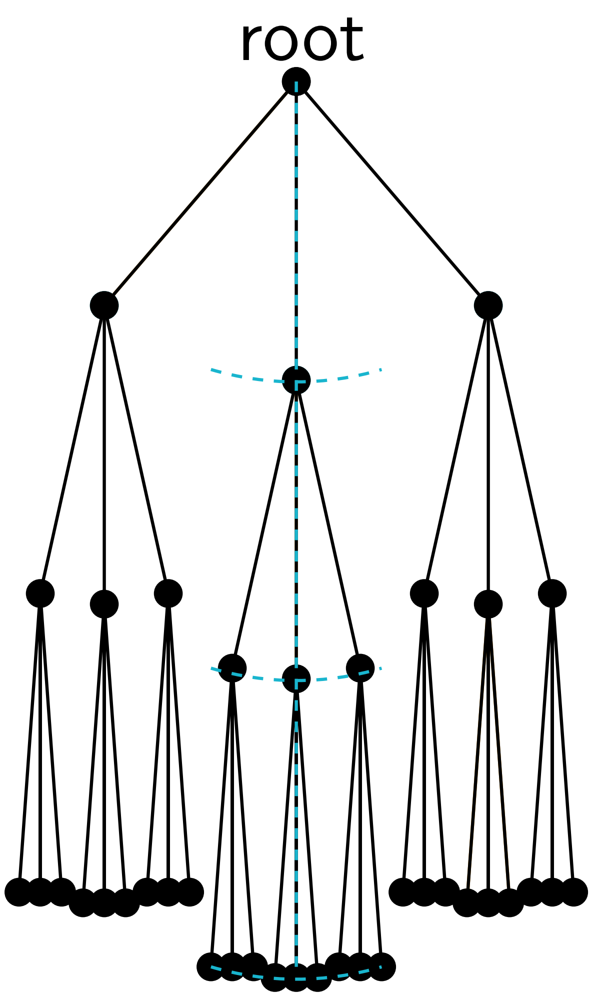
Results: Complete k-ary Trees

- Every straight-line is $\sim n$ long
  - Ratio of drawings value $1 + \varepsilon$, $0 \leq \varepsilon < 1$
- Area width in $\mathcal{O}(n)$
- Height of rooted tree T determines
  - Radius of circular arcs: n



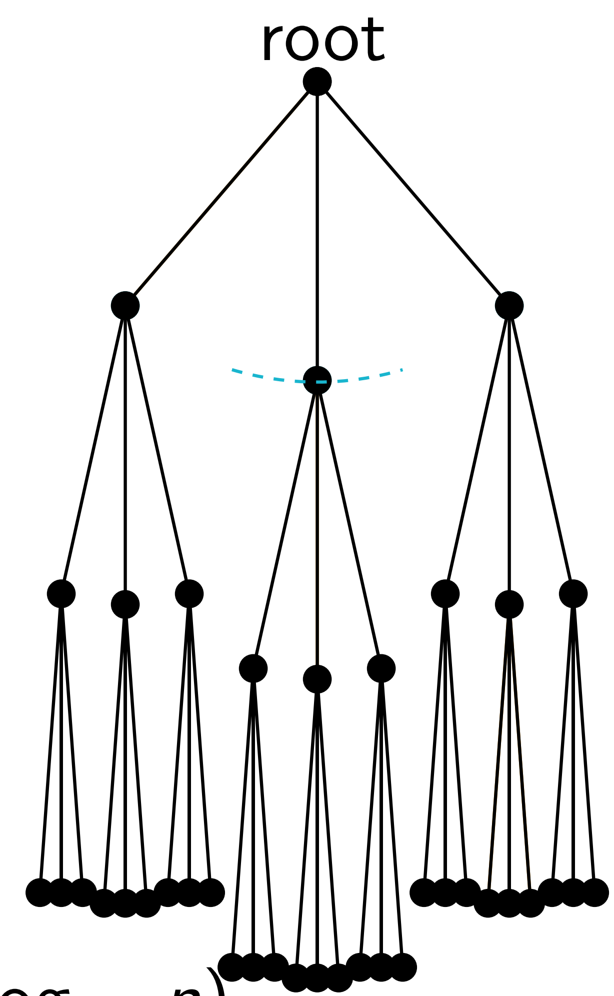
Results: Complete k-ary Trees

- Every straight-line is $\sim n$ long
  - Ratio of drawings value $1 + \varepsilon$, $0 \leq \varepsilon < 1$
- Area width in $\mathcal{O}(n)$
- Height of rooted tree T determines
  - Radius of circular arcs: n
  - Height of the drawing: $h \cdot n$



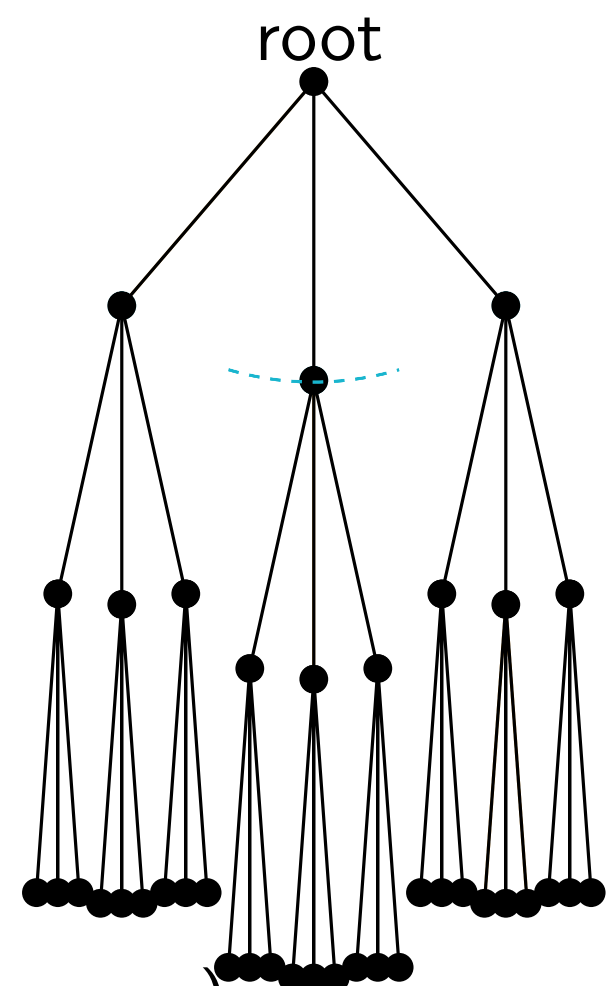
Results: Complete k-ary Trees

- Every straight-line is $\sim n$ long
  - Ratio of drawings value $1 + \varepsilon$, $0 \leq \varepsilon < 1$
- Area width in $\mathcal{O}(n)$
- Height of rooted tree T determines
  - Radius of circular arcs: n
  - Height of the drawing: $h \cdot n$
- Complete k-ary tree with height $\mathcal{O}(\log_{(k)} n)$



Results: Complete k-ary Trees

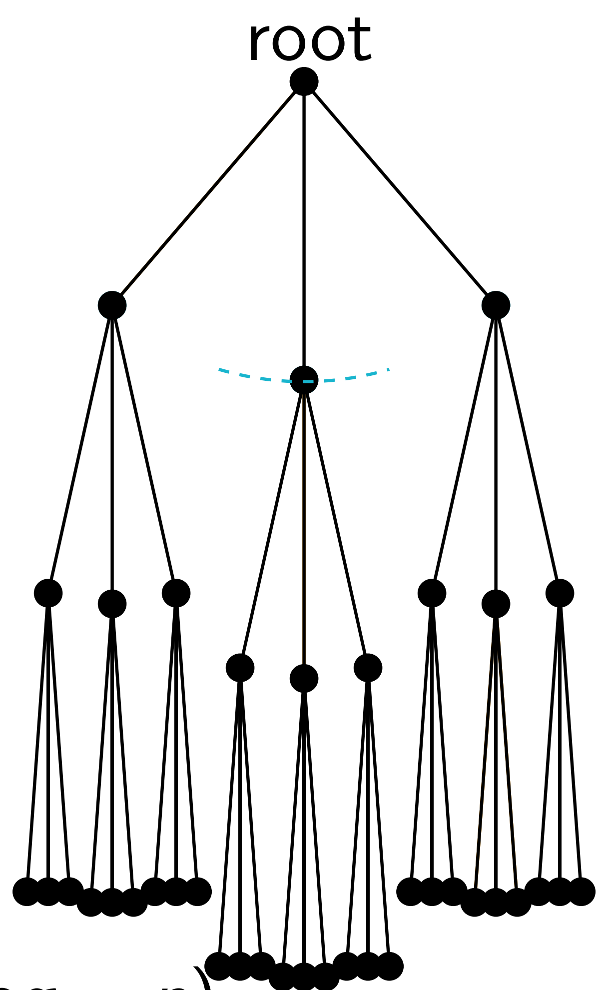
- Every straight-line is $\sim n$ long
  - Ratio of drawings value $1 + \varepsilon$, $0 \leq \varepsilon < 1$
- Area width in $\mathcal{O}(n)$
- Height of rooted tree T determines
  - Radius of circular arcs: n
  - Height of the drawing: $h \cdot n$



- Complete k-ary tree with height $\mathcal{O}(\log_{(k)} n)$ admit a straight-line drawing with nearly-optimal ratio on area $\mathcal{O}(n^2 \log n)$

Results: Complete k-ary Trees

- Every straight-line is $\sim n$ long
  - Ratio of drawings value $1 + \varepsilon$, $0 \leq \varepsilon < 1$
- Area width in $\mathcal{O}(n)$
- Height of rooted tree T determines
  - Radius of circular arcs: n
  - Height of the drawing: $h \cdot n$



- Complete k-ary tree with height $\mathcal{O}(\log_{(k)} n)$ admit a straight-line drawing with nearly-optimal ratio on area $\mathcal{O}(n^2 \log n)$

Results: General Trees

- Consider height h between $\Omega(\log n)$ and up to $\mathcal{O}(n)$

Results: General Trees

- Consider height h between $\Omega(\log n)$ and up to $\mathcal{O}(n)$

Results: General Trees

- Consider height h between $\Omega(\log n)$ and up to $\mathcal{O}(n)$

Results: General Trees

- Consider height h between $\Omega(\log n)$ and up to $\mathcal{O}(n)$

Results: General Trees

- Consider height h between $\Omega(\log n)$ and up to $\mathcal{O}(n)$

Results: General Trees

- Consider height h between $\Omega(\log n)$ and up to $\mathcal{O}(n)$

Results: General Trees

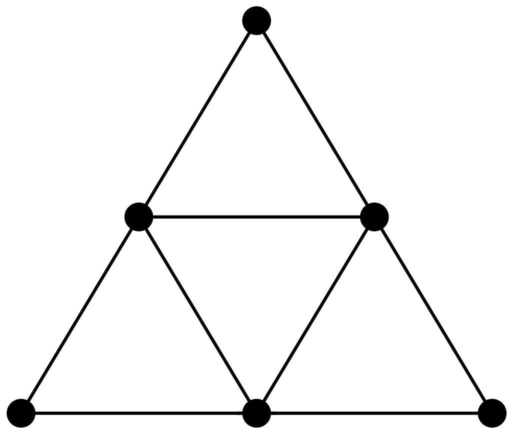
- Consider height h between $\Omega(\log n)$ and up to $\mathcal{O}(n)$

Outerplanar Graphs and Series-Parallel Graphs

Outerplanar Graphs

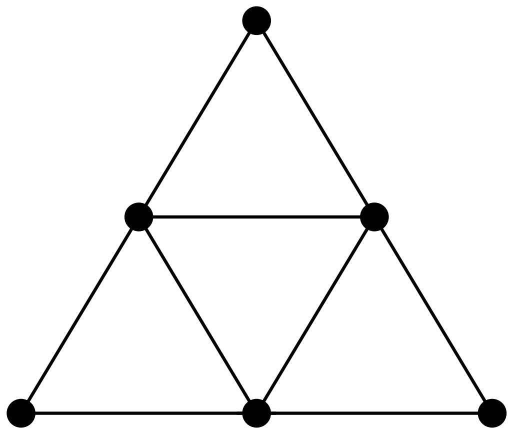
Outerplanar Graphs

- G admitting a drawing where every vertex lies on the outer face is *outerplanar*



Outerplanar Graphs

- G admitting a drawing where every vertex lies on the outer face is *outerplanar*



- outerplanar graph *maximal* if no edges can be inserted without destroying outerplanarity

2-Terminal Series-Parallel Graphs

- Recursively defined class of planar graphs

2-Terminal Series-Parallel Graphs

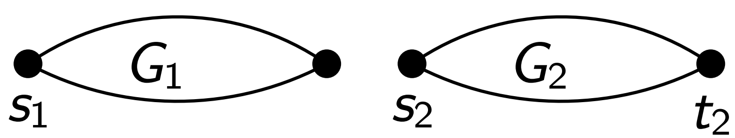
- Recursively defined class of planar graphs
- *2-terminal series-parallel* graph with *terminals* s, t

2-Terminal Series-Parallel Graphs

- Recursively defined class of planar graphs
- *2-terminal series-parallel* graph with *terminals* s, t
  - Base case: Edge (s, t) is a 2-terminal SP-graph

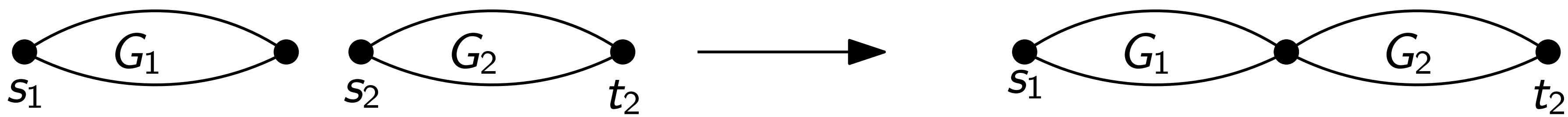
2-Terminal Series-Parallel Graphs

- Recursively defined class of planar graphs
- *2-terminal series-parallel* graph with *terminals* s, t
  - Base case: Edge (s, t) is a 2-terminal SP-graph
  - Serial case: Composition of two 2-terminal SP-graphs is a 2-terminal SP-graph



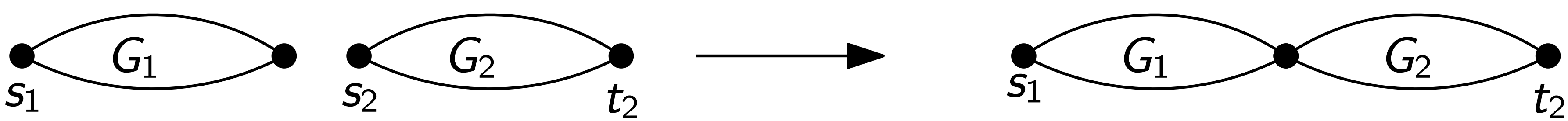
2-Terminal Series-Parallel Graphs

- Recursively defined class of planar graphs
- *2-terminal series-parallel* graph with *terminals* s, t
  - Base case: Edge (s, t) is a 2-terminal SP-graph
  - Serial case: Composition of two 2-terminal SP-graphs is a 2-terminal SP-graph

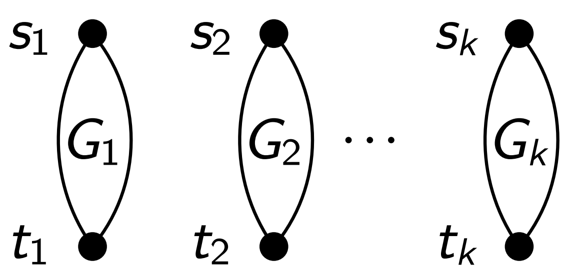


2-Terminal Series-Parallel Graphs

- Recursively defined class of planar graphs
- *2-terminal series-parallel* graph with *terminals* s, t
  - Base case: Edge (s, t) is a 2-terminal SP-graph
  - Serial case: Composition of two 2-terminal SP-graphs is a 2-terminal SP-graph

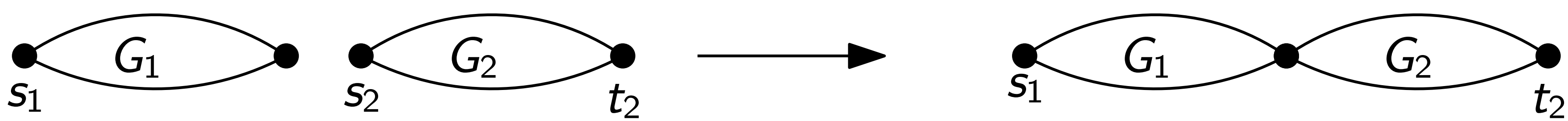


  - Parallel case: Composition of k 2-terminal SP-graphs is a 2-terminal SP-graph



2-Terminal Series-Parallel Graphs

- Recursively defined class of planar graphs
- *2-terminal series-parallel* graph with *terminals* s, t
  - Base case: Edge (s, t) is a 2-terminal SP-graph
  - Serial case: Composition of two 2-terminal SP-graphs is a 2-terminal SP-graph



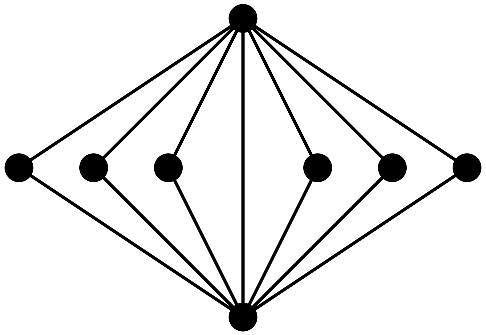
  - Parallel case: Composition of k 2-terminal SP-graphs is a 2-terminal SP-graph

Series-Parallel Graph

- a *series-parallel* graph has 2-terminal SP-graphs as biconnected components

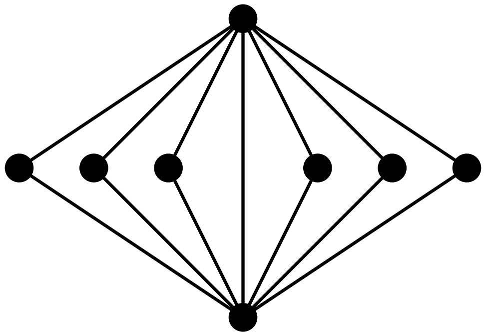
Series-Parallel Graph

- a *series-parallel* graph has 2-terminal SP-graphs as biconnected components



Series-Parallel Graph

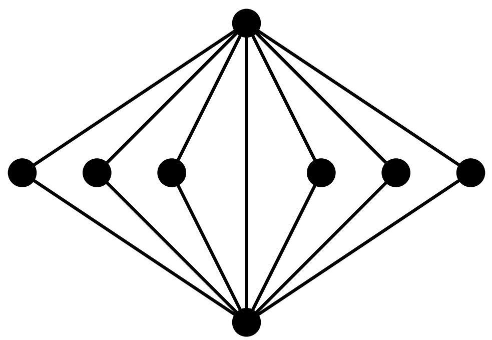
- a *series-parallel* graph has 2-terminal SP-graphs as biconnected components



- a series-parallel graph is *maximal* if no edges can be inserted while maintaining a series-parallel graph

Series-Parallel Graph

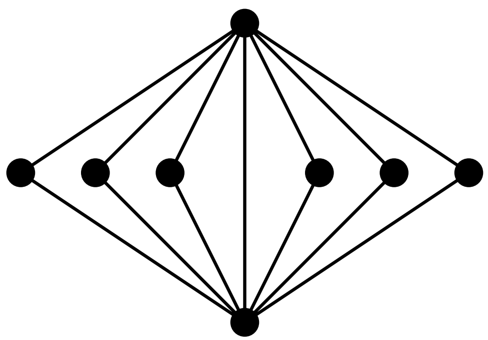
- a *series-parallel* graph has 2-terminal SP-graphs as biconnected components



- a series-parallel graph is *maximal* if no edges can be inserted while maintaining a series-parallel graph
- maximal outerplanar graphs are maximal SP-graphs

Series-Parallel Graph

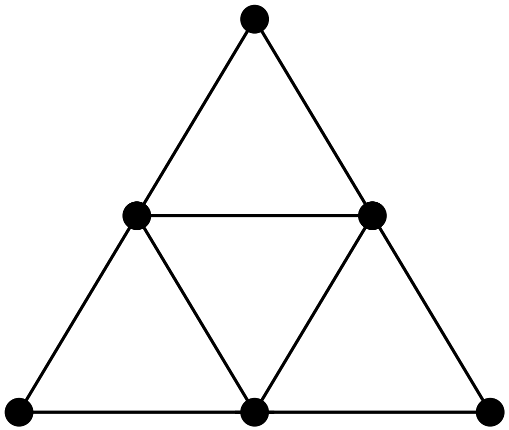
- a *series-parallel* graph has 2-terminal SP-graphs as biconnected components



- a series-parallel graph is *maximal* if no edges can be inserted while maintaining a series-parallel graph
- maximal outerplanar graphs are maximal SP-graphs
  - Drawing approaches deal with outerplanar graphs first
  - Extended to SP-graphs, if possible

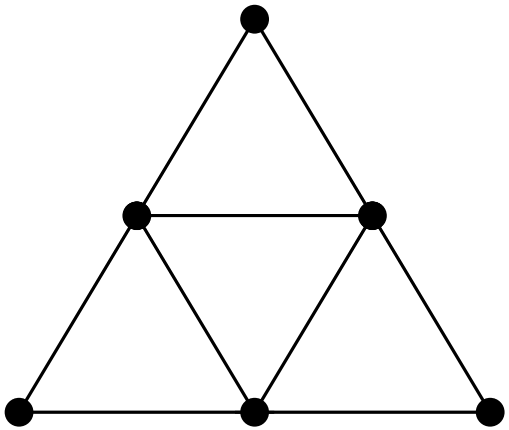
Approach I: Using Weak Dual Graph

- outerplanar graph *maximal* if no edges can be inserted without destroying outerplanarity



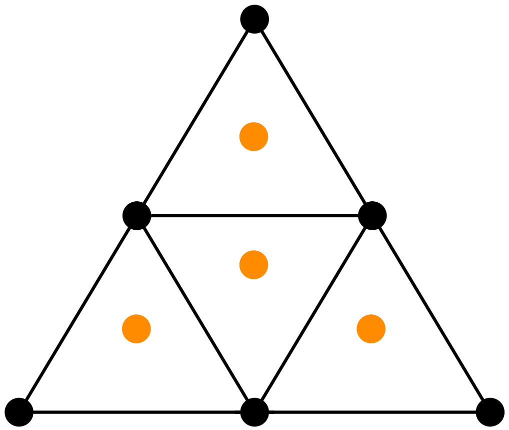
Approach I: Using Weak Dual Graph

- outerplanar graph *maximal* if no edges can be inserted without destroying outerplanarity



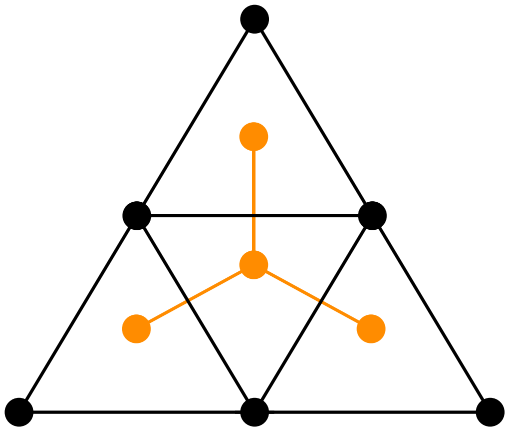
Approach I: Using Weak Dual Graph

- outerplanar graph *maximal* if no edges can be inserted without destroying outerplanarity



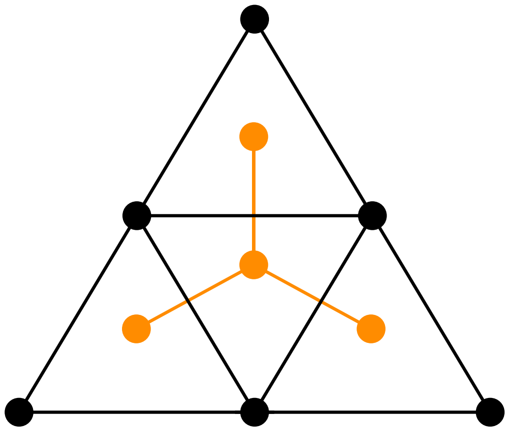
Approach I: Using Weak Dual Graph

- outerplanar graph *maximal* if no edges can be inserted without destroying outerplanarity



Approach I: Using Weak Dual Graph

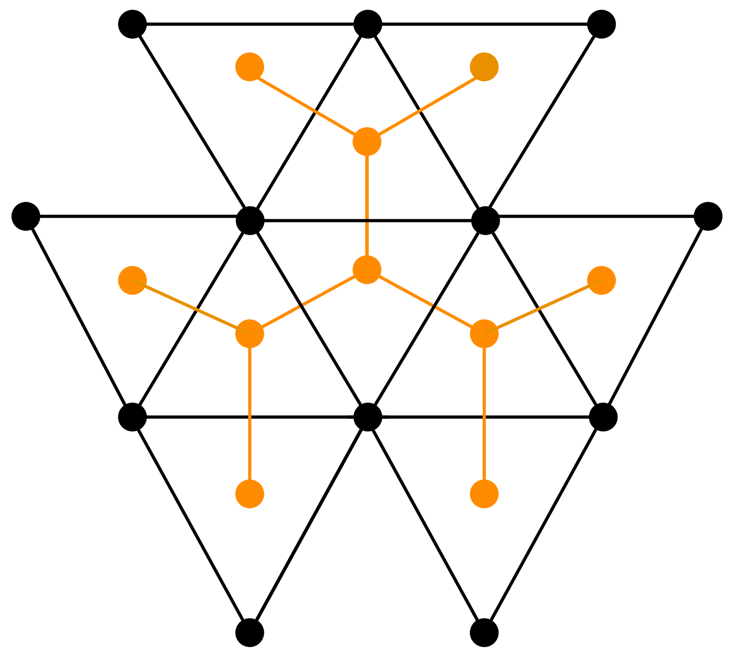
- outerplanar graph *maximal* if no edges can be inserted without destroying outerplanarity



- G^* for maximal outerplanar graph is *simple tree*

Approach I: Using Weak Dual Graph

- outerplanar graph *maximal* if no edges can be inserted without destroying outerplanarity



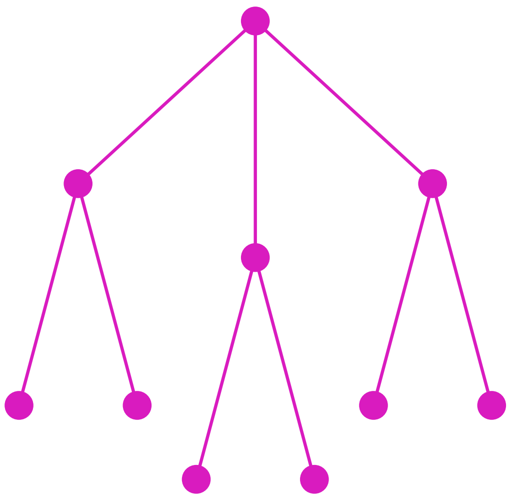
- G^* for maximal outerplanar graph is *simple tree*
- Degree of G^* is at most 3

Approach I: Using Weak Dual Graph

- Make G maximal

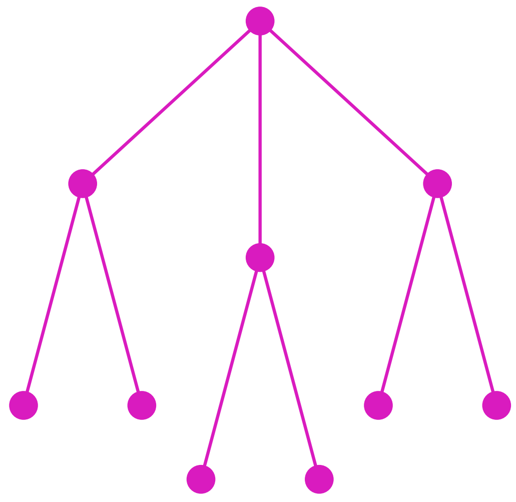
Approach I: Using Weak Dual Graph

- Make G maximal
- Draw G^*



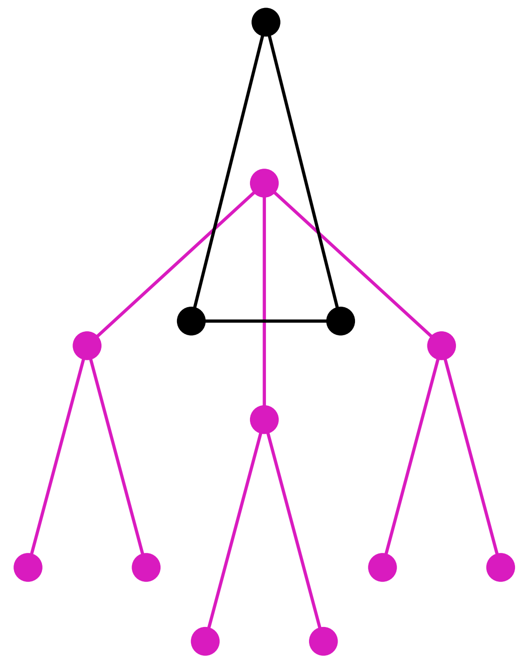
Approach I: Using Weak Dual Graph

- Make G maximal
- Draw G^*
- Draw triangles at each vertex of Γ_{G^*}



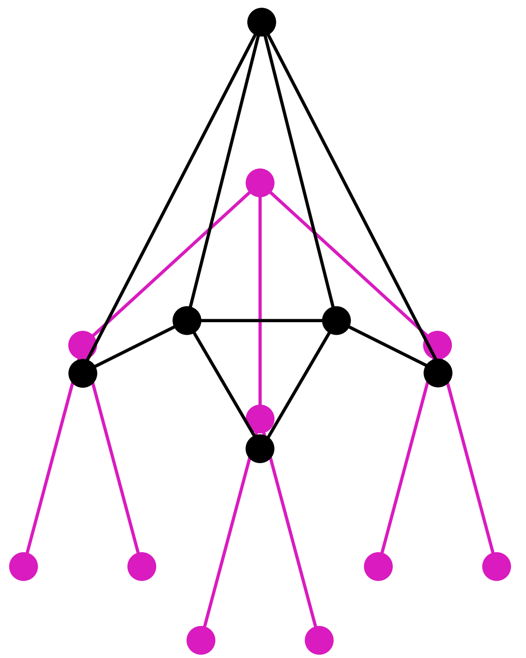
Approach I: Using Weak Dual Graph

- Make G maximal
- Draw G^*
- Draw triangles at each vertex of Γ_{G^*}



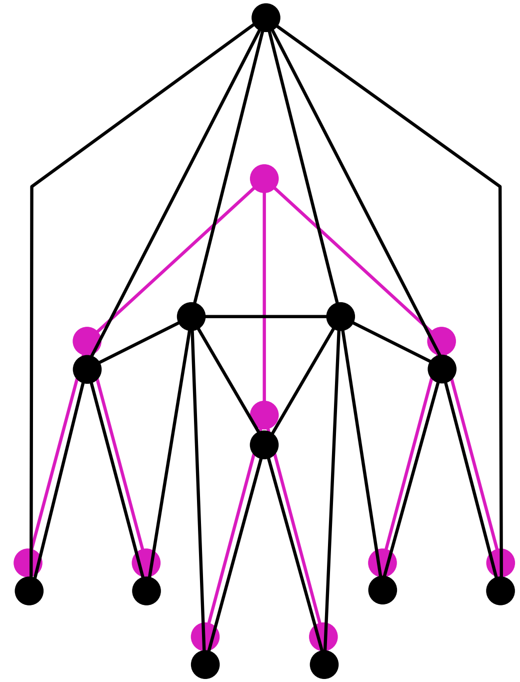
Approach I: Using Weak Dual Graph

- Make G maximal
- Draw G^*
- Draw triangles at each vertex of Γ_{G^*}



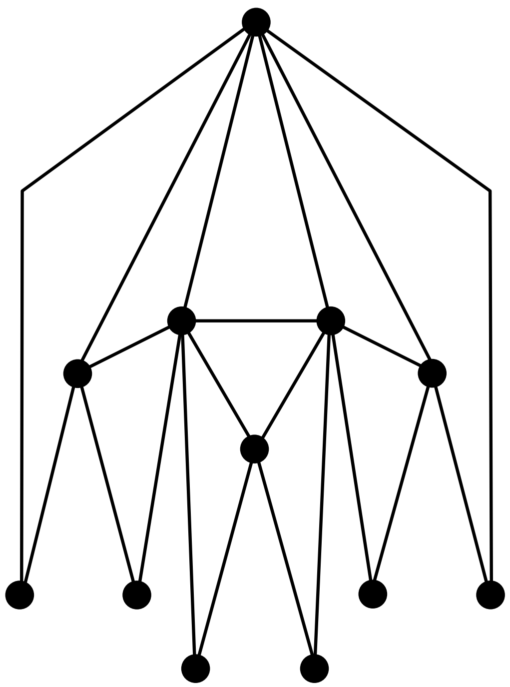
Approach I: Using Weak Dual Graph

- Make G maximal
- Draw G^*
- Draw triangles at each vertex of Γ_{G^*}



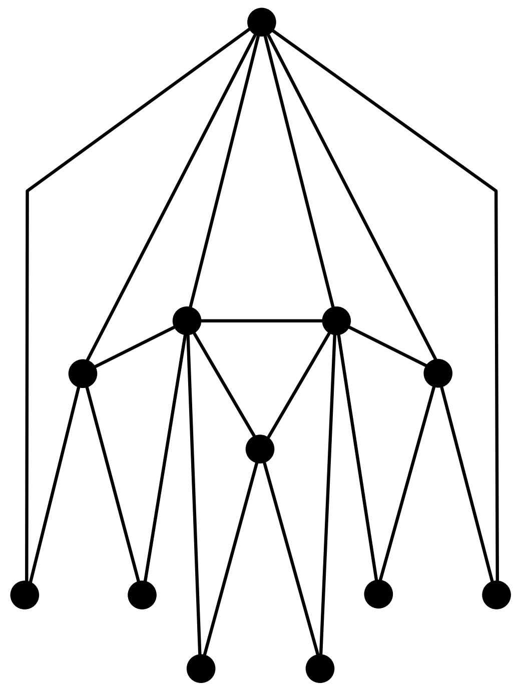
Approach I: Using Weak Dual Graph

- Make G maximal
- Draw G^*
- Draw triangles at each vertex of Γ_{G^*}
- Remove Γ_{G^*}



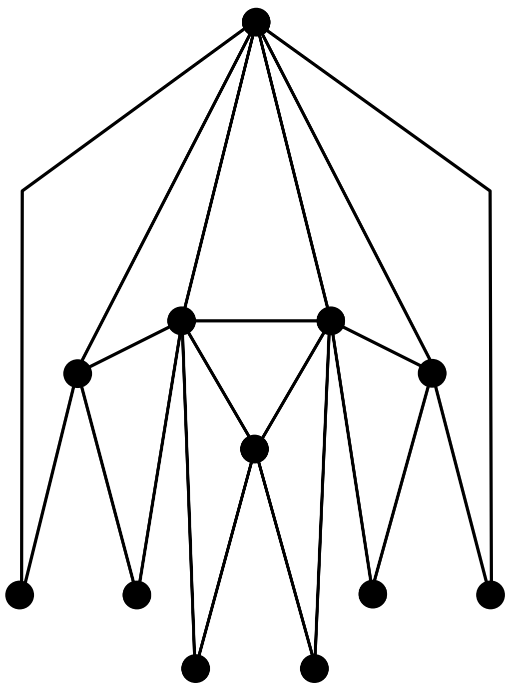
Approach I: Using Weak Dual Graph

- Make G maximal
- Draw G^*
- Draw triangles at each vertex of Γ_{G^*}
- Remove Γ_{G^*}
- Shortest Euclidian distance equals radius used in tree drawing



Approach I: Using Weak Dual Graph

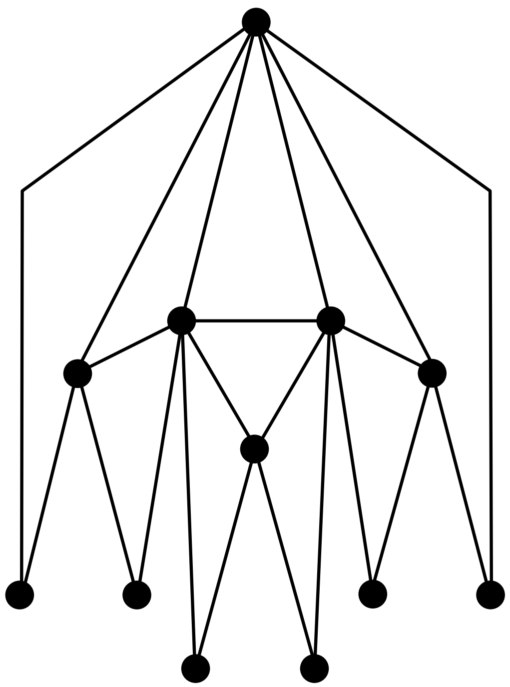
- Make G maximal
- Draw G^*
- Draw triangles at each vertex of Γ_{G^*}
- Remove Γ_{G^*}
- Shortest Euclidian distance equals radius used in tree drawing
- Length of longest polyline defined by height of drawing



Approach I: Using Weak Dual Graph

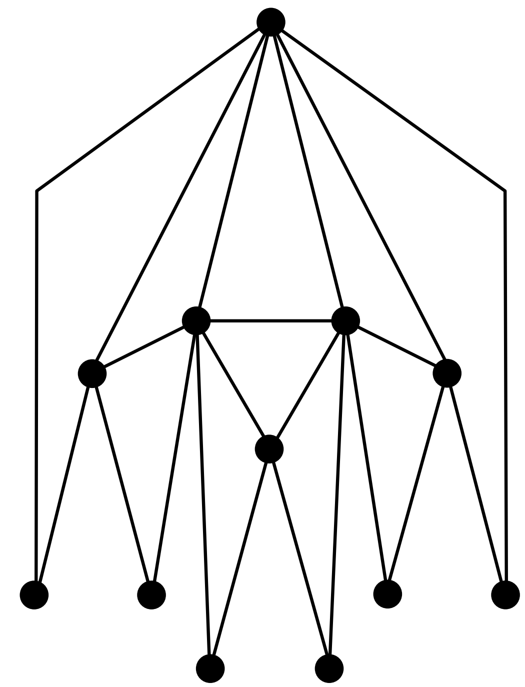
- Make G maximal
- Draw G^*
- Draw triangles at each vertex of Γ_{G^*}
- Remove Γ_{G^*}
- Shortest Euclidian distance equals radius used in tree drawing
- Length of longest polyline defined by height of drawing

- If height of G^* in $\mathcal{O}(\log n)$



Approach I: Using Weak Dual Graph

- Make G maximal
- Draw G^*
- Draw triangles at each vertex of Γ_{G^*}
- Remove Γ_{G^*}
- Shortest Euclidian distance equals radius used in tree drawing
- Length of longest polyline defined by height of drawing



- If height of G^* in $\mathcal{O}(\log n)$
  - Radius in $\Theta(n)$, drawing height in $\mathcal{O}(n \log n)$
  - ratio in $\mathcal{O}(\log n)$ on area $\mathcal{O}(n^2 \log n)$, 1 bend per edge

Approach I: Limitations

- Ratio of drawing depends of height of G^*

Approach I: Limitations

- Ratio of drawing depends of height of G^*
  - ratio lies in $\mathcal{O}\left(\frac{\text{drawing height}}{\text{radius}}\right) = \mathcal{O}\left(\frac{h \cdot \text{radius}}{\text{radius}}\right) = \mathcal{O}(h)$

Approach I: Limitations

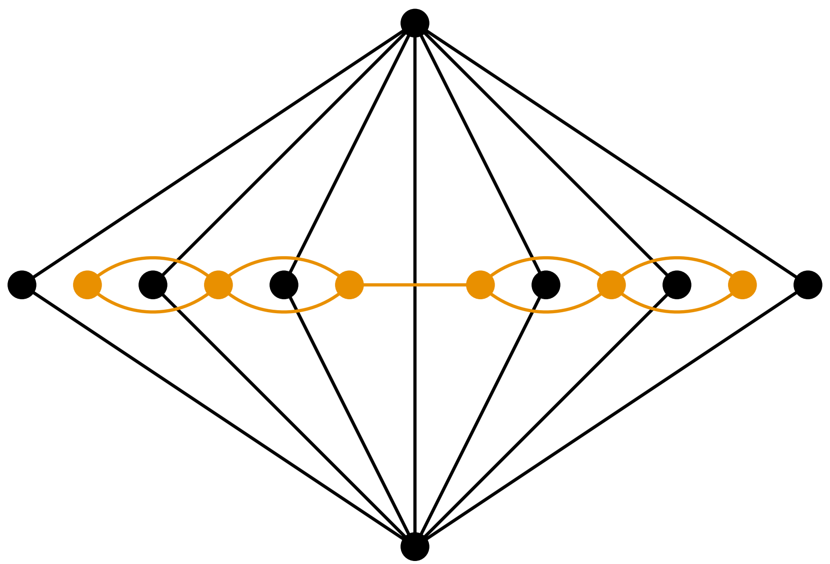
- Ratio of drawing depends of height of G^*
  - ratio lies in $\mathcal{O}\left(\frac{\text{drawing height}}{\text{radius}}\right) = \mathcal{O}\left(\frac{h \cdot \text{radius}}{\text{radius}}\right) = \mathcal{O}(h)$
  - If height of G^* is in $\mathcal{O}(n)$, so is the ratio

Approach I: Limitations

- Ratio of drawing depends of height of G^*
  - ratio lies in $\mathcal{O}\left(\frac{\text{drawing height}}{\text{radius}}\right) = \mathcal{O}\left(\frac{h \cdot \text{radius}}{\text{radius}}\right) = \mathcal{O}(h)$
  - If height of G^* is in $\mathcal{O}(n)$, so is the ratio
- Not applicable for Series-Parallel Graphs

Approach I: Limitations

- Ratio of drawing depends of height of G^*
  - ratio lies in $\mathcal{O}\left(\frac{\text{drawing height}}{\text{radius}}\right) = \mathcal{O}\left(\frac{h \cdot \text{radius}}{\text{radius}}\right) = \mathcal{O}(h)$
  - If height of G^* is in $\mathcal{O}(n)$, so is the ratio
- Not applicable for Series-Parallel Graphs
  - Weak dual graph can be a multigraph



Approach II: Using tree decomposition

- Tree decomposition is a tool to evaluate the "tree-like" structure of a graph G

Approach II: Using tree decomposition

- Tree decomposition is a tool to evaluate the "tree-like" structure of a graph G
- (T, W)

Approach II: Using tree decomposition

- Tree decomposition is a tool to evaluate the "tree-like" structure of a graph G
- (T, W)
  - T tree

Approach II: Using tree decomposition

- Tree decomposition is a tool to evaluate the “tree-like“ structure of a graph G
- (T, W)
  - T tree
  - W family of sets of vertices, called *bags*

Approach II: Using tree decomposition

- Tree decomposition is a tool to evaluate the "tree-like" structure of a graph G
- (T, W)
  - T tree
  - W family of sets of vertices, called *bags*
  - For every $t_i \in T$ there is a bag $w_i \in W$

Approach II: Using tree decomposition

- Tree decomposition is a tool to evaluate the "tree-like" structure of a graph G
- (T, W)
  - T tree
  - W family of sets of vertices, called *bags*
  - For every $t_i \in T$ there is a bag $w_i \in W$
  - All vertices of G lie in in at least one bag

Approach II: Using tree decomposition

- Tree decomposition is a tool to evaluate the "tree-like" structure of a graph G
- (T, W)
  - T tree
  - W family of sets of vertices, called *bags*
  - For every $t_i \in T$ there is a bag $w_i \in W$
  - All vertices of G lie in in at least one bag
  - For $(u, v) \in E(G)$, there is a bag that inherits u and v

Approach II: Using tree decomposition

- Tree decomposition is a tool to evaluate the "tree-like" structure of a graph G
- (T, W)
  - T tree
  - W family of sets of vertices, called *bags*
  - For every $t_i \in T$ there is a bag $w_i \in W$
  - All vertices of G lie in in at least one bag
  - For $(u, v) \in E(G)$, there is a bag that inherits u and v
  - $\texttt{Query}(v, T) = T'_v$, T'_v connected subtree of T

Approach II: Using tree decomposition

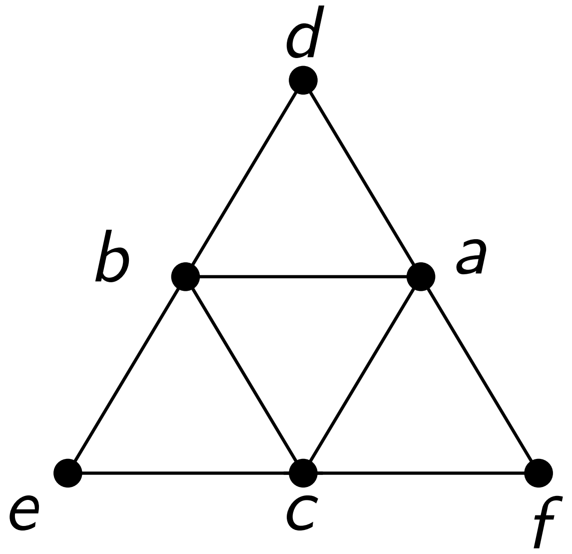
- Tree decomposition is a tool to evaluate the "tree-like" structure of a graph G
- (T, W)
  - T tree
  - W family of sets of vertices, called *bags*
  - For every $t_i \in T$ there is a bag $w_i \in W$
  - All vertices of G lie in in at least one bag
  - For $(u, v) \in E(G)$, there is a bag that inherits u and v
  - $\texttt{Query}(v, T) = T'_v$, T'_v connected subtree of T
- $\textit{width } w(T, W) = \max\{|w_t|, t \in T\} - 1$

Approach II: Using tree decomposition

- Tree decomposition is a tool to evaluate the "tree-like" structure of a graph G
- (T, W)
  - T tree
  - W family of sets of vertices, called *bags*
  - For every $t_i \in T$ there is a bag $w_i \in W$
  - All vertices of G lie in in at least one bag
  - For $(u, v) \in E(G)$, there is a bag that inherits u and v
  - $\texttt{Query}(v, T) = T'_v$, T'_v connected subtree of T
- *width* $w(T, W) = \max\{|w_t|, t \in T\} - 1$
- *treewidth* is least width of any tree decomposition of G

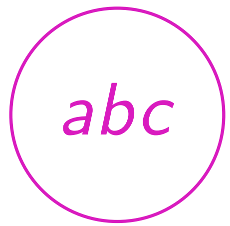
Tree Decomposition Of SP-Graphs

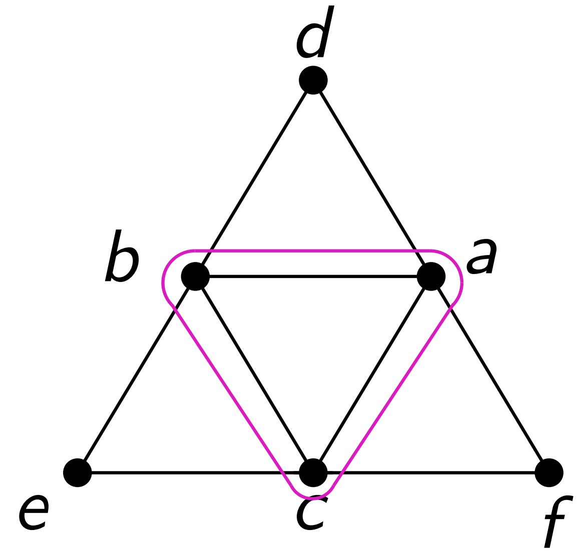
- (T, W)



Tree Decomposition Of SP-Graphs

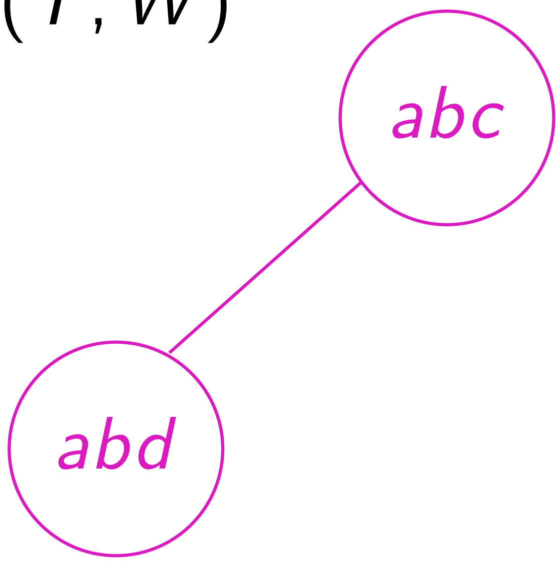
- (T, W)

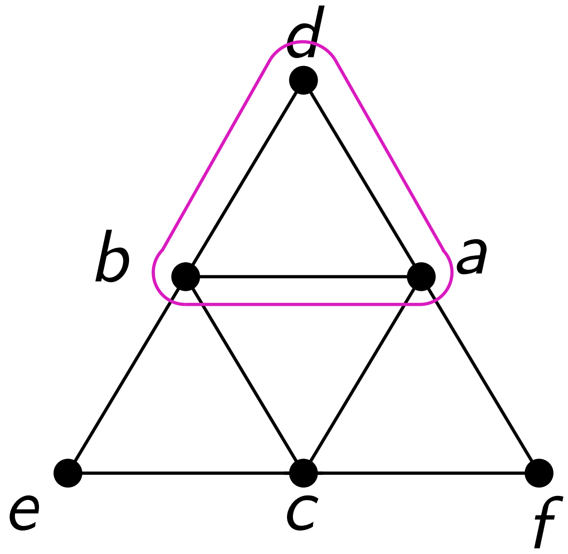




Tree Decomposition Of SP-Graphs

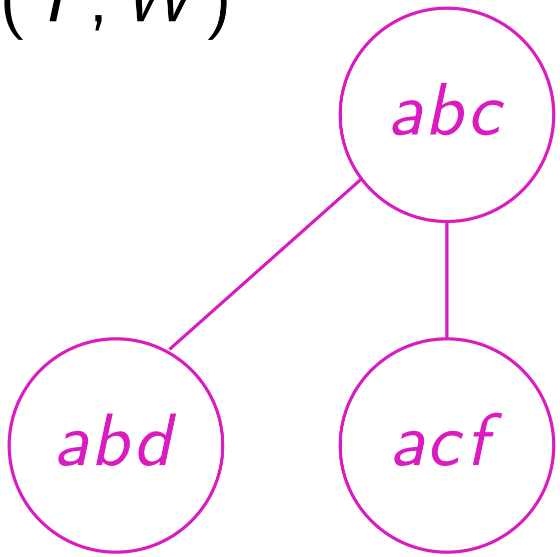
- (T, W)

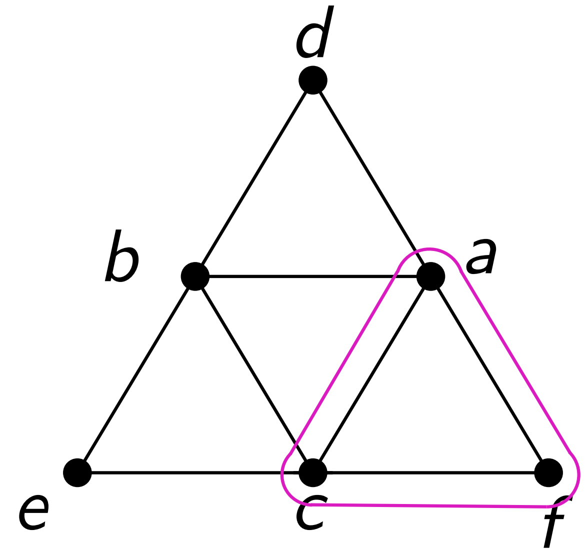




Tree Decomposition Of SP-Graphs

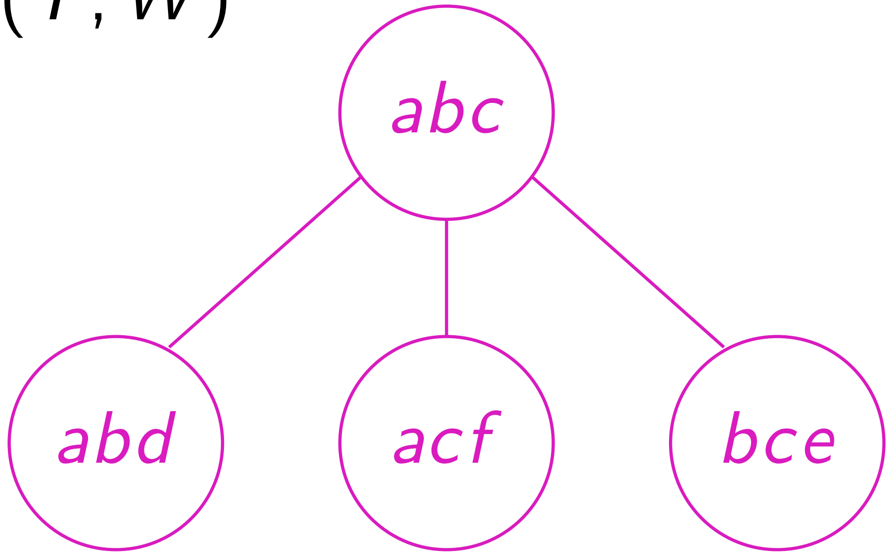
- (T, W)

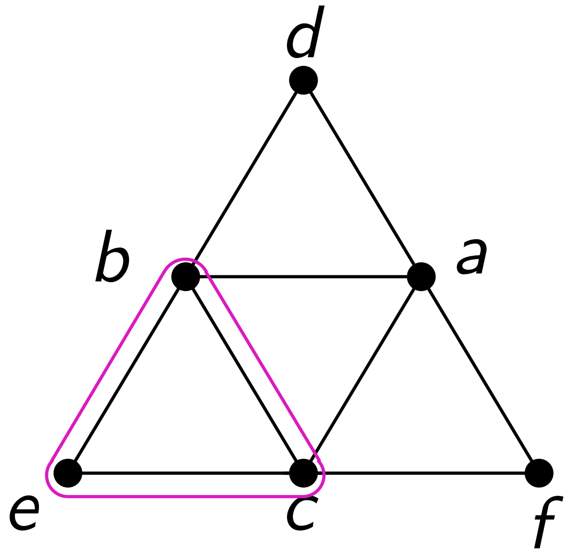




Tree Decomposition Of SP-Graphs

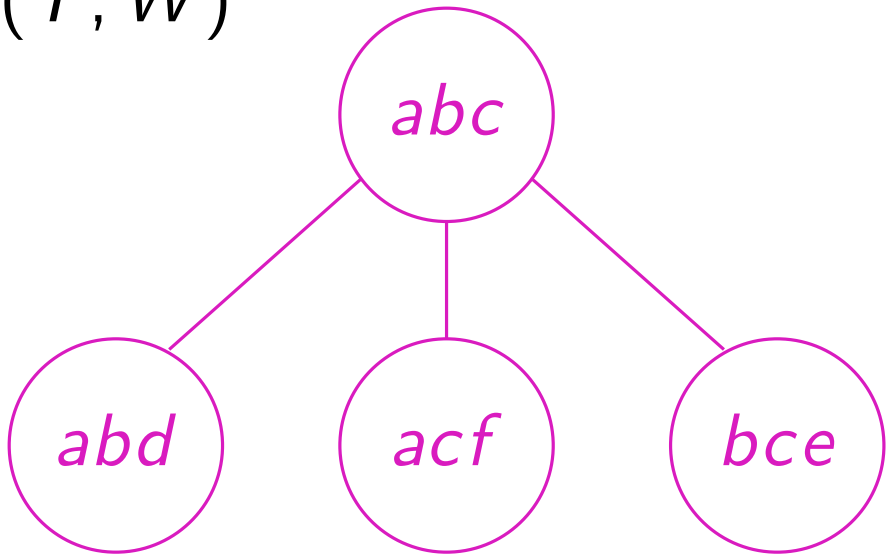
- (T, W)

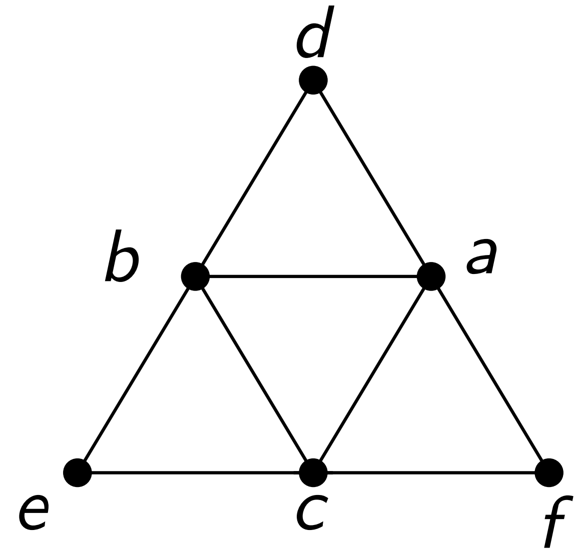




Tree Decomposition Of SP-Graphs

- (T, W)

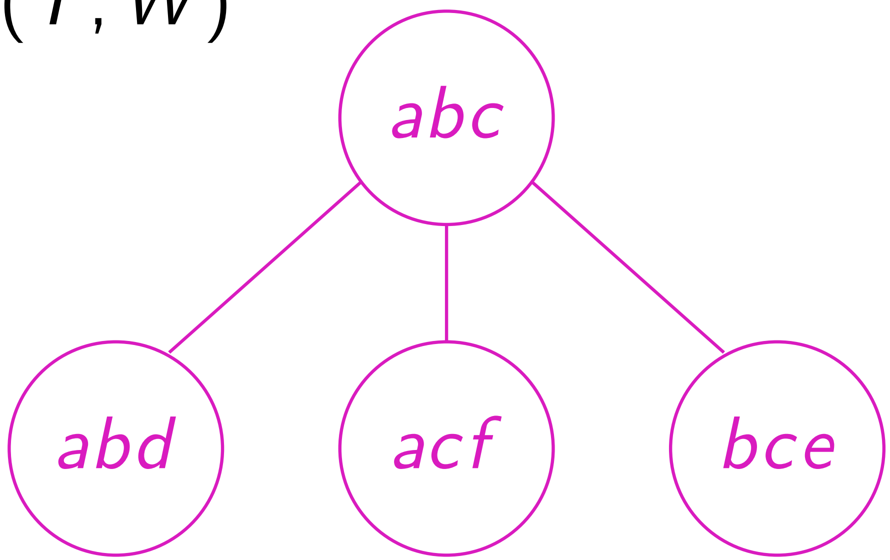


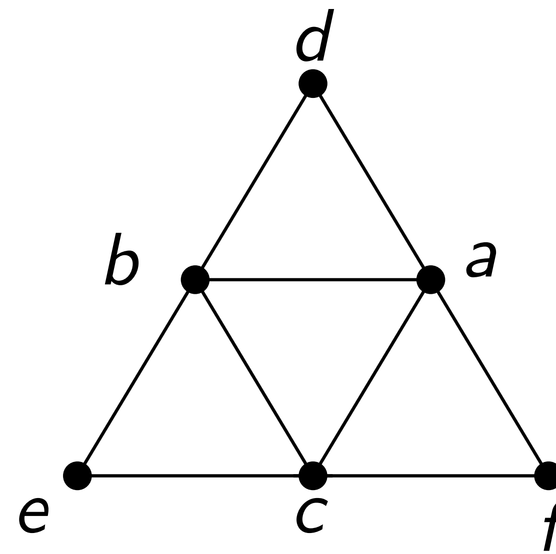


- treewidth of 2 for any maximal series-parallel graph

Tree Decomposition Of SP-Graphs

- (T, W)

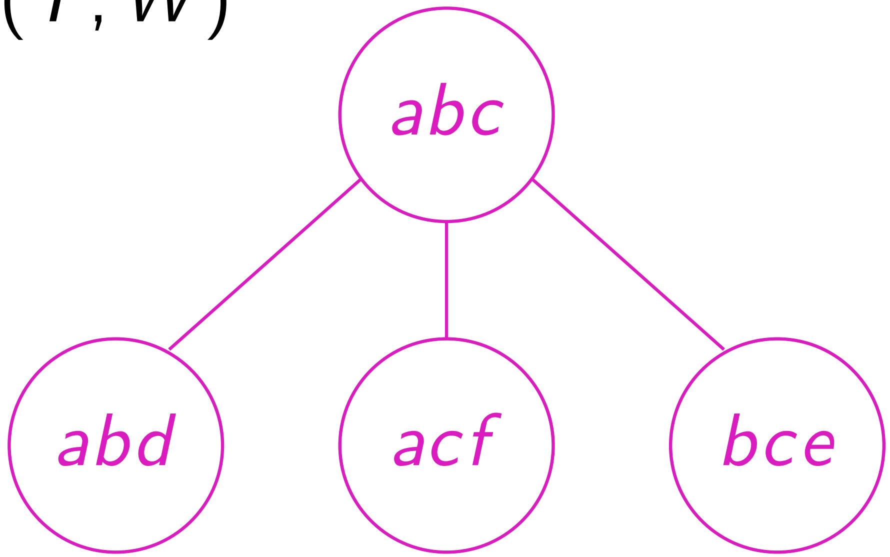


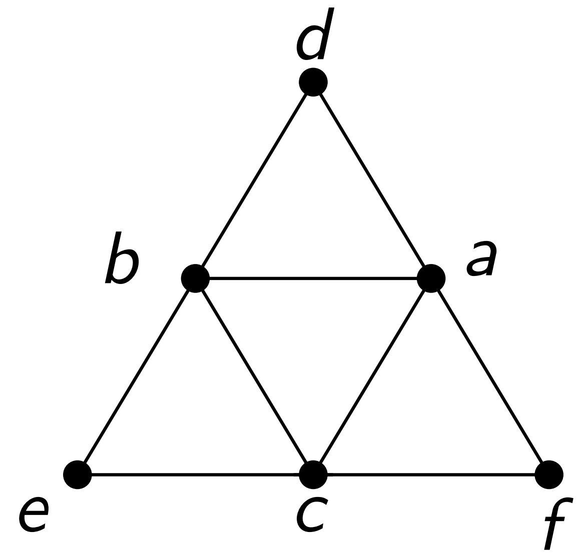


- treewidth of 2 for any maximal series-parallel graph
- Every child of t shares exactly two vertices with t

Tree Decomposition Of SP-Graphs

- (T, W)

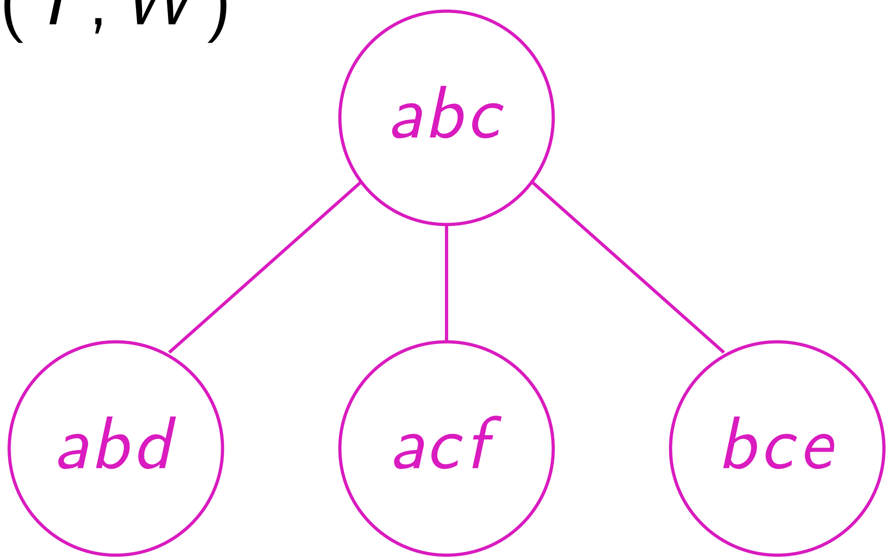


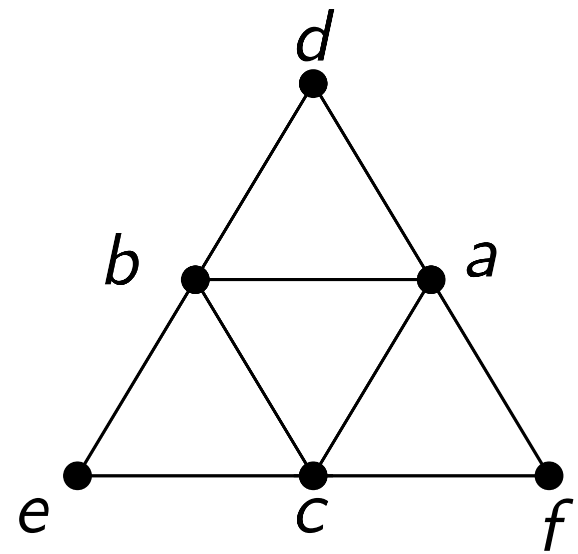


- treewidth of 2 for any maximal series-parallel graph
- Every child of t shares exactly two vertices with t
- For maximal outerplanar graphs
  - Siblings share exactly one vertex

Tree Decomposition Of SP-Graphs

- (T, W)

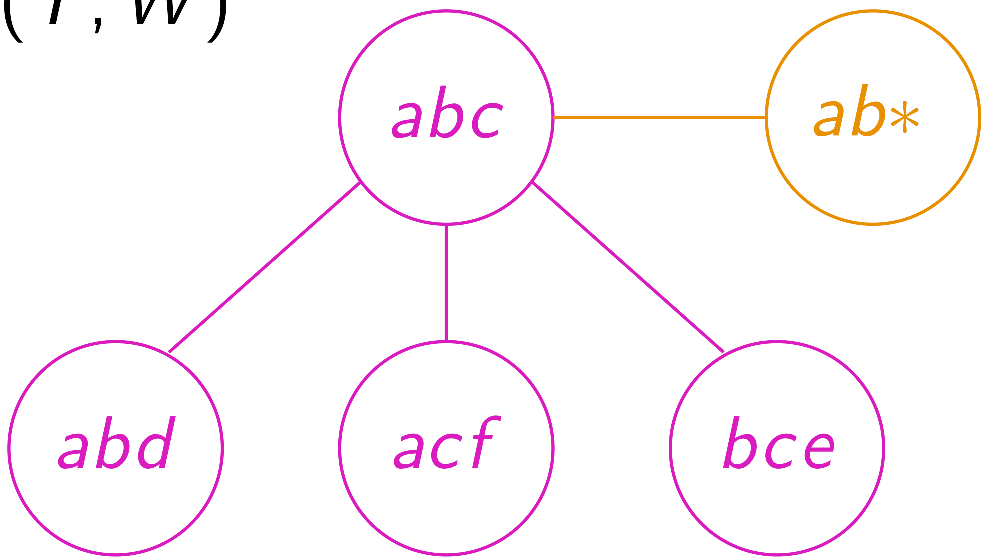


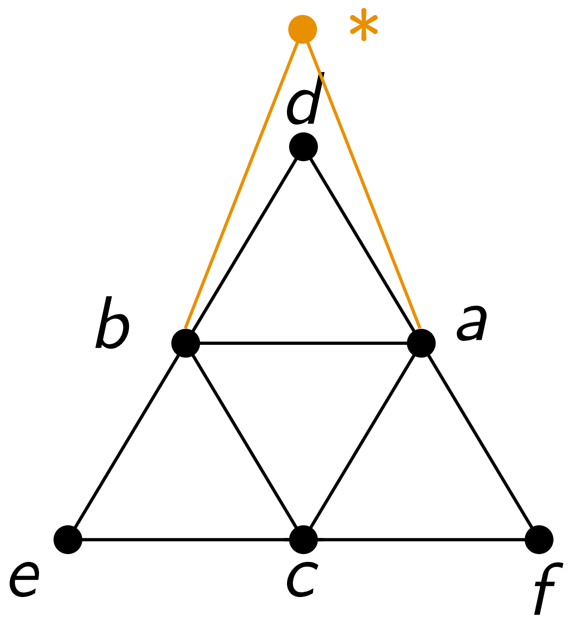


- treewidth of 2 for any maximal series-parallel graph
- Every child of t shares exactly two vertices with t
- For maximal outerplanar graphs
  - Siblings share exactly one vertex
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices

Tree Decomposition Of SP-Graphs

- (T, W)

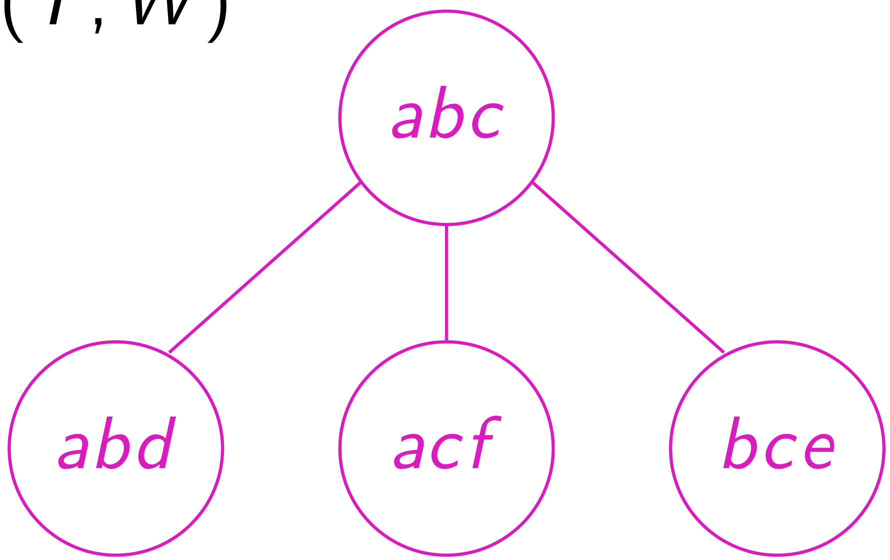


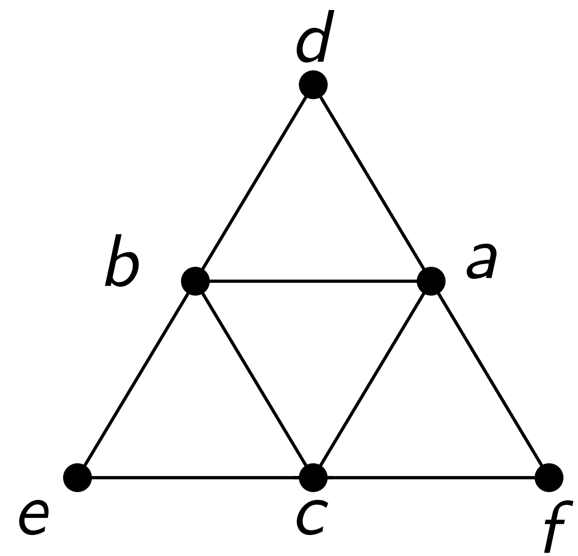


- treewidth of 2 for any maximal series-parallel graph
- Every child of t shares exactly two vertices with t
- For maximal outerplanar graphs
  - Siblings share exactly one vertex
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices

Tree Decomposition Of SP-Graphs

- (T, W)

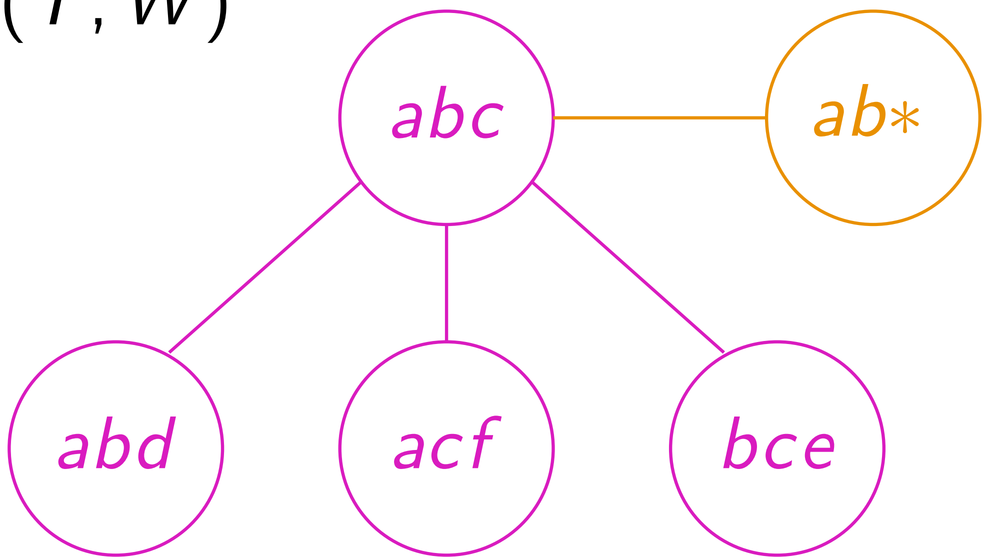


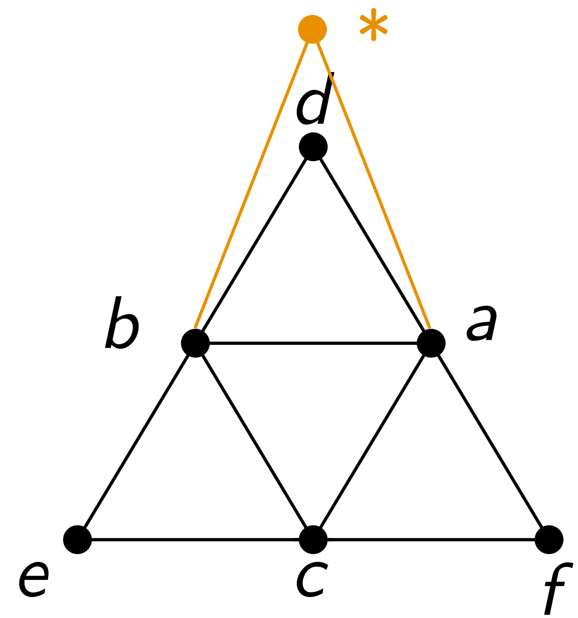


- treewidth of 2 for any maximal series-parallel graph
- Every child of t shares exactly two vertices with t
- For maximal outerplanar graphs
  - Siblings share exactly one vertex
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices
  - Also, $T'_{v,w} := T'_v \cap T'_w$, $|T'_{v,w}| \leq 2$

Tree Decomposition Of SP-Graphs

- (T, W)





- treewidth of 2 for any maximal series-parallel graph
- Every child of t shares exactly two vertices with t
- For maximal outerplanar graphs
  - Siblings share exactly one vertex
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices
  - Also, $T'_{v,w} := T'_v \cap T'_w$, $|T'_{v,w}| \leq 2$

Approach II: Scheme

- Make outerplanar graph G maximal, if necessary

Approach II: Scheme

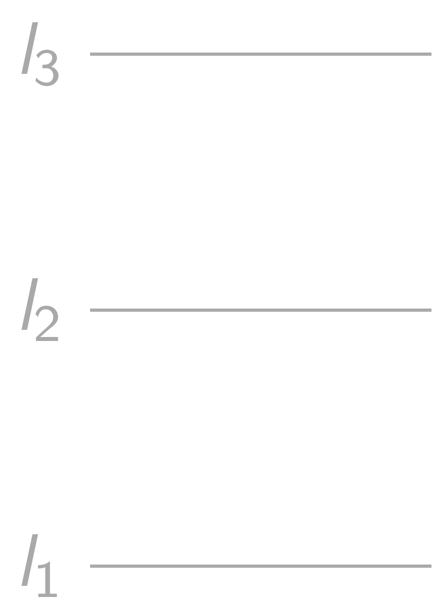
- Make outerplanar graph G maximal, if necessary
- Calculate (T, W)

Approach II: Scheme

- Make outerplanar graph G maximal, if necessary
- Calculate (T, W)
- Traverse T and draw G in a *layered box drawing* B

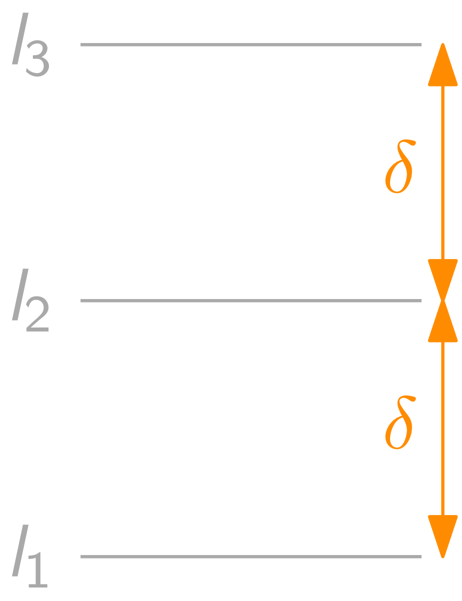
Approach II: Scheme

- Make outerplanar graph G maximal, if necessary
- Calculate (T, W)
- Traverse T and draw G in a *layered box drawing* B
  - Layering property: If $(v, w) \in E(G)$, then v and w do not share the same y value in drawing



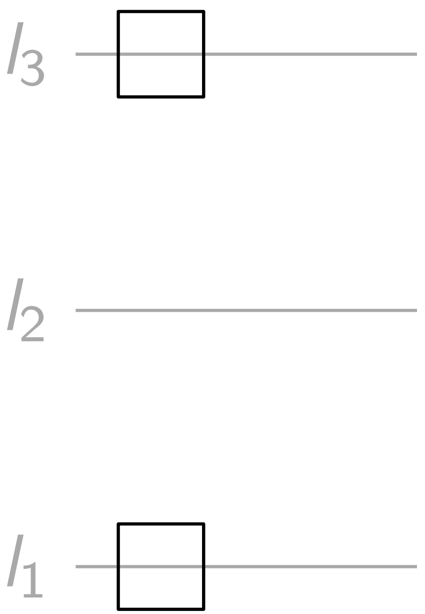
Approach II: Scheme

- Make outerplanar graph G maximal, if necessary
- Calculate (T, W)
- Traverse T and draw G in a *layered box drawing* B
  - Layering property: If $(v, w) \in E(G)$, then v and w do not share the same y value in drawing
  - Minimal distance δ between layers induces minimal Euclidian distance



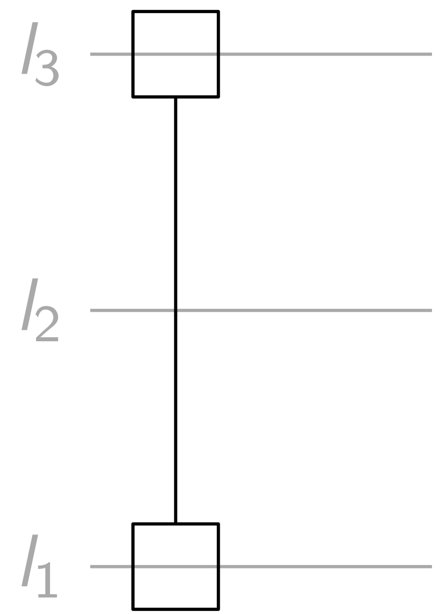
Approach II: Scheme

- Make outerplanar graph G maximal, if necessary
- Calculate (T, W)
- Traverse T and draw G in a *layered box drawing* B
  - Layering property: If $(v, w) \in E(G)$, then v and w do not share the same y value in drawing
  - Minimal distance δ between layers induces minimal Euclidian distance
  - *Box drawing*: vertices as axis-aligned boxes with minimal width and height of 2



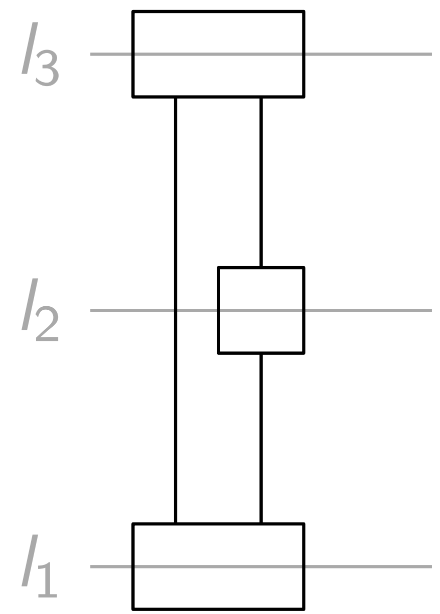
Approach II: Scheme

- Make outerplanar graph G maximal, if necessary
- Calculate (T, W)
- Traverse T and draw G in a *layered box drawing* B
  - Layering property: If $(v, w) \in E(G)$, then v and w do not share the same y value in drawing
  - Minimal distance δ between layers induces minimal Euclidian distance
  - *Box drawing*: vertices as axis-aligned boxes with minimal width and height of 2
  - Edges only vertically



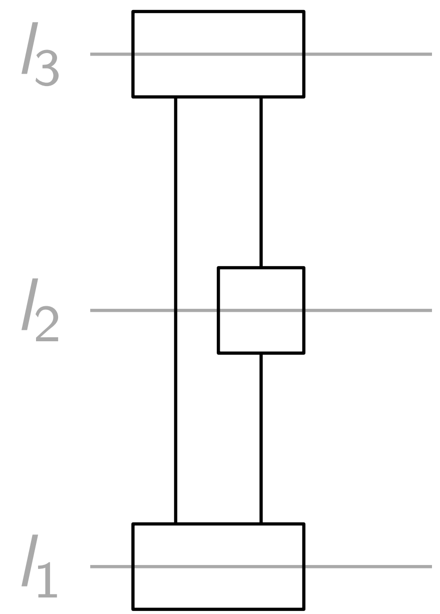
Approach II: Scheme

- Make outerplanar graph G maximal, if necessary
- Calculate (T, W)
- Traverse T and draw G in a *layered box drawing* B
  - Layering property: If $(v, w) \in E(G)$, then v and w do not share the same y value in drawing
  - Minimal distance δ between layers induces minimal Euclidian distance
  - *Box drawing*: vertices as axis-aligned boxes with minimal width and height of 2
  - Edges only vertically
  - Vertex insertions in free, *reachable* layers



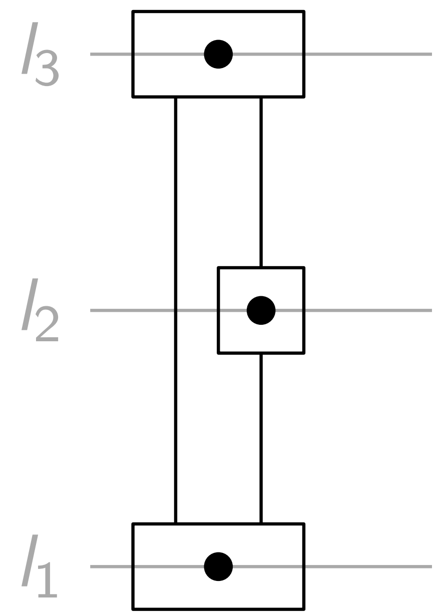
Approach II: Scheme

- Make outerplanar graph G maximal, if necessary
- Calculate (T, W)
- Traverse T and draw G in a *layered box drawing* B
  - Layering property: If $(v, w) \in E(G)$, then v and w do not share the same y value in drawing
  - Minimal distance δ between layers induces minimal Euclidian distance
  - *Box drawing*: vertices as axis-aligned boxes with minimal width and height of 2
  - Edges only vertically
  - Vertex insertions in free, *reachable* layers
  - Box drawings transferable to polyline drawings with same area bounds, 2 bends



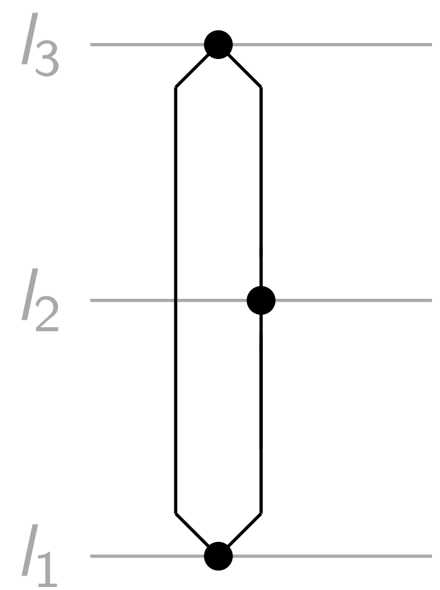
Approach II: Scheme

- Make outerplanar graph G maximal, if necessary
- Calculate (T, W)
- Traverse T and draw G in a *layered box drawing* B
  - Layering property: If $(v, w) \in E(G)$, then v and w do not share the same y value in drawing
  - Minimal distance δ between layers induces minimal Euclidian distance
  - *Box drawing*: vertices as axis-aligned boxes with minimal width and height of 2
  - Edges only vertically
  - Vertex insertions in free, *reachable* layers
  - Box drawings transferable to polyline drawings with same area bounds, 2 bends



Approach II: Scheme

- Make outerplanar graph G maximal, if necessary
- Calculate (T, W)
- Traverse T and draw G in a *layered box drawing* B
  - Layering property: If $(v, w) \in E(G)$, then v and w do not share the same y value in drawing
  - Minimal distance δ between layers induces minimal Euclidian distance
  - *Box drawing*: vertices as axis-aligned boxes with minimal width and height of 2
  - Edges only vertically
  - Vertex insertions in free, *reachable* layers
  - Box drawings transferable to polyline drawings with same area bounds, 2 bends



- Transfer B into a polyline drawing, remove edges

Vertex Insertions In Box Drawing

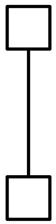
- Vertices inserted on layers as extendable boxes

Vertex Insertions In Box Drawing

- Vertices inserted on layers as extendable boxes
- Box drawing will be computed from left to right

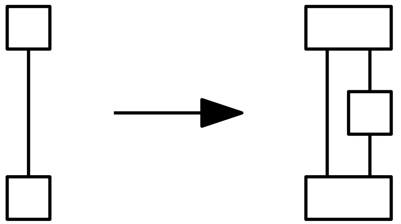
Vertex Insertions In Box Drawing

- Vertices inserted on layers as extendable boxes
- Box drawing will be computed from left to right



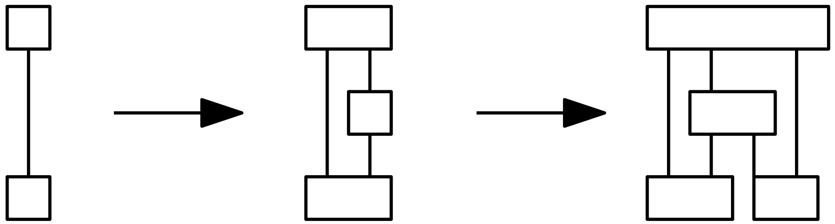
Vertex Insertions In Box Drawing

- Vertices inserted on layers as extendable boxes
- Box drawing will be computed from left to right



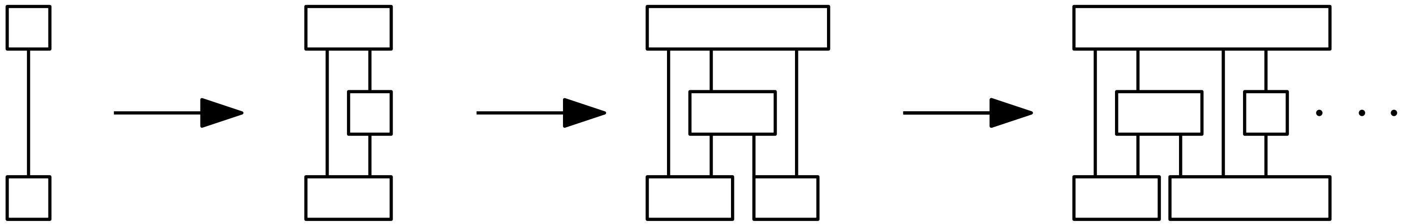
Vertex Insertions In Box Drawing

- Vertices inserted on layers as extendable boxes
- Box drawing will be computed from left to right



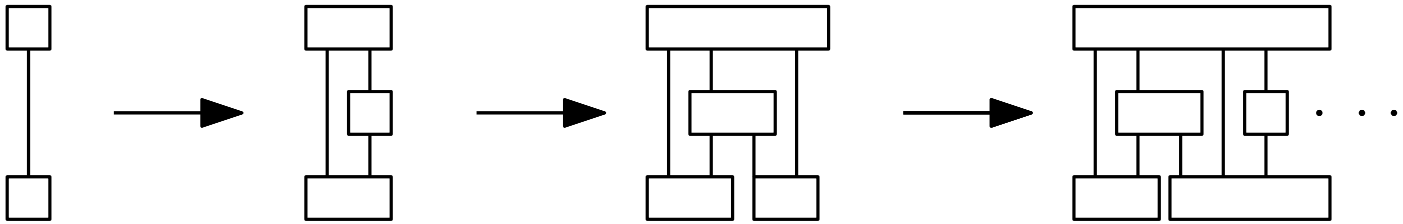
Vertex Insertions In Box Drawing

- Vertices inserted on layers as extendable boxes
- Box drawing will be computed from left to right



Vertex Insertions In Box Drawing

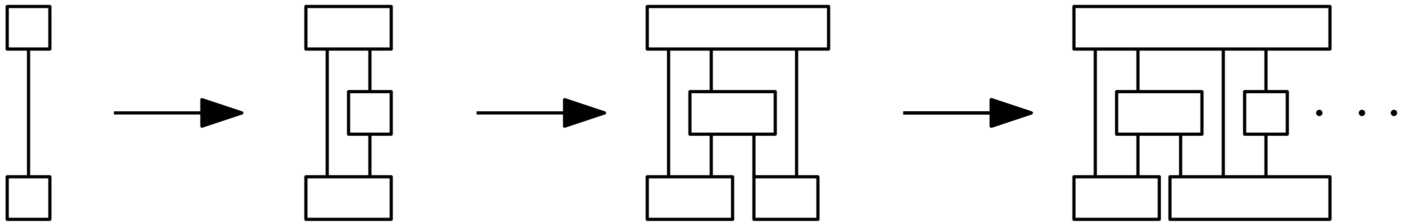
- Vertices inserted on layers as extendable boxes
- Box drawing will be computed from left to right



- Goal: Re-use layers as often as possible and insert new layers only when necessary

Vertex Insertions In Box Drawing

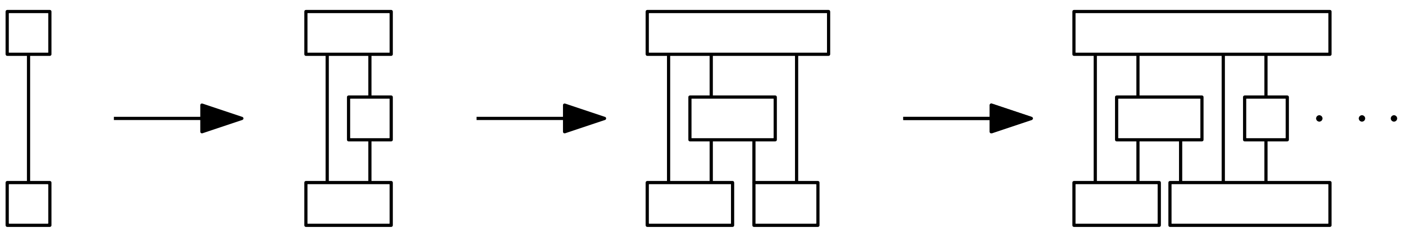
- Vertices inserted on layers as extendable boxes
- Box drawing will be computed from left to right



- Goal: Re-use layers as often as possible and insert new layers only when necessary
  - Amount of layers determines the height and the ratio of the drawing

Vertex Insertions In Box Drawing

- Vertices inserted on layers as extendable boxes
- Box drawing will be computed from left to right



- Goal: Re-use layers as often as possible and insert new layers only when necessary
  - Amount of layers determines the height and the ratio of the drawing

- A layer is *reachable* if it is free, v can be placed on it and edges to a, b can be drawn without destroying planarity

Reachability of Layers

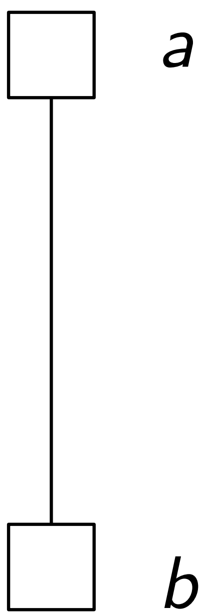
- Case 1: Reachable layer is inbetween layers of a, b

Reachability of Layers

- Case 1: Reachable layer is inbetween layers of a, b
  - Insert v on this layer, extend boxes if necessary

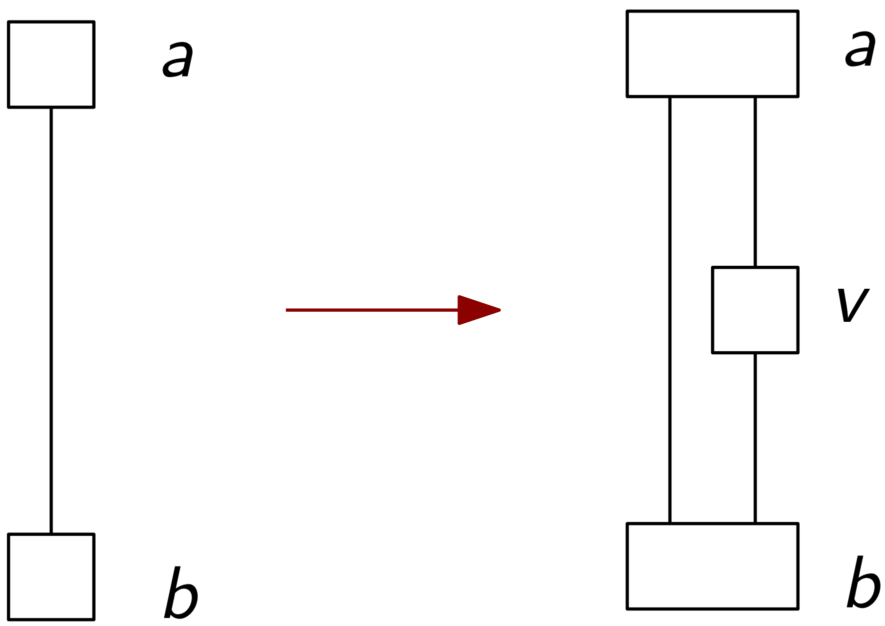
Reachability of Layers

- Case 1: Reachable layer is inbetween layers of a, b
  - Insert v on this layer, extend boxes if necessary



Reachability of Layers

- Case 1: Reachable layer is inbetween layers of a, b
  - Insert v on this layer, extend boxes if necessary

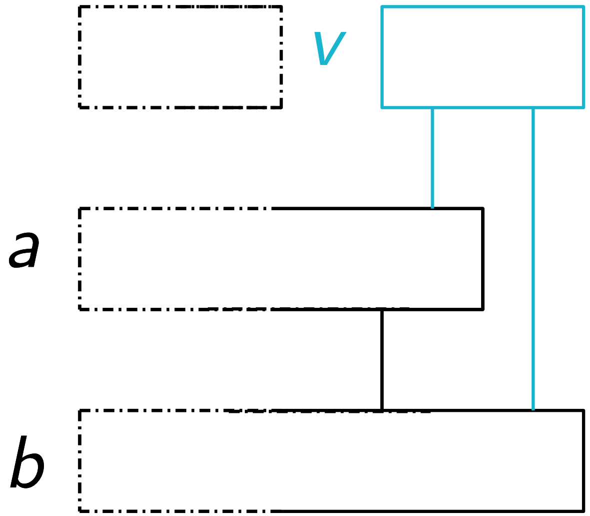


Reachability of Layers

- Case 2: Reachable layer is atop or below both layers of a, b (symmetrical case)

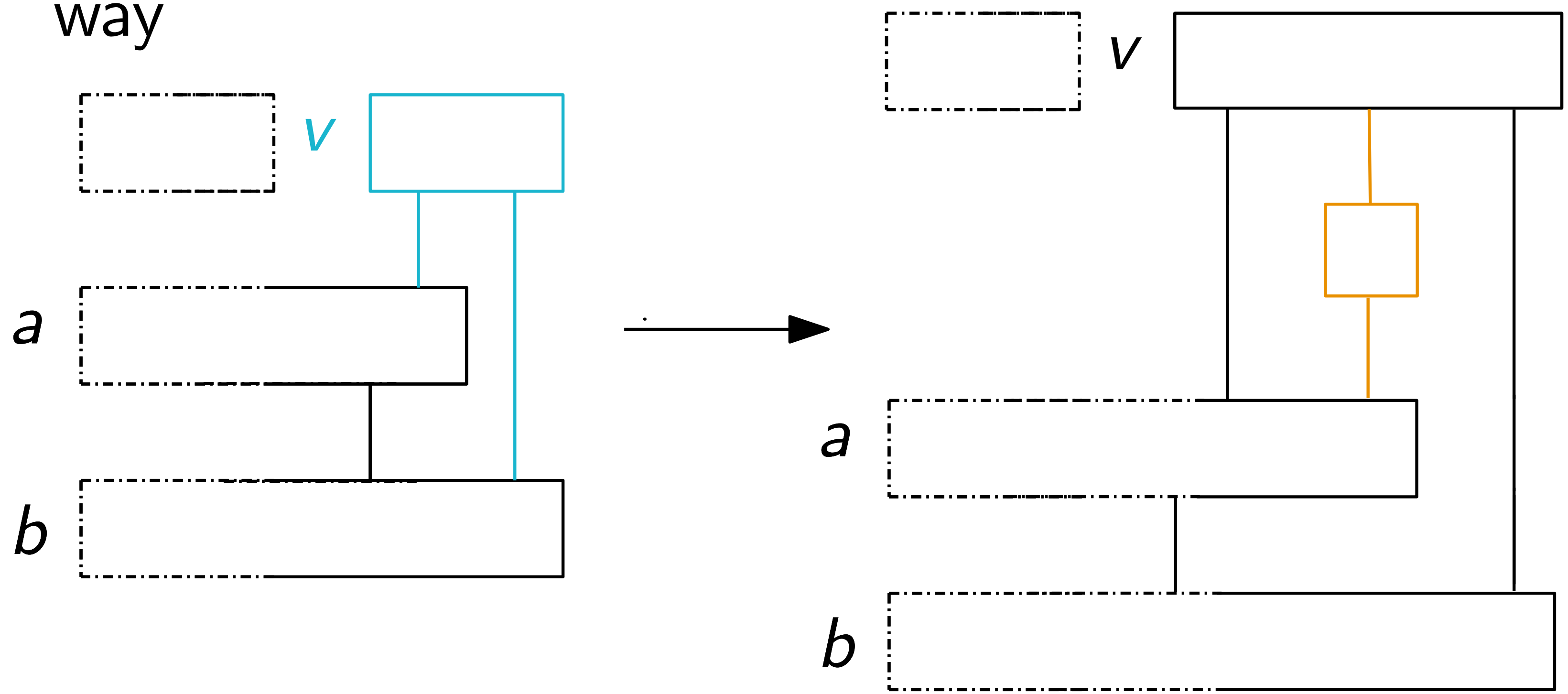
Reachability of Layers

- Case 2: Reachable layer is atop or below both layers of a, b (symmetrical case)
  - Extend boxes and Insert v on this layer the following way



Reachability of Layers

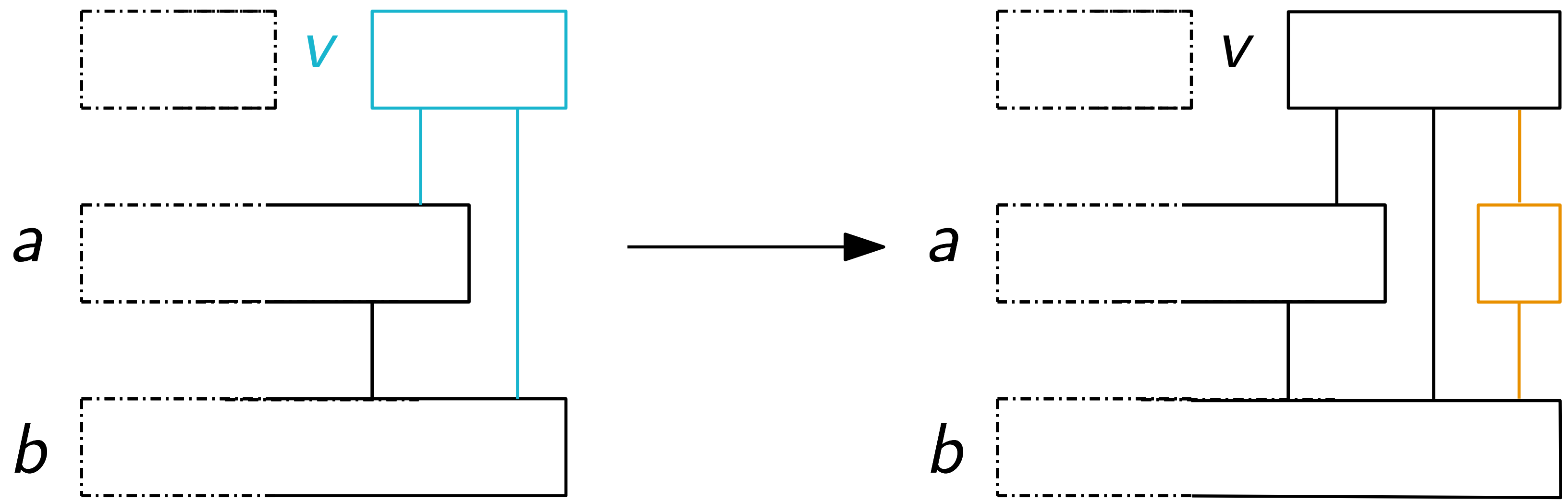
- Case 2: Reachable layer is atop or below both layers of a, b (symmetrical case)
  - Extend boxes and Insert v on this layer the following way



  - Subsequent vertex insertions between v and a
  - Destroy outerplanarity property

Reachability of Layers

- Case 2: Reachable layer is atop or below both layers of a, b (symmetrical case)
  - Extend boxes and Insert v on this layer the following way



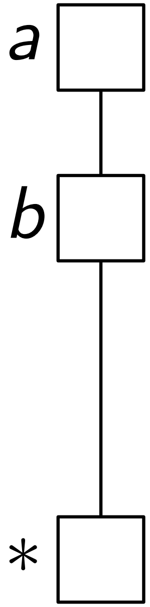
  - Subsequent insertions between v and b

Reachability of Layers

- Case 3: Layer becomes reachable by layer reassignments of already drawn vertices

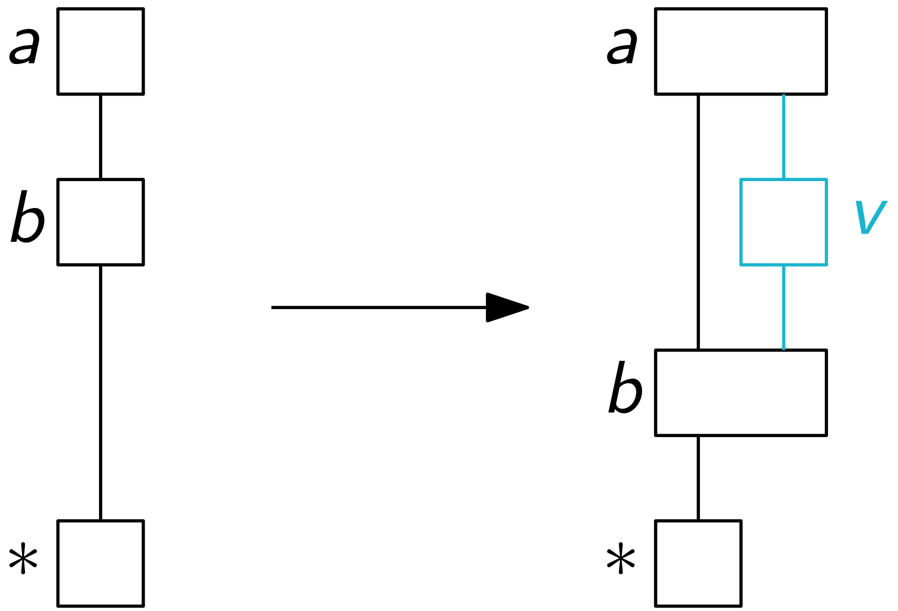
Reachability of Layers

- Case 3: Layer becomes reachable by layer reassignments of already drawn vertices



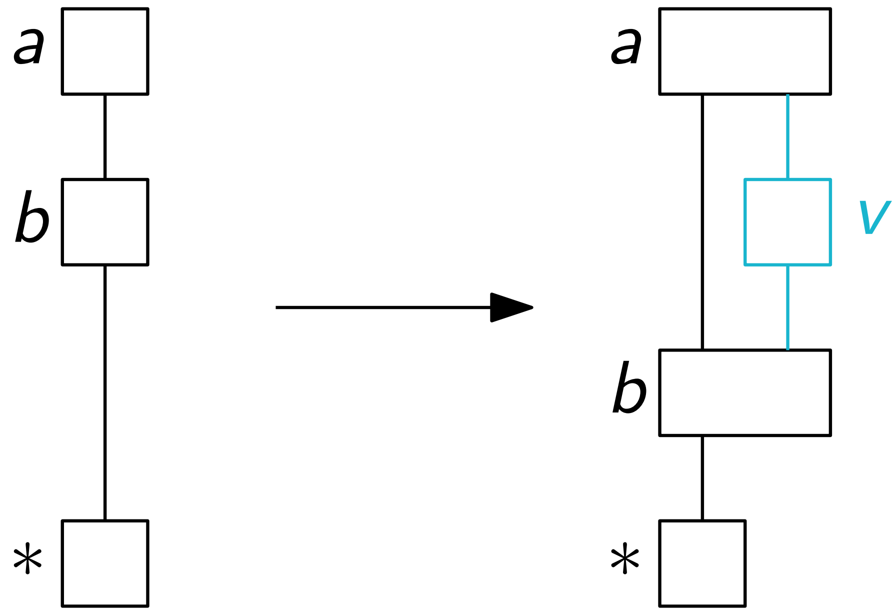
Reachability of Layers

- Case 3: Layer becomes reachable by layer reassignments of already drawn vertices



Reachability of Layers

- Case 3: Layer becomes reachable by layer reassignments of already drawn vertices



- Then, either case 1 or case 2 will apply
- If neither of those cases possible, insert a new layer between a and b

Traversal Of Tree Decomposition

- Traversal of T determines the order of vertices inserted

Traversal Of Tree Decomposition

- Traversal of T determines the order of vertices inserted
- subtree priority $+$ DFS

Traversal Of Tree Decomposition

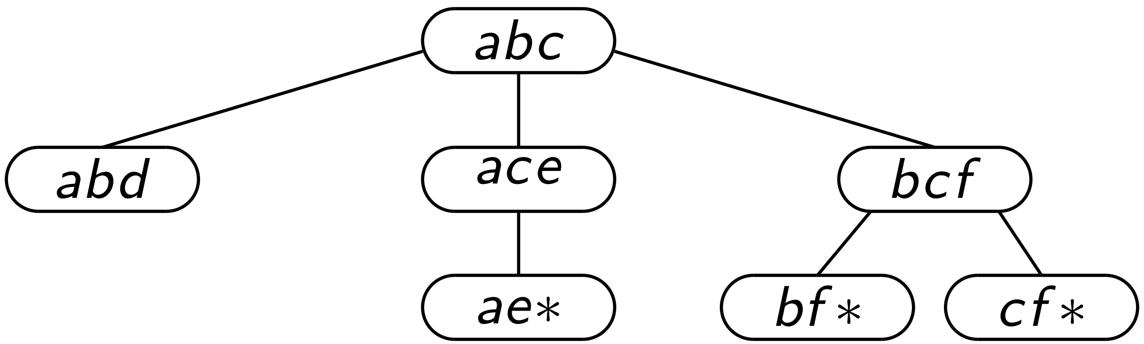
- Traversal of T determines the order of vertices inserted
- subtree priority $+$ DFS
  - Siblings of T share exactly one vertex in their bags

Traversal Of Tree Decomposition

- Traversal of T determines the order of vertices inserted
- subtree priority $+$ DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex

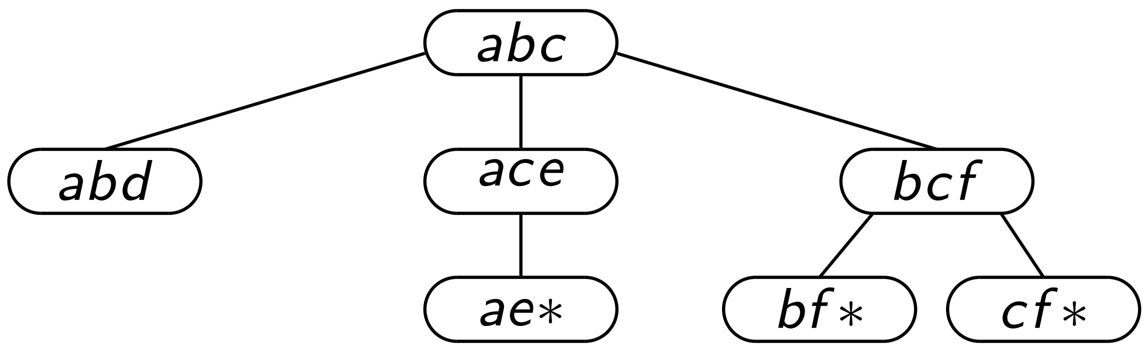
Traversal Of Tree Decomposition

- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



Traversal Of Tree Decomposition

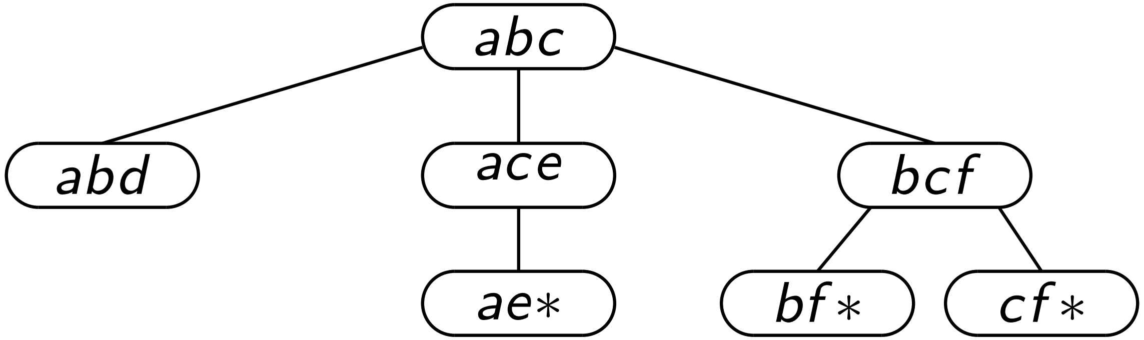
- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



  - Choose subtree with lowest height or amount of vertices

Traversal Of Tree Decomposition

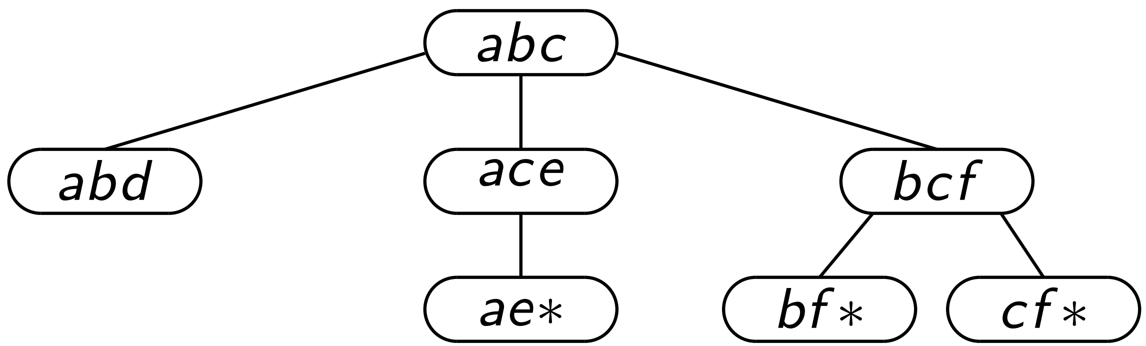
- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



  - Choose subtree with lowest height or amount of vertices
  - Finish drawing this subtree before proceeding to next one

Traversal Of Tree Decomposition

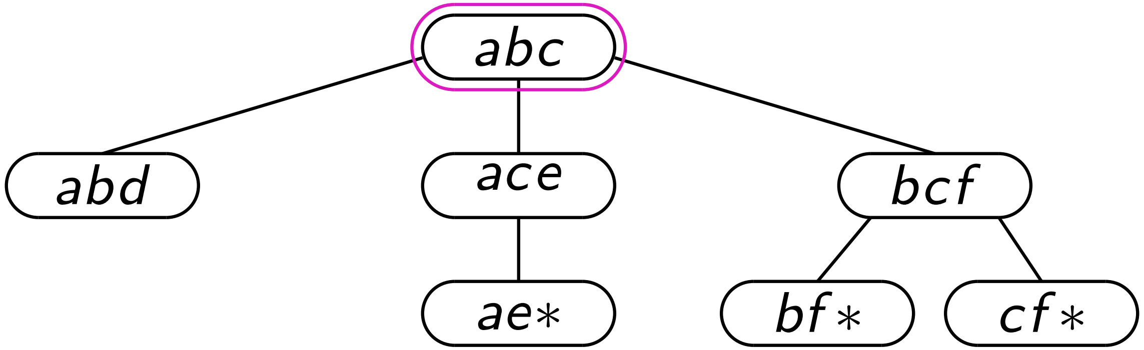
- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



  - Choose subtree with lowest height or amount of vertices
  - Finish drawing this subtree before proceeding to next one
  - This priority holds at every vertex of T

Traversal Of Tree Decomposition

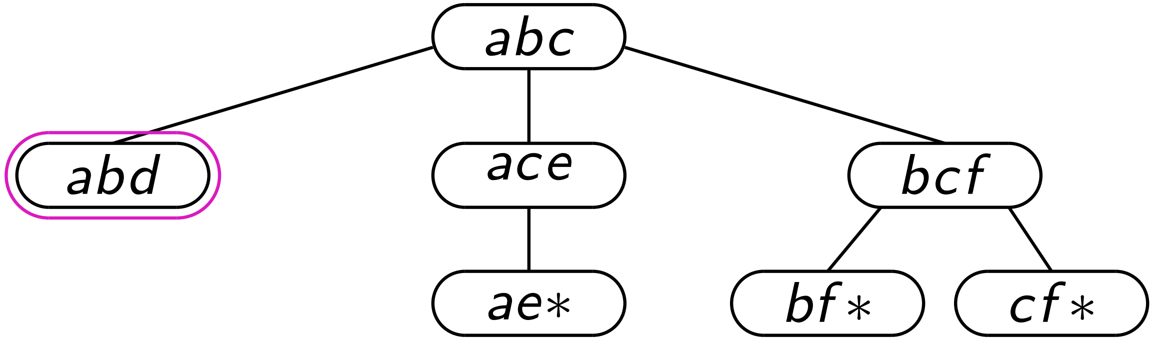
- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



  - Choose subtree with lowest height or amount of vertices
  - Finish drawing this subtree before proceeding to next one
  - This priority holds at every vertex of T

Traversal Of Tree Decomposition

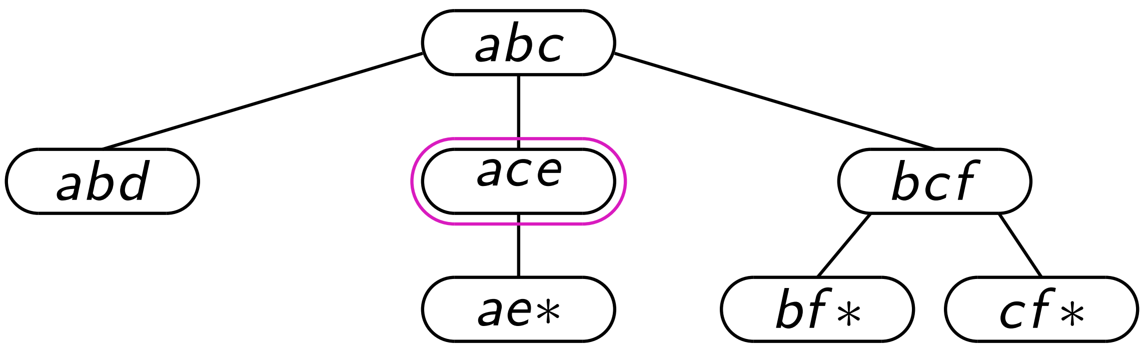
- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



  - Choose subtree with lowest height or amount of vertices
  - Finish drawing this subtree before proceeding to next one
  - This priority holds at every vertex of T

Traversal Of Tree Decomposition

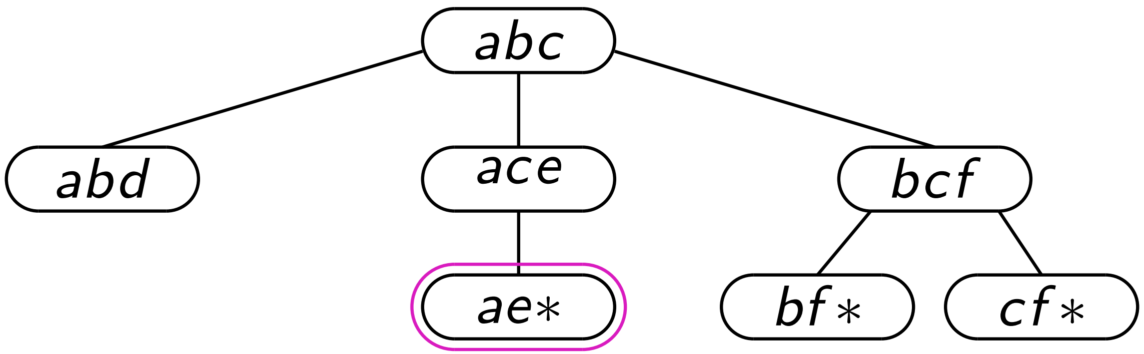
- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



  - Choose subtree with lowest height or amount of vertices
  - Finish drawing this subtree before proceeding to next one
  - This priority holds at every vertex of T

Traversal Of Tree Decomposition

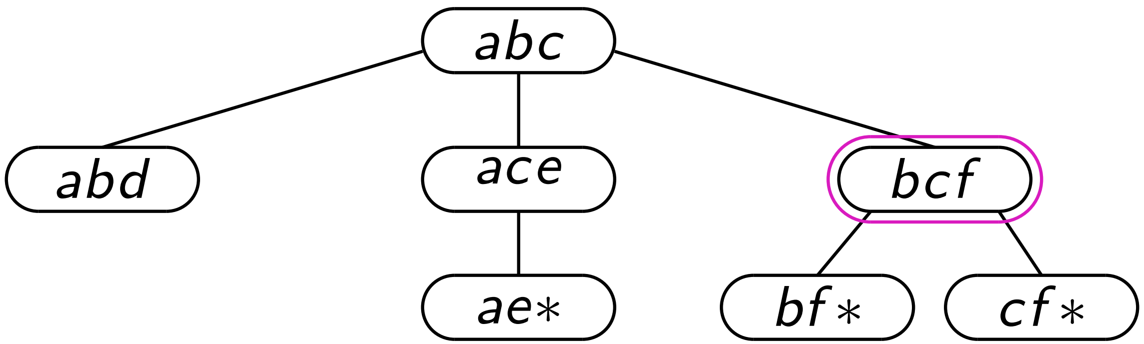
- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



  - Choose subtree with lowest height or amount of vertices
  - Finish drawing this subtree before proceeding to next one
  - This priority holds at every vertex of T

Traversal Of Tree Decomposition

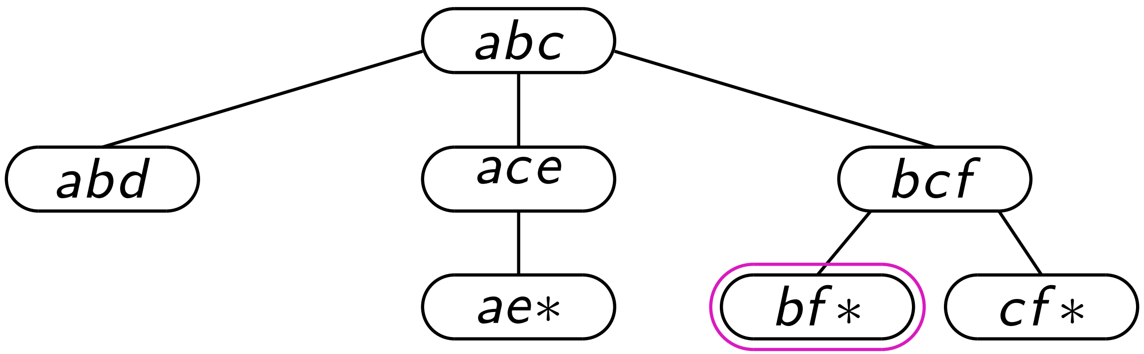
- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



  - Choose subtree with lowest height or amount of vertices
  - Finish drawing this subtree before proceeding to next one
  - This priority holds at every vertex of T

Traversal Of Tree Decomposition

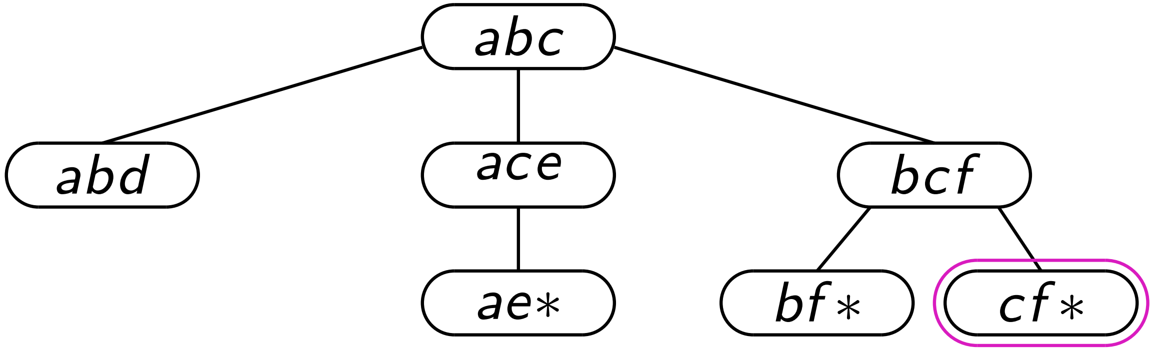
- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



  - Choose subtree with lowest height or amount of vertices
  - Finish drawing this subtree before proceeding to next one
  - This priority holds at every vertex of T

Traversal Of Tree Decomposition

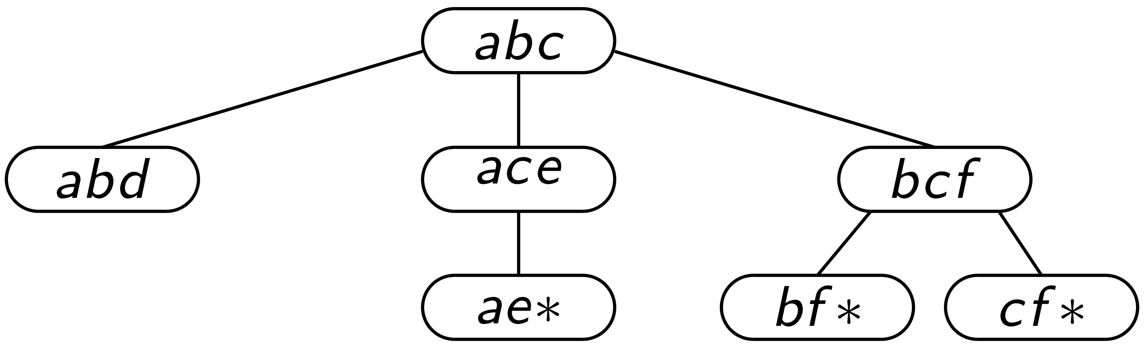
- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



  - Choose subtree with lowest height or amount of vertices
  - Finish drawing this subtree before proceeding to next one
  - This priority holds at every vertex of T

Traversal Of Tree Decomposition

- Traversal of T determines the order of vertices inserted
- subtree priority + DFS
  - Siblings of T share exactly one vertex in their bags
  - Subtrees adjacent to same vertex share exactly one vertex



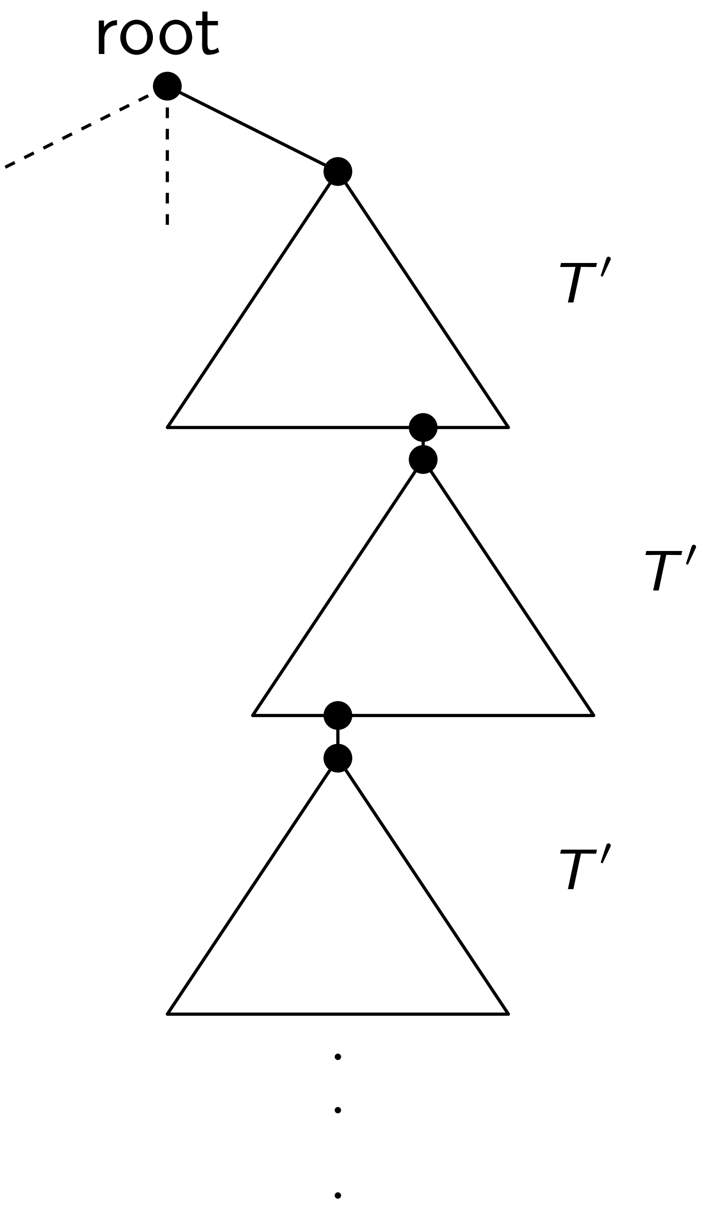
  - Choose subtree with lowest height or amount of vertices
  - Finish drawing this subtree before proceeding to next one
  - This priority holds at every vertex of T
  - Realizable by DFS traversal

Advantage of Prioritized DFS Traversal

- Consider following subtree of T

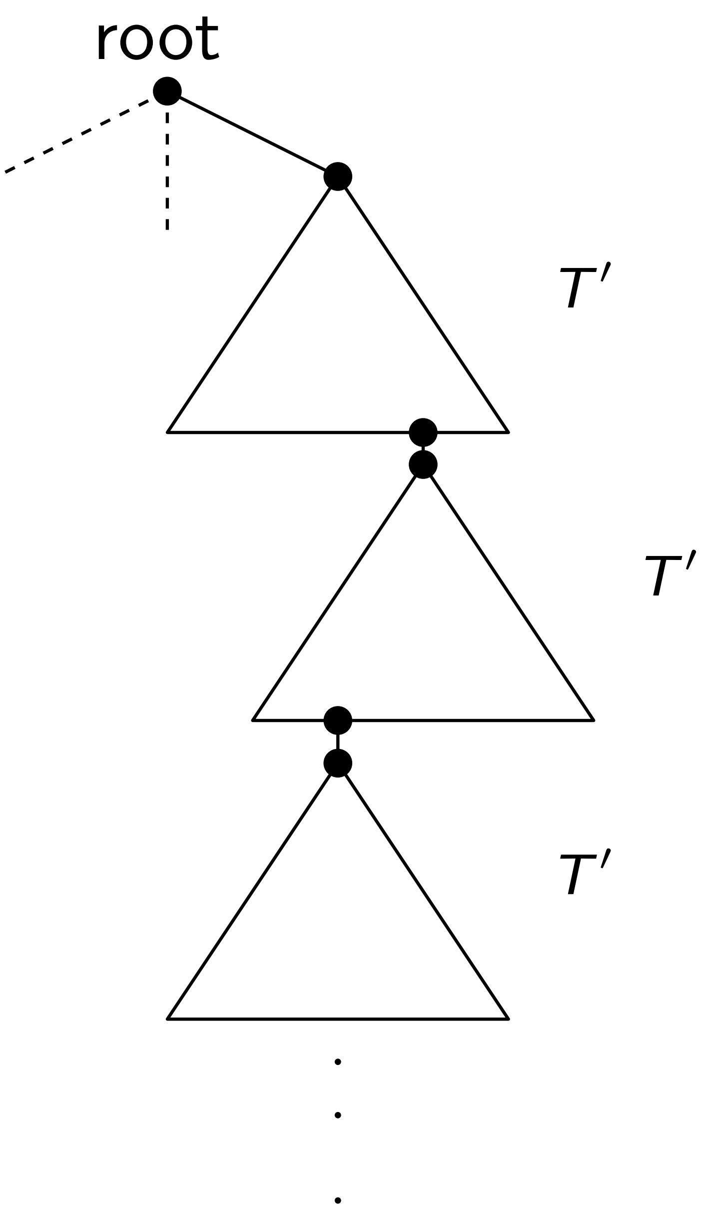
Advantage of Prioritized DFS Traversal

- Consider following subtree of T



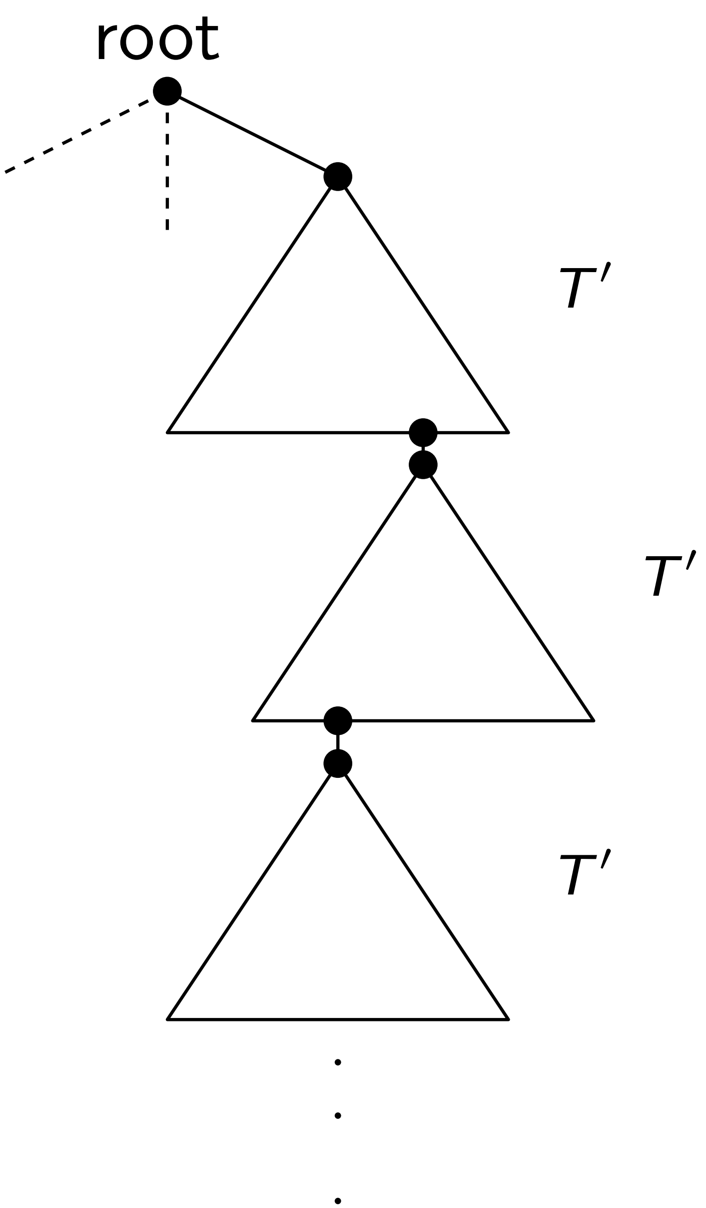
Advantage of Prioritized DFS Traversal

- Consider following subtree of T
- T' is a complete binary tree



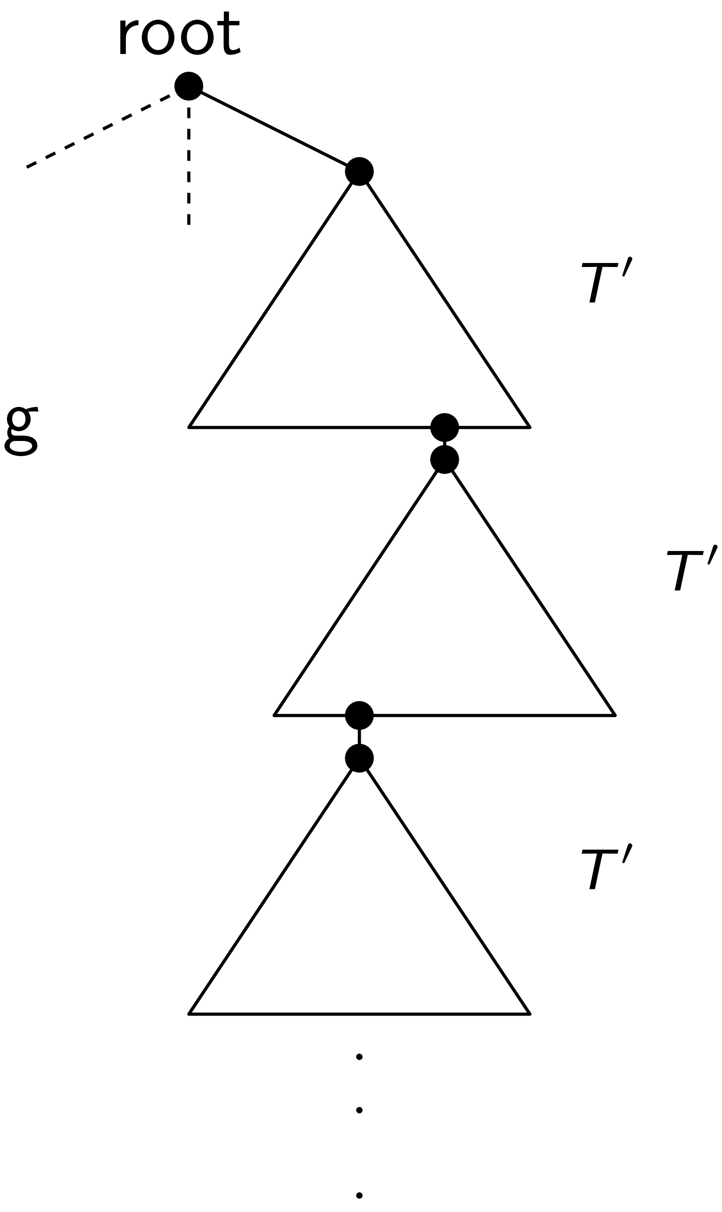
Advantage of Prioritized DFS Traversal

- Consider following subtree of T
- T' is a complete binary tree
- Prioritize subtrees with lower height at every step



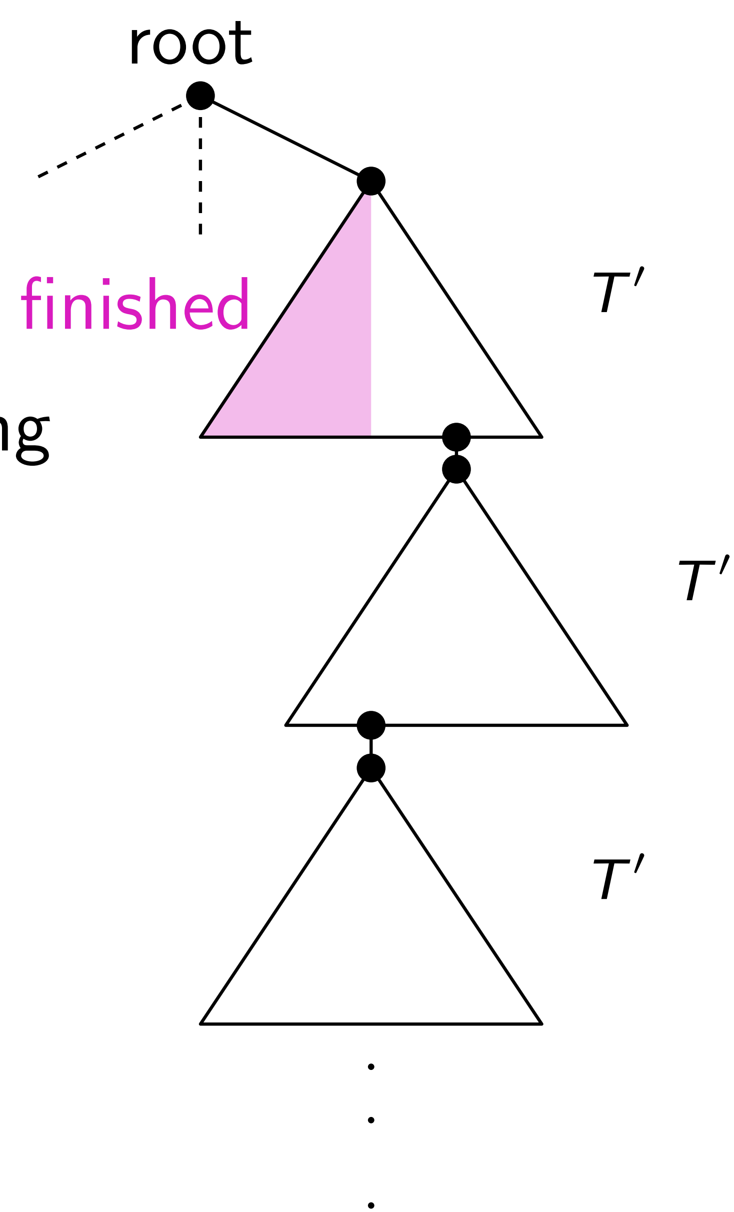
Advantage of Prioritized DFS Traversal

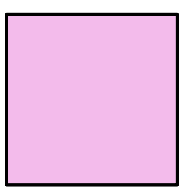
- Consider following subtree of T
- T' is a complete binary tree
- Prioritize subtrees with lower height at every step
- every T' finished before continuing



Advantage of Prioritized DFS Traversal

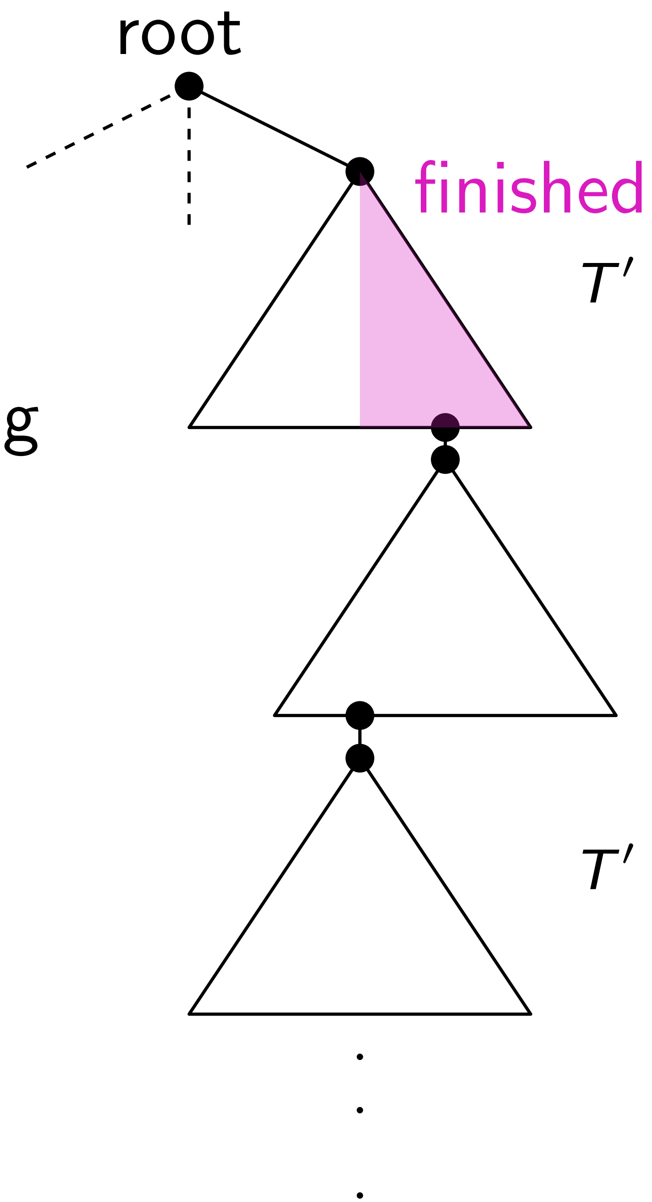
- Consider following subtree of T
- T' is a complete binary tree
- Prioritize subtrees with lower height at every step
- every T' finished before continuing





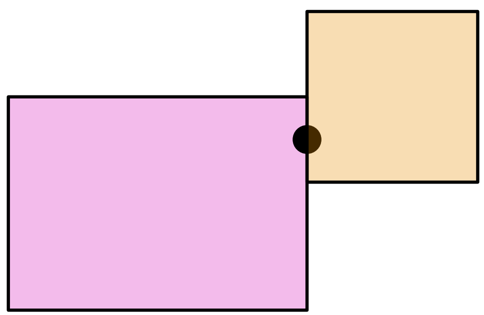
Advantage of Prioritized DFS Traversal

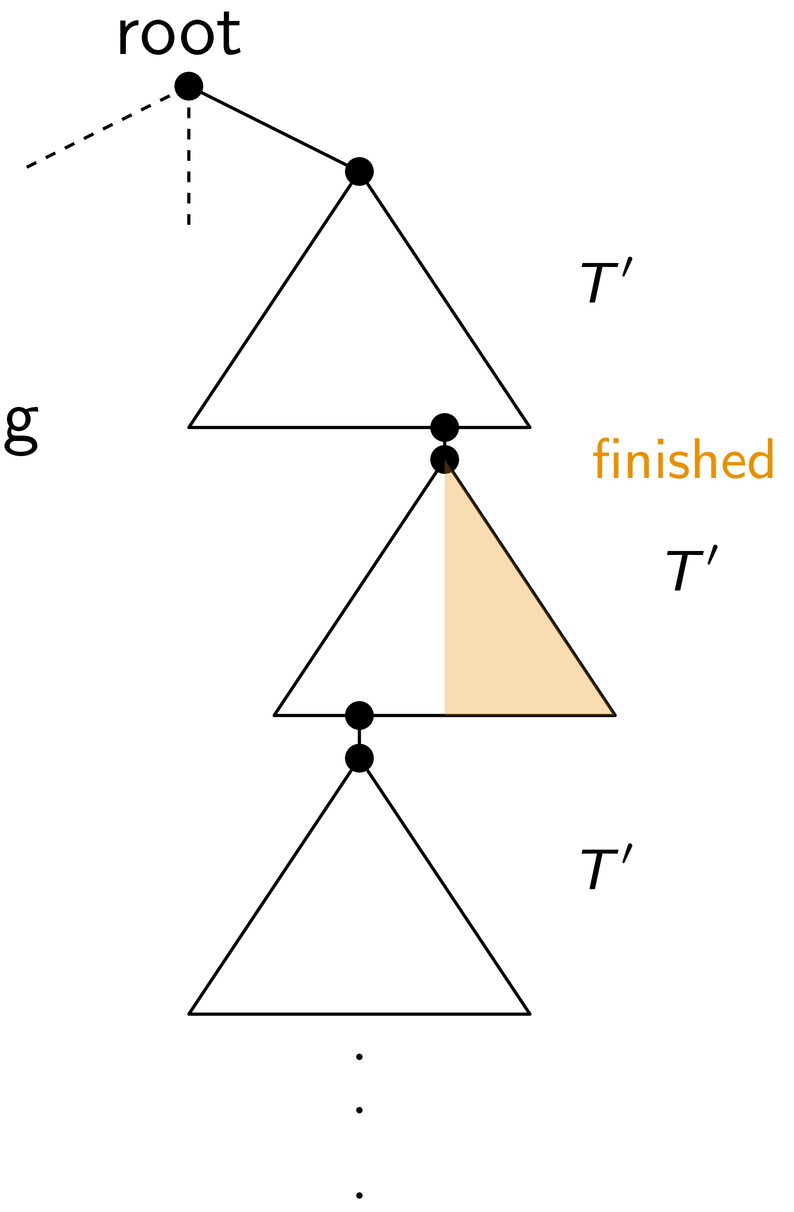
- Consider following subtree of T
- T' is a complete binary tree
- Prioritize subtrees with lower height at every step
- every T' finished before continuing



Advantage of Prioritized DFS Traversal

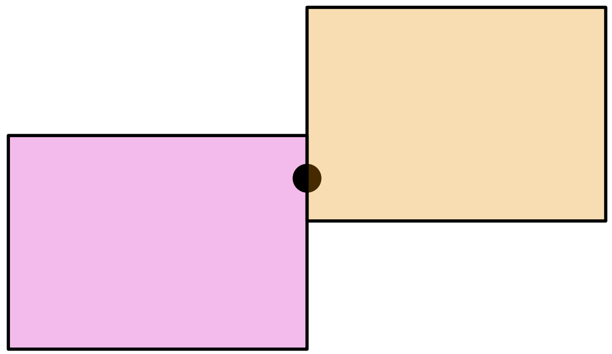
- Consider following subtree of T
- T' is a complete binary tree
- Prioritize subtrees with lower height at every step
- every T' finished before continuing

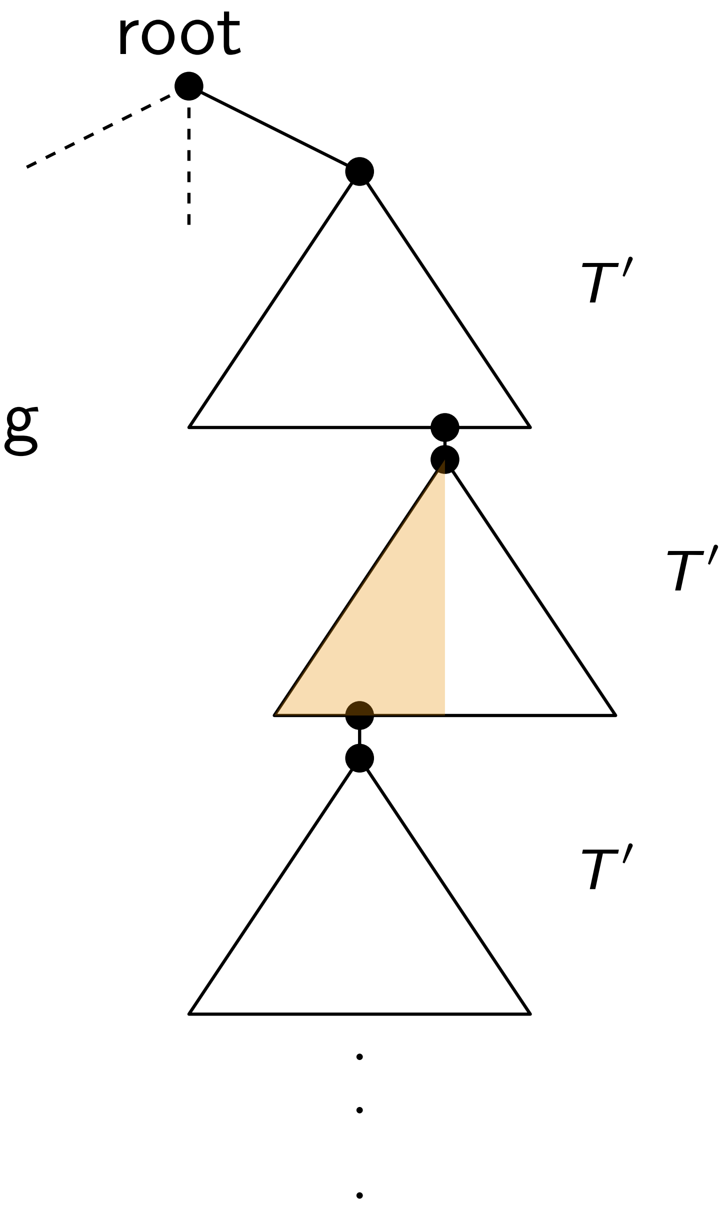




Advantage of Prioritized DFS Traversal

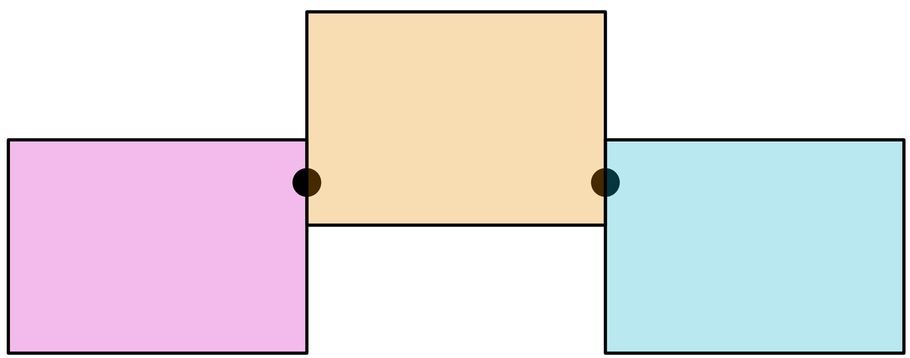
- Consider following subtree of T
- T' is a complete binary tree
- Prioritize subtrees with lower height at every step
- every T' finished before continuing

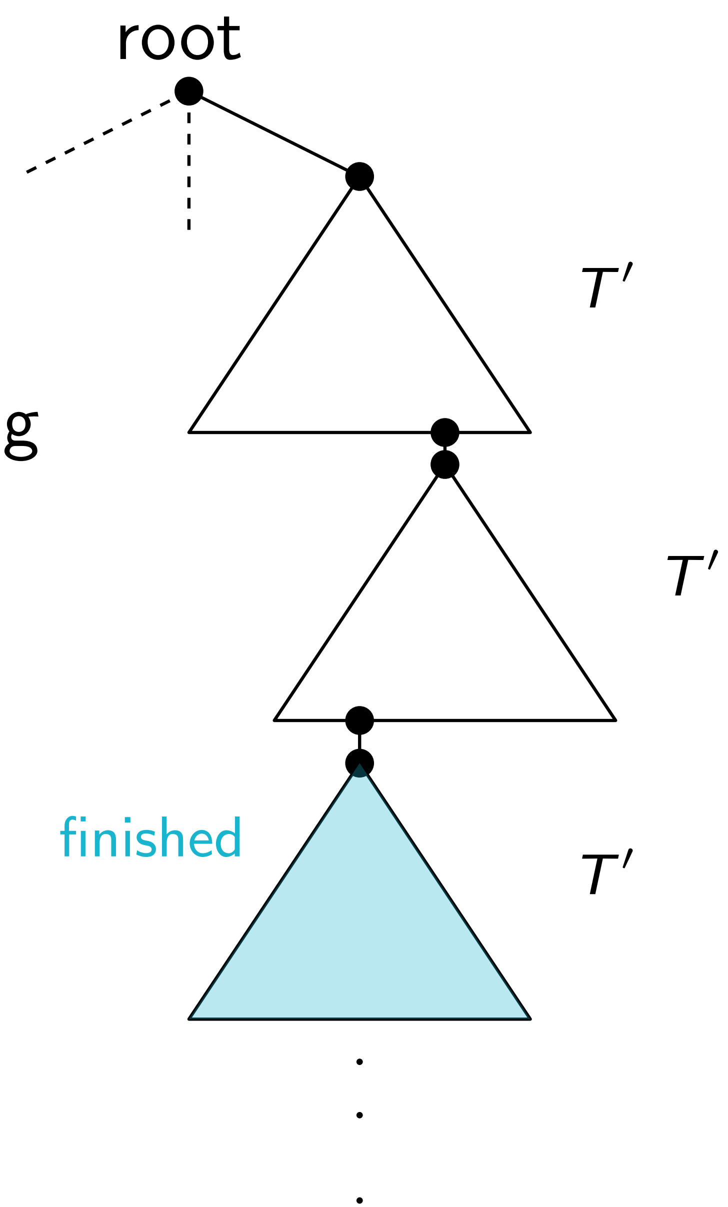




Advantage of Prioritized DFS Traversal

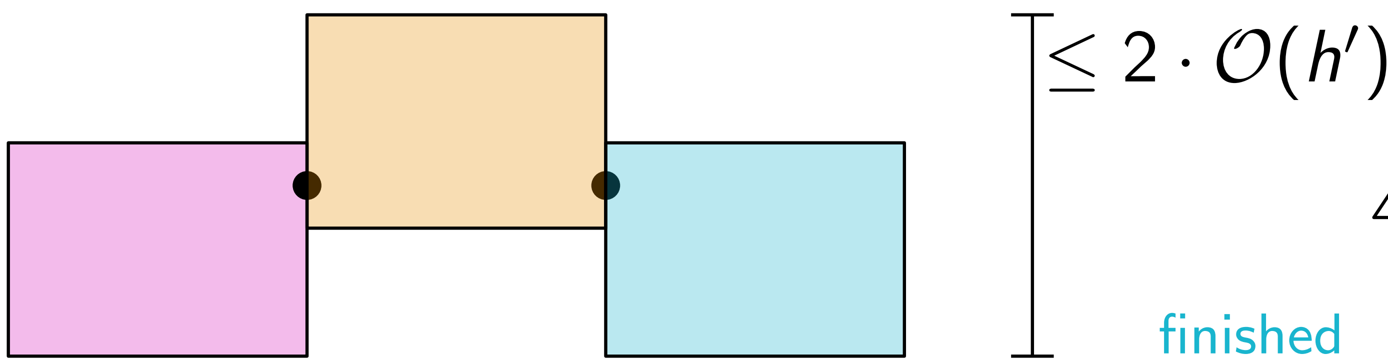
- Consider following subtree of T
- T' is a complete binary tree
- Prioritize subtrees with lower height at every step
- every T' finished before continuing

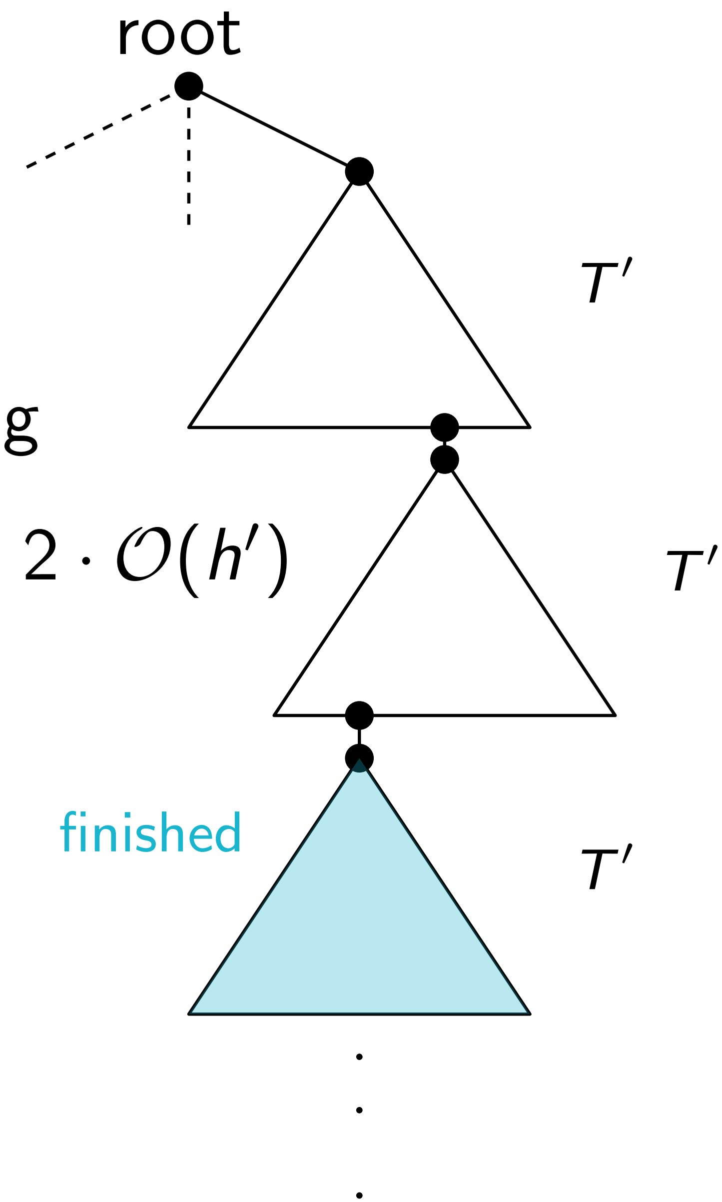




Advantage of Prioritized DFS Traversal

- Consider following subtree of T
- T' is a complete binary tree
- Prioritize subtrees with lower height at every step
- every T' finished before continuing





- Total amount of layers equals constant multiple of amount of layers induced by drawing T'

Total Height of Box Drawing

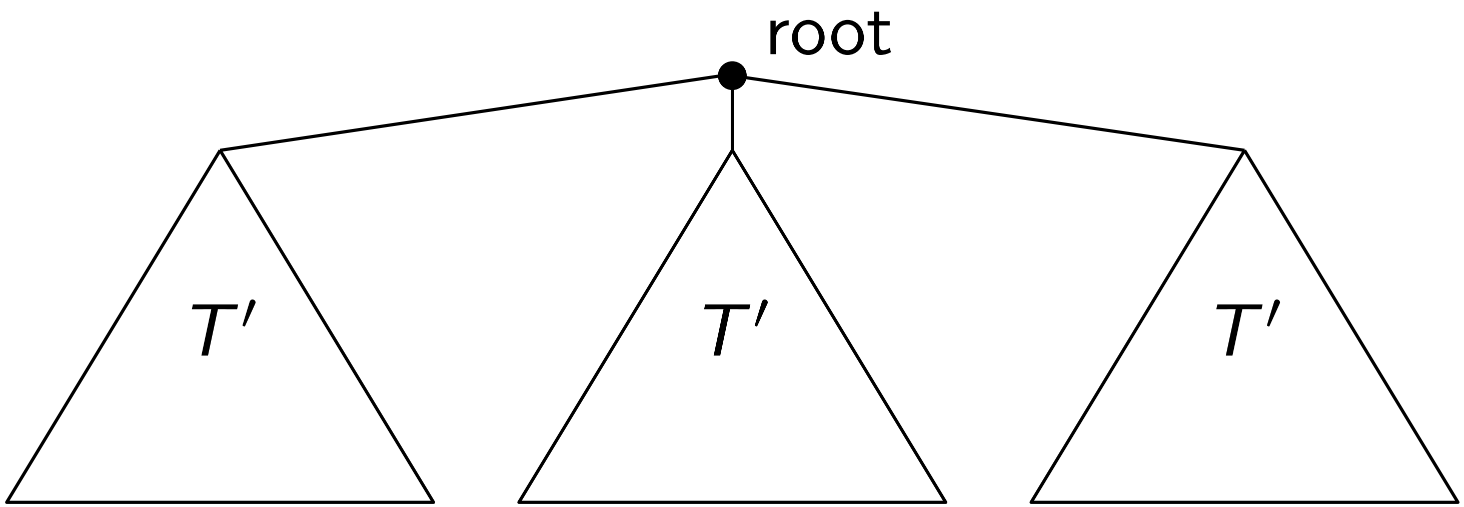
- Height of Box Drawing: $\delta \cdot \#$layers

Total Height of Box Drawing

- Height of Box Drawing: $\delta \cdot \#$layers
- DFS traversal, prioritizing subtrees of minimal height

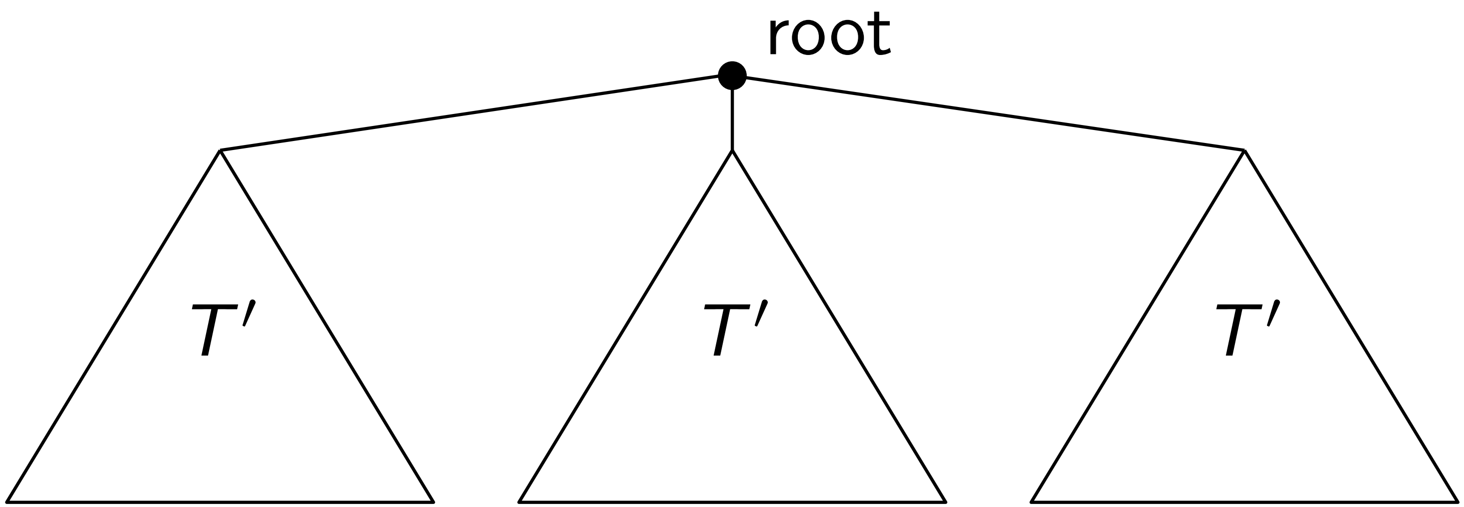
Total Height of Box Drawing

- Height of Box Drawing: $\delta \cdot \#\text{layers}$
- DFS traversal, prioritizing subtrees of minimal height
- Consider T with complete binary trees T' attached to root



Total Height of Box Drawing

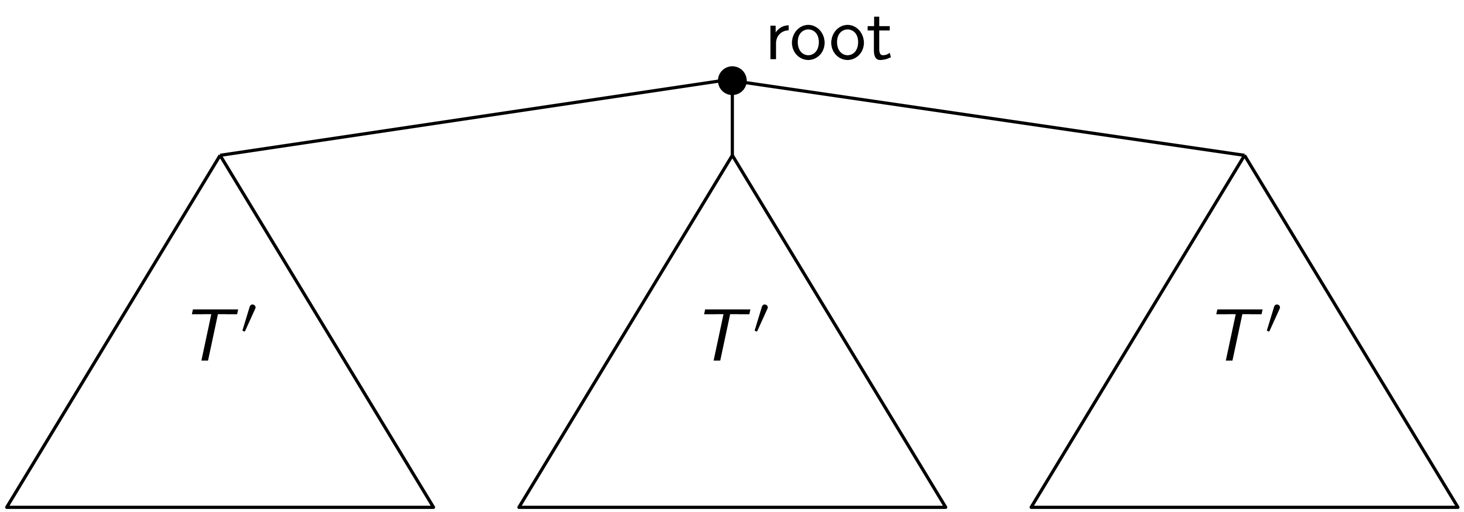
- Height of Box Drawing: $\delta \cdot \#\text{layers}$
- DFS traversal, prioritizing subtrees of minimal height
- Consider T with complete binary trees T' attached to root



- Subtree priority does not take effect at any stage of DFS

Total Height of Box Drawing

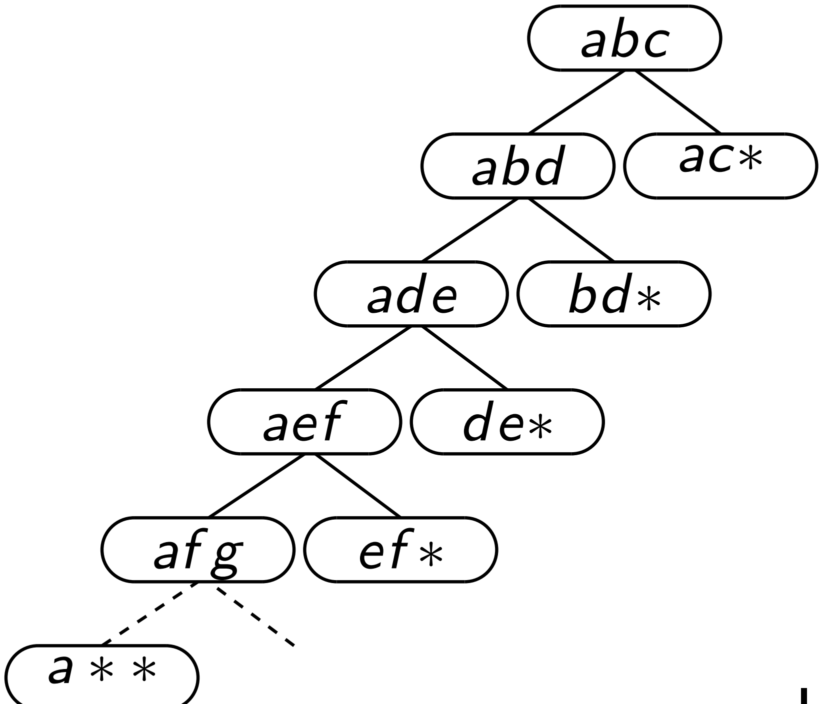
- Height of Box Drawing: $\delta \cdot \#\text{layers}$
- DFS traversal, prioritizing subtrees of minimal height
- Consider T with complete binary trees T' attached to root

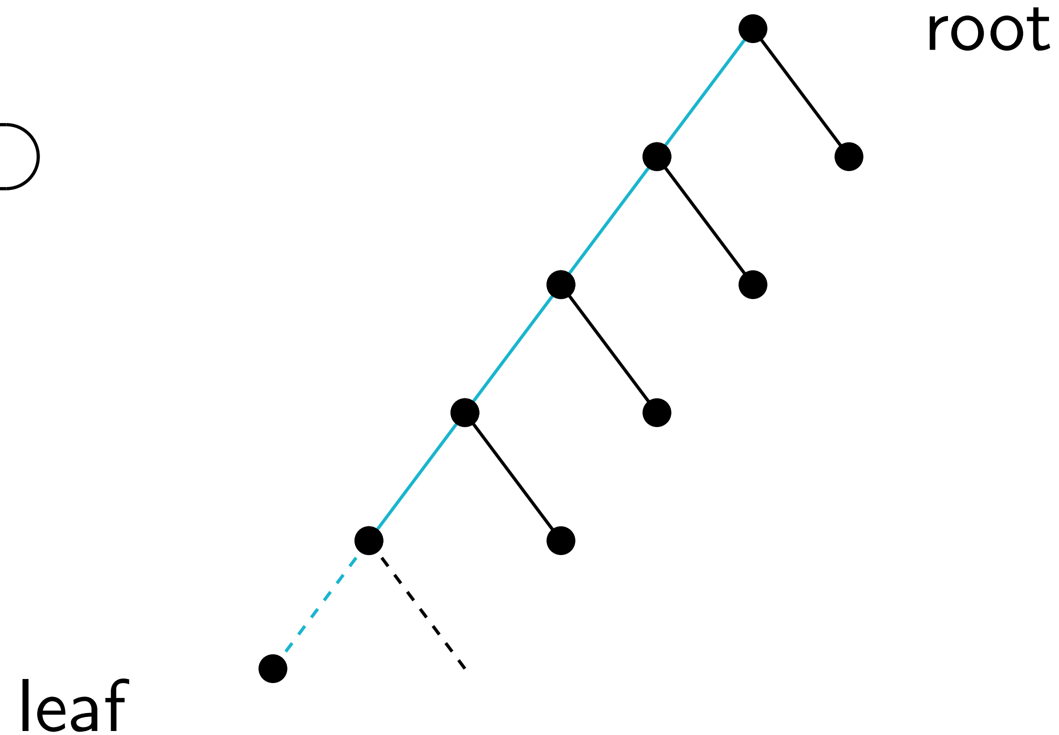


- Subtree priority does not take effect at any stage of DFS
  - Enforcing new layers

Layer Amount in T'

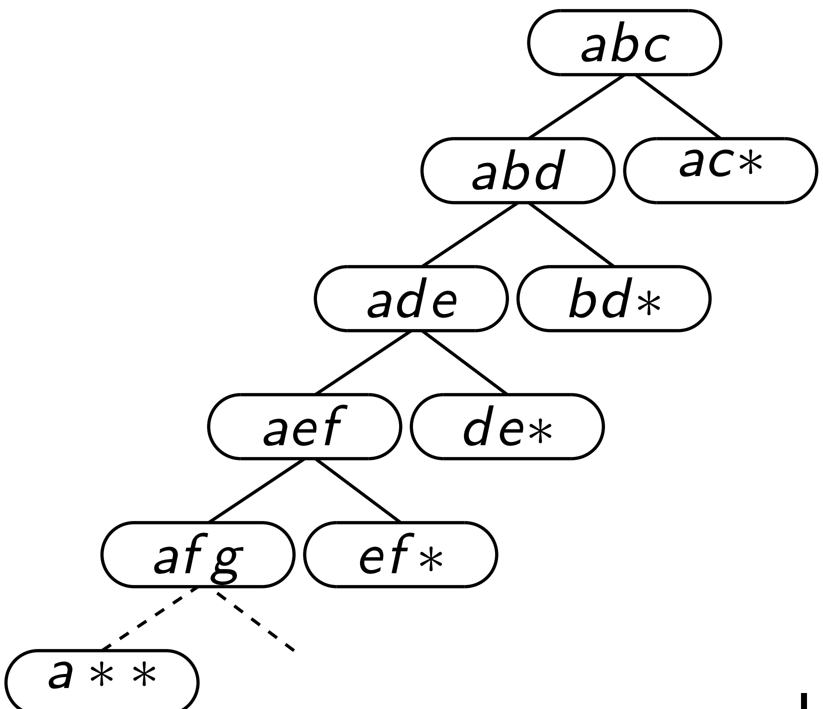
- In T', consider path from root to a leaf

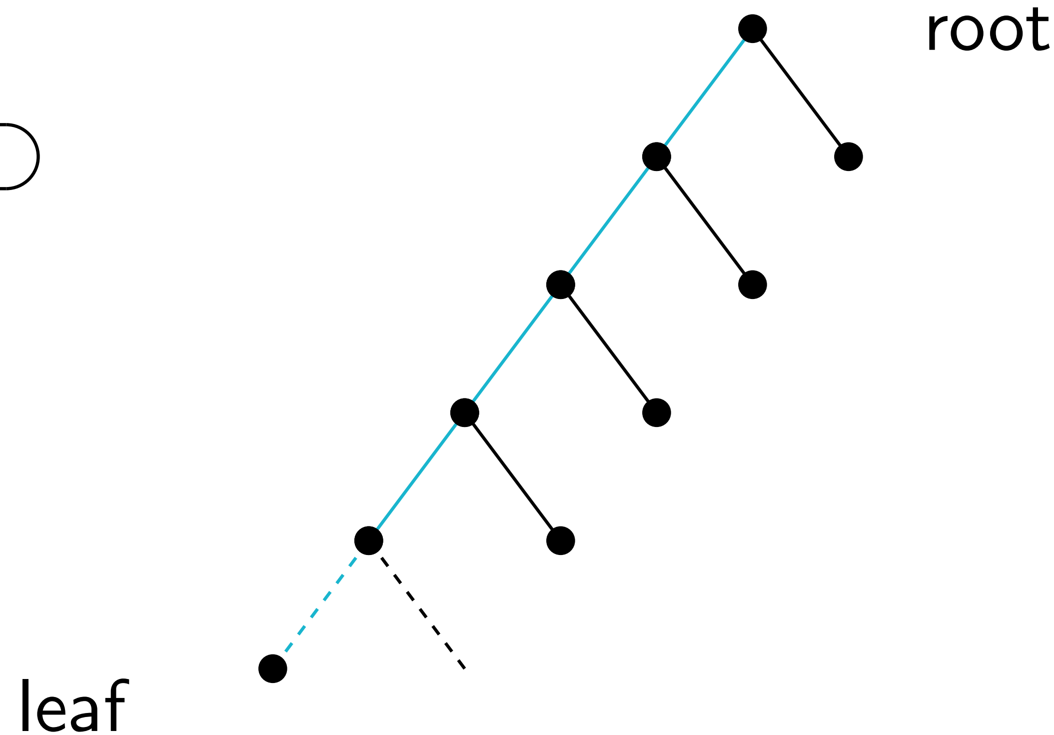




Layer Amount in T'

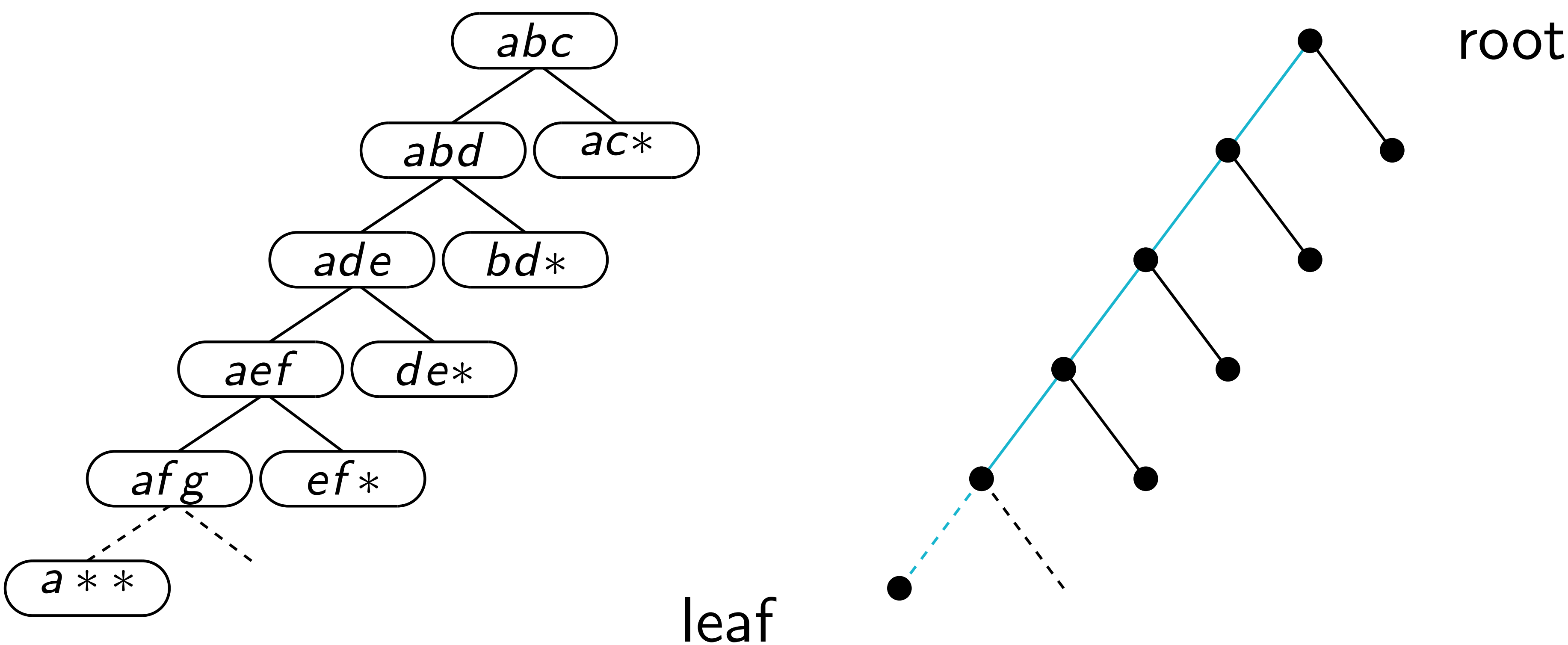
- In T', consider path from root to a leaf
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices





Layer Amount in T'

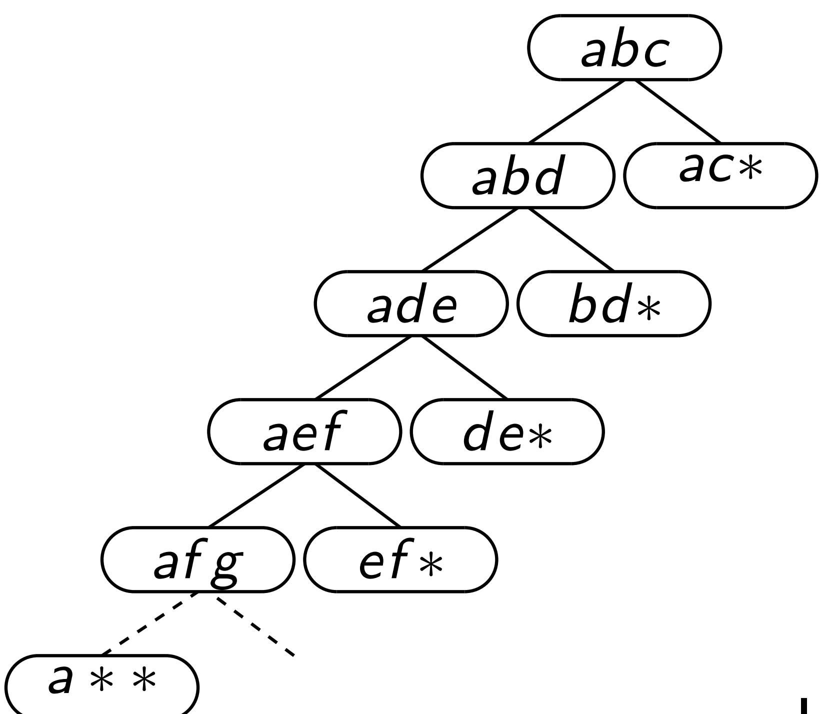
- In T', consider path from root to a leaf
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices
  - $T'_{v,w} := T'_v \cap T'_w$, $|T'_{v,w}| \leq 2$



Layer Amount in T'

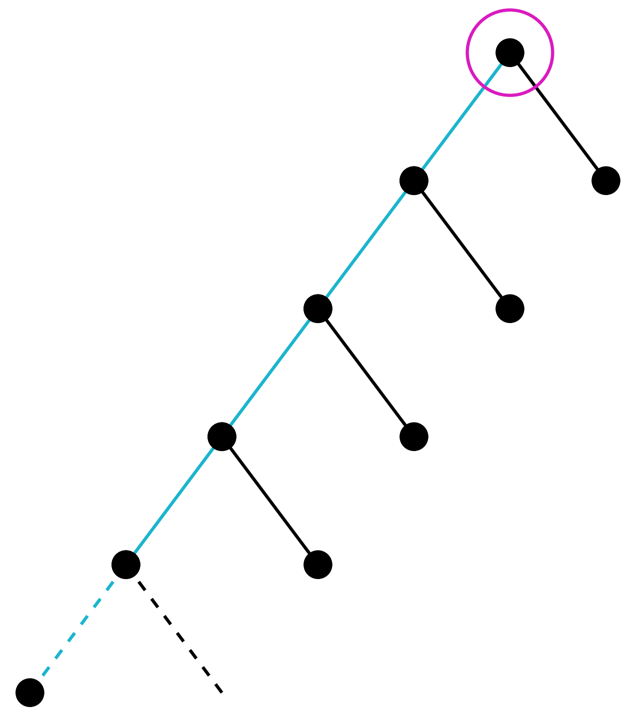
- In T', consider path from root to a leaf
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices
  - $T'_{v,w} := T'_v \cap T'_w$, $|T'_{v,w}| \leq 2$

$\texttt{Amount of layers occupied:} \geq 3$



DFS starts at root

root



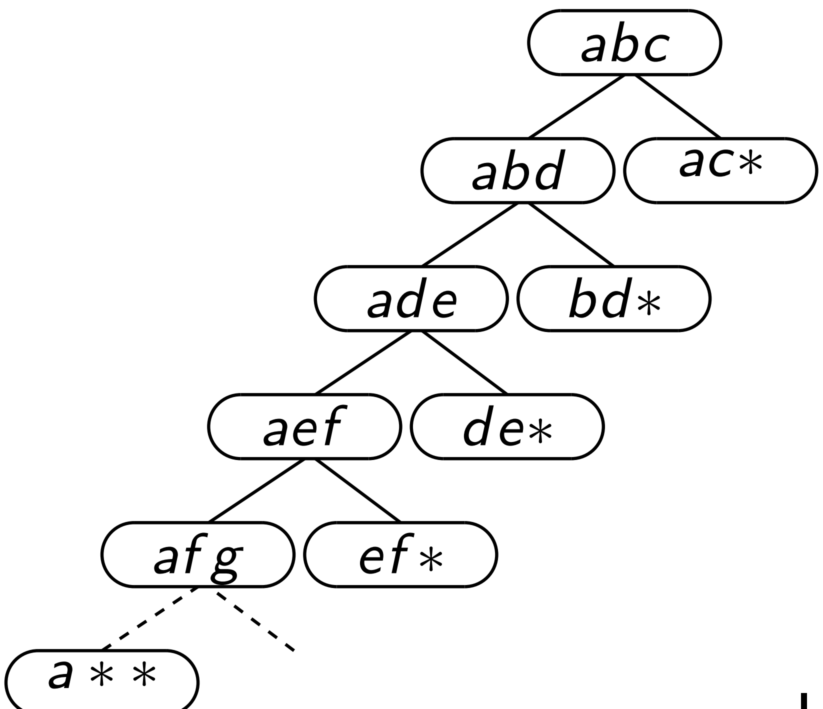
leaf

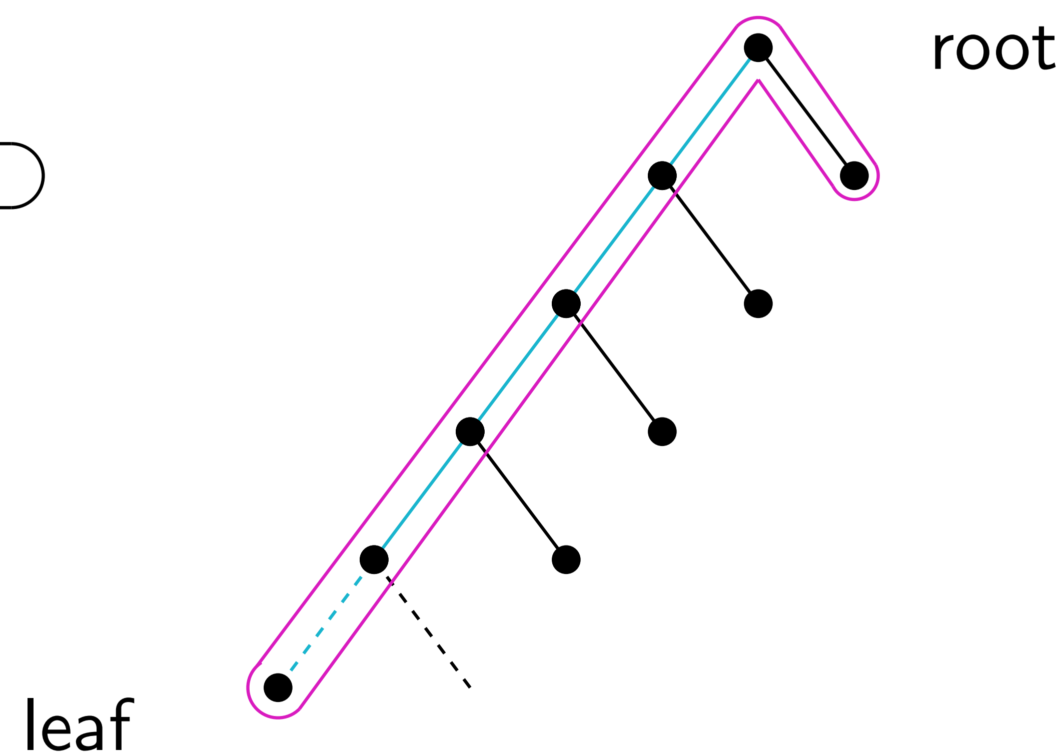
Layer Amount in T'

- In T', consider path from root to a leaf
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices
  - $T'_{v,w} := T'_v \cap T'_w$, $|T'_{v,w}| \leq 2$

$\texttt{Amount of layers occupied:} \geq 3$

Occupation of a

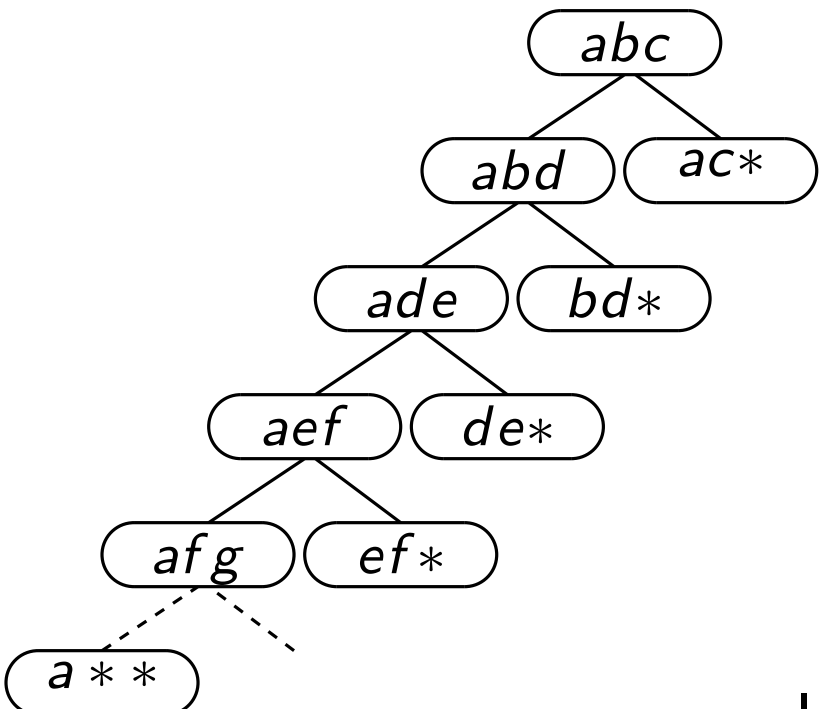


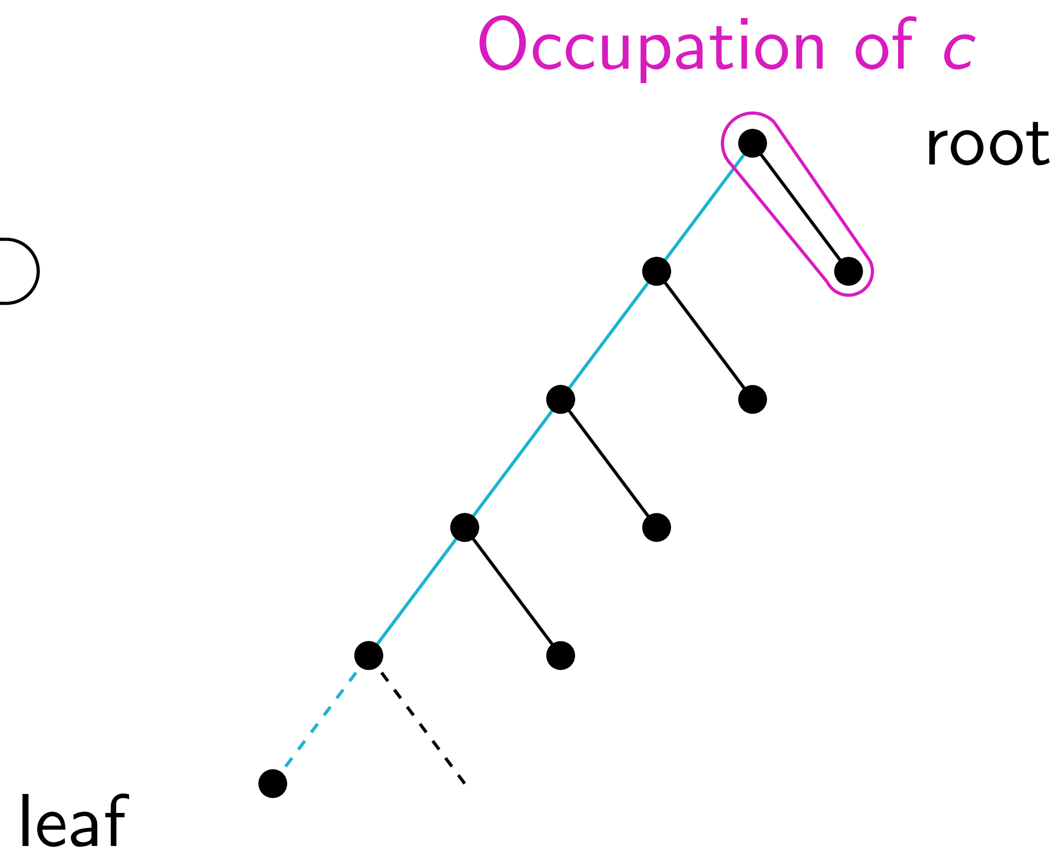


Layer Amount in T'

- In T', consider path from root to a leaf
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices
  - $T'_{v,w} := T'_v \cap T'_w$, $|T'_{v,w}| \leq 2$

$\texttt{Amount of layers occupied:} \geq 3$

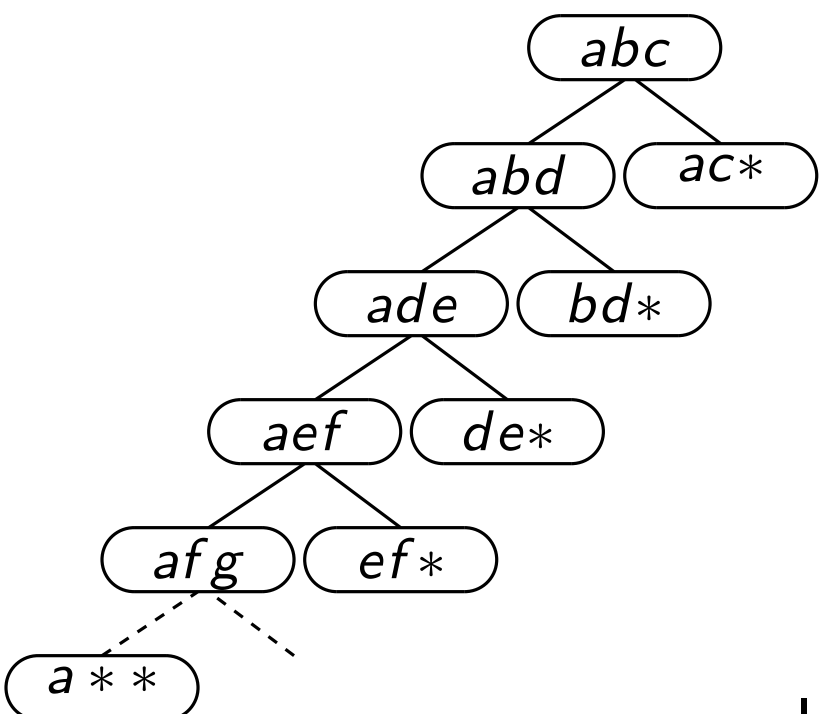


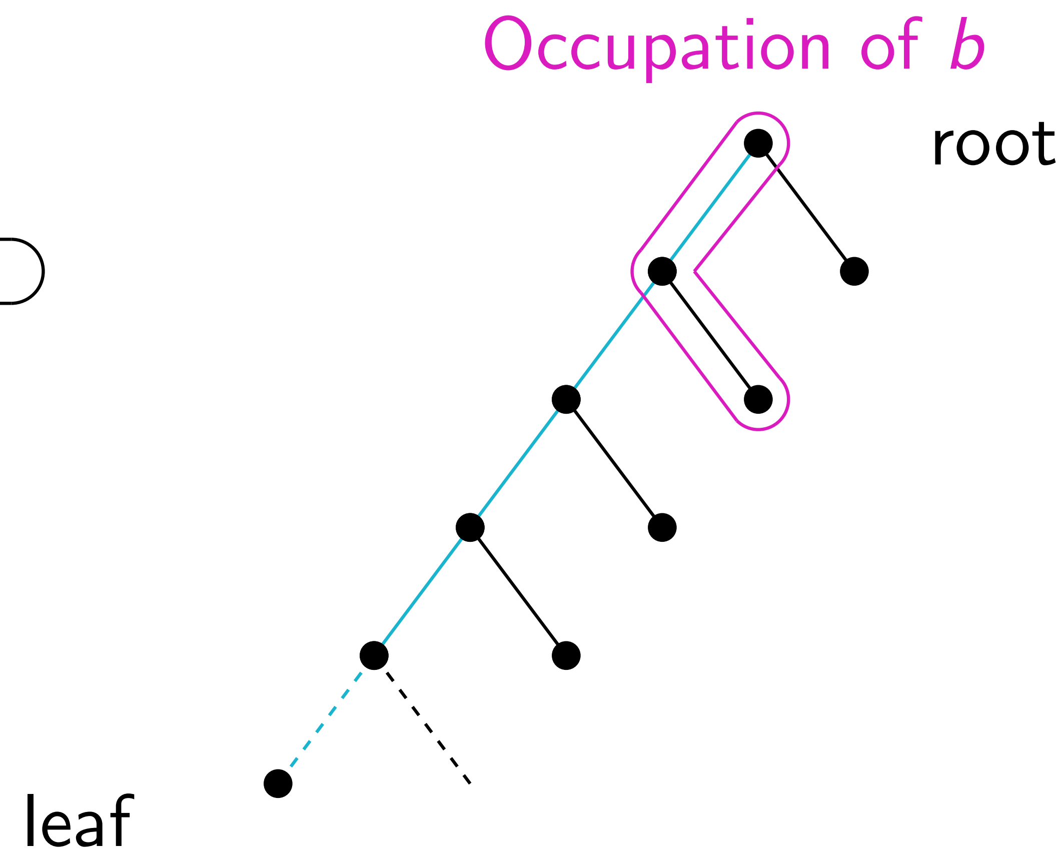


Layer Amount in T'

- In T', consider path from root to a leaf
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices
  - $T'_{v,w} := T'_v \cap T'_w$, $|T'_{v,w}| \leq 2$

`Amount of layers occupied:` ≥ 3

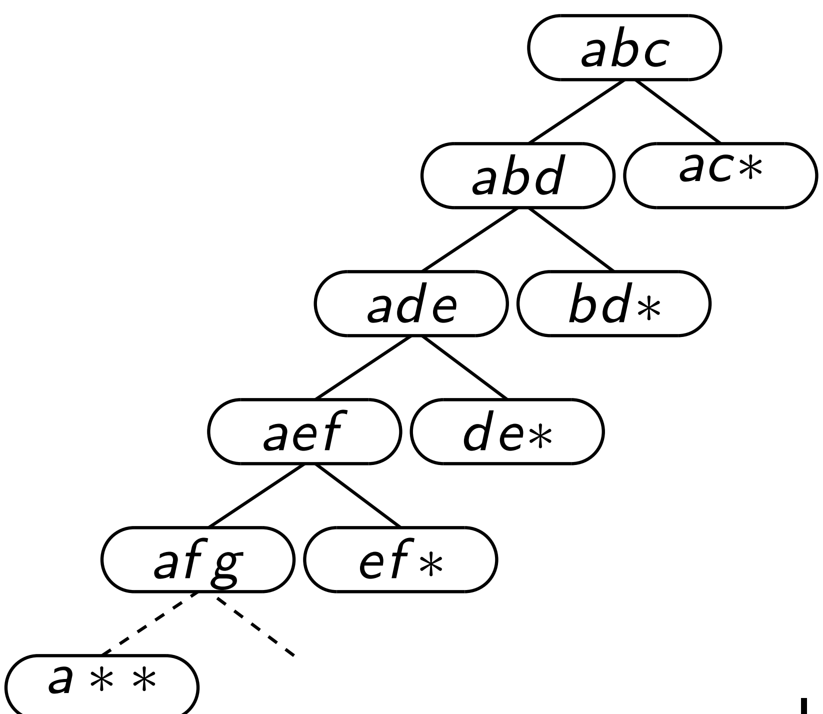




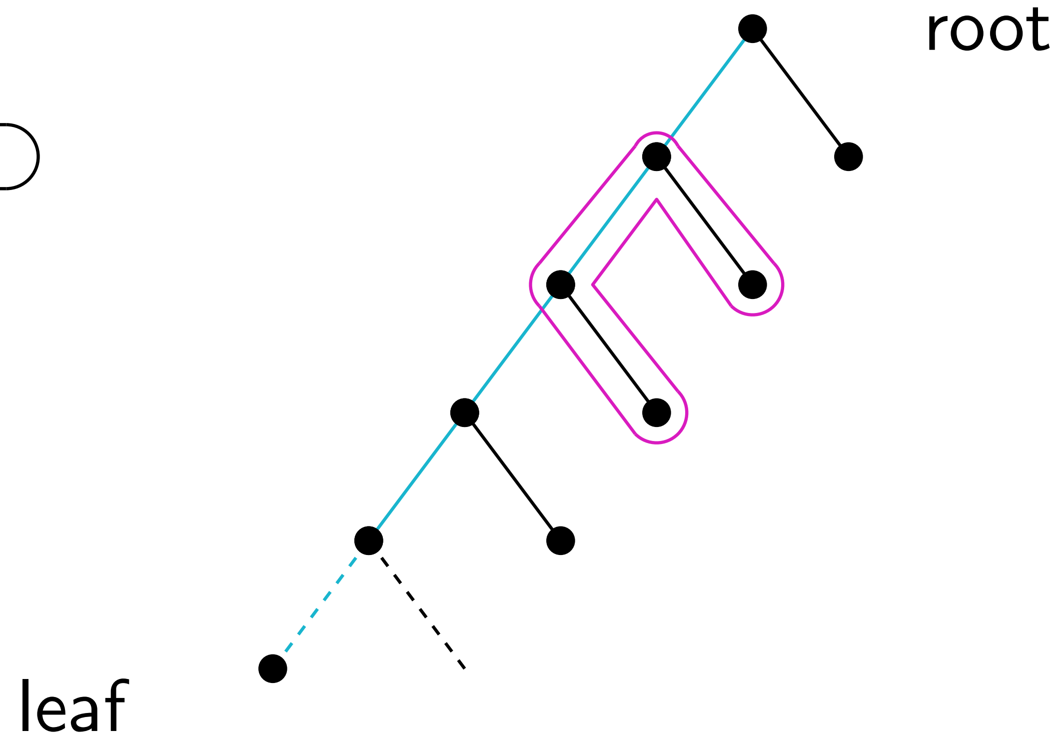
Layer Amount in T'

- In T', consider path from root to a leaf
  - `Query`$(v, T) = T'_v$ is a chain of vertices
  - $T'_{v,w} := T'_v \cap T'_w$, $|T'_{v,w}| \leq 2$

`Amount of layers occupied:` ≥ 4



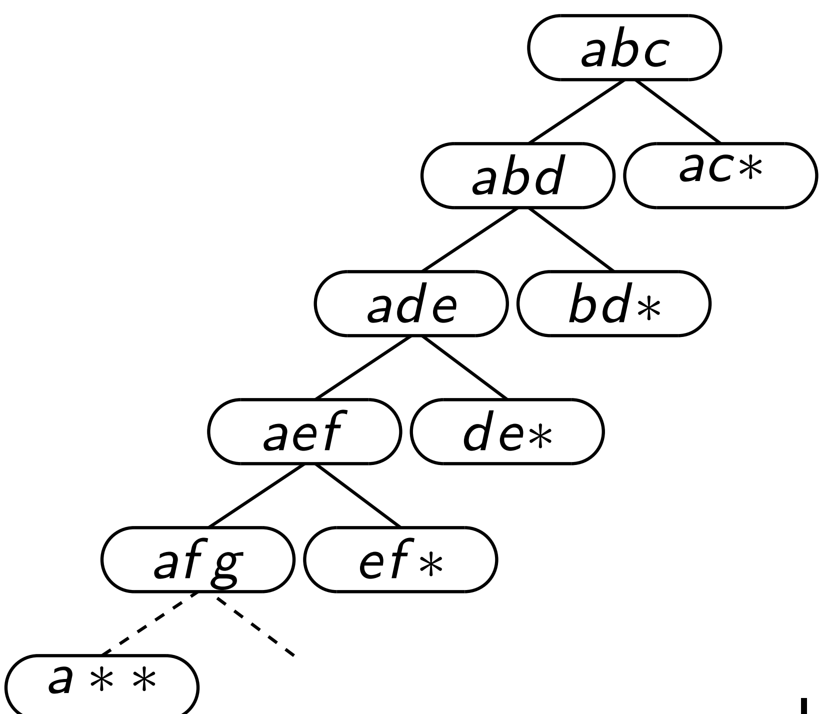
Occupation of d

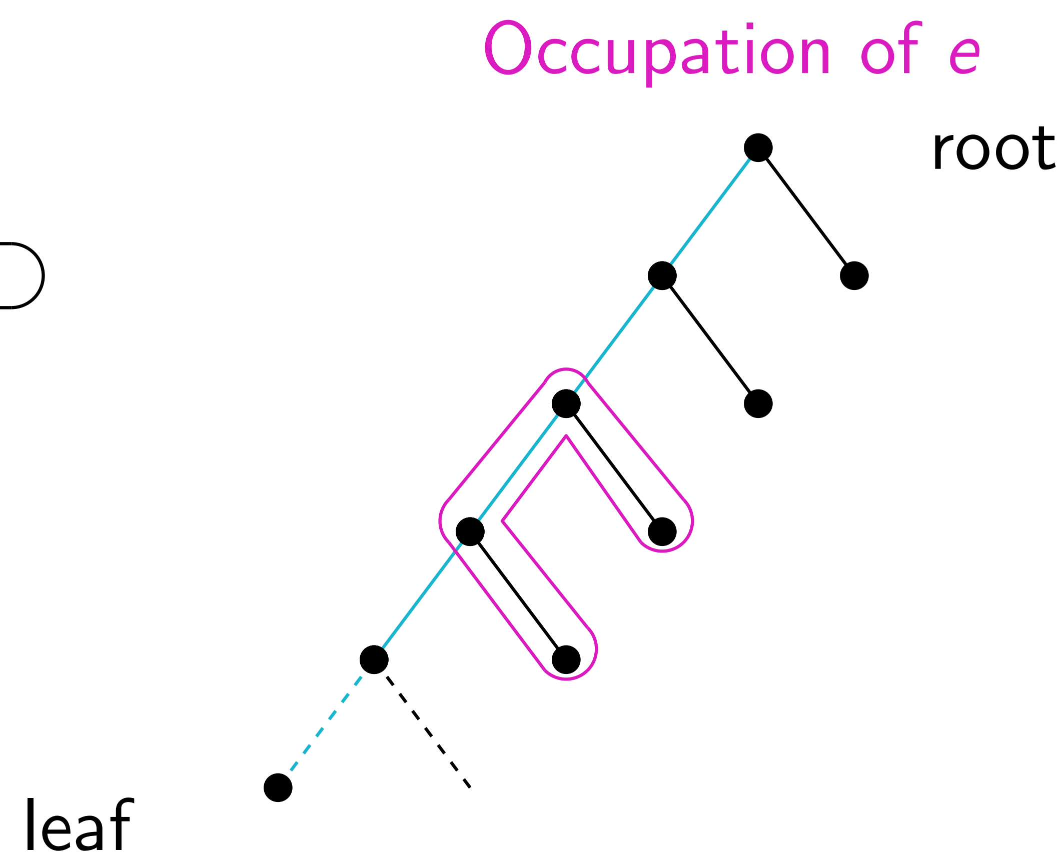


Layer Amount in T'

- In T', consider path from root to a leaf
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices
  - $T'_{v,w} := T'_v \cap T'_w$, $|T'_{v,w}| \leq 2$

`Amount of layers occupied:` ≥ 5

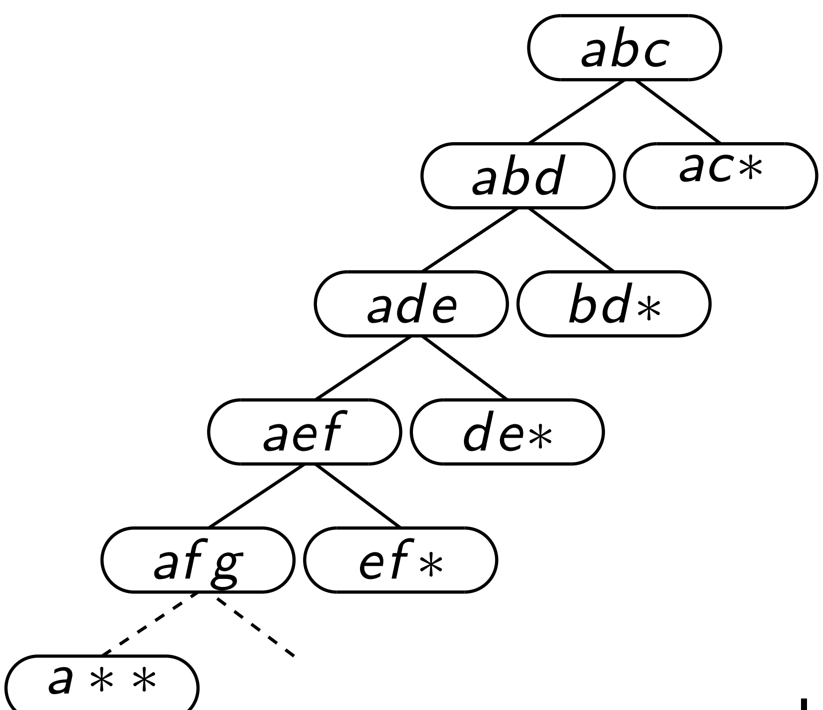


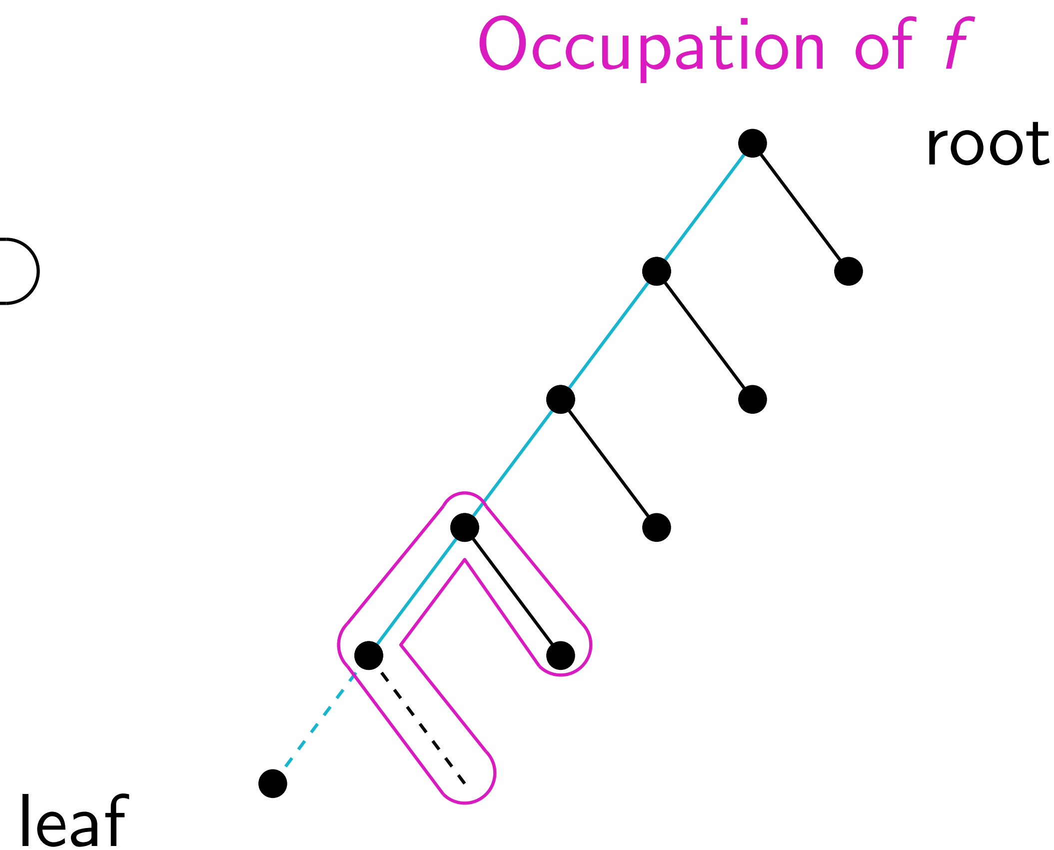


Layer Amount in T'

- In T', consider path from root to a leaf
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices
  - $T'_{v,w} := T'_v \cap T'_w$, $|T'_{v,w}| \leq 2$

`Amount of layers occupied:` ≥ 6

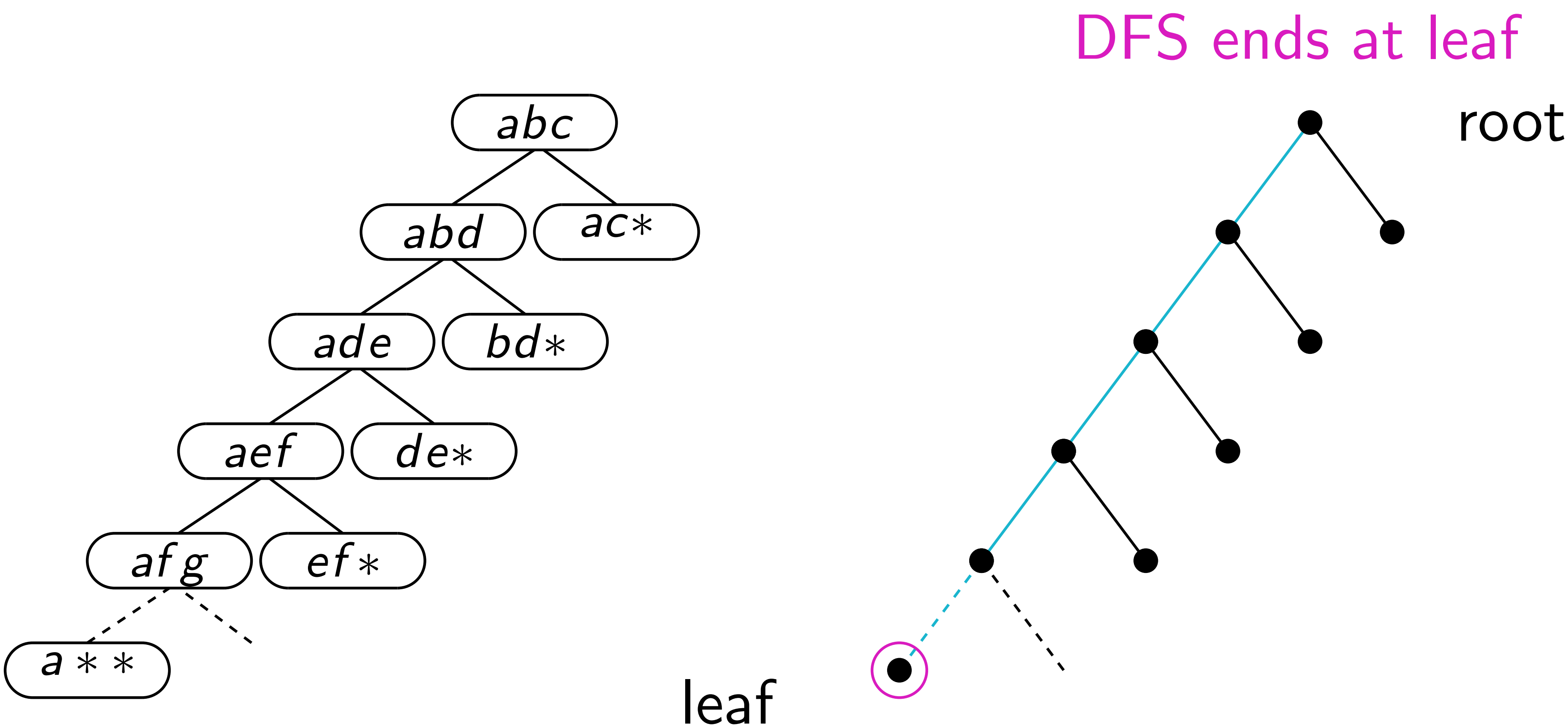




Layer Amount in T'

- In T', consider path from root to a leaf
  - $\texttt{Query}(v, T) = T'_v$ is a chain of vertices
  - $T'_{v,w} := T'_v \cap T'_w$, $|T'_{v,w}| \leq 2$

`Amount of layers occupied:` $\mathcal{O}(\log n')$



Total Amount of Layers

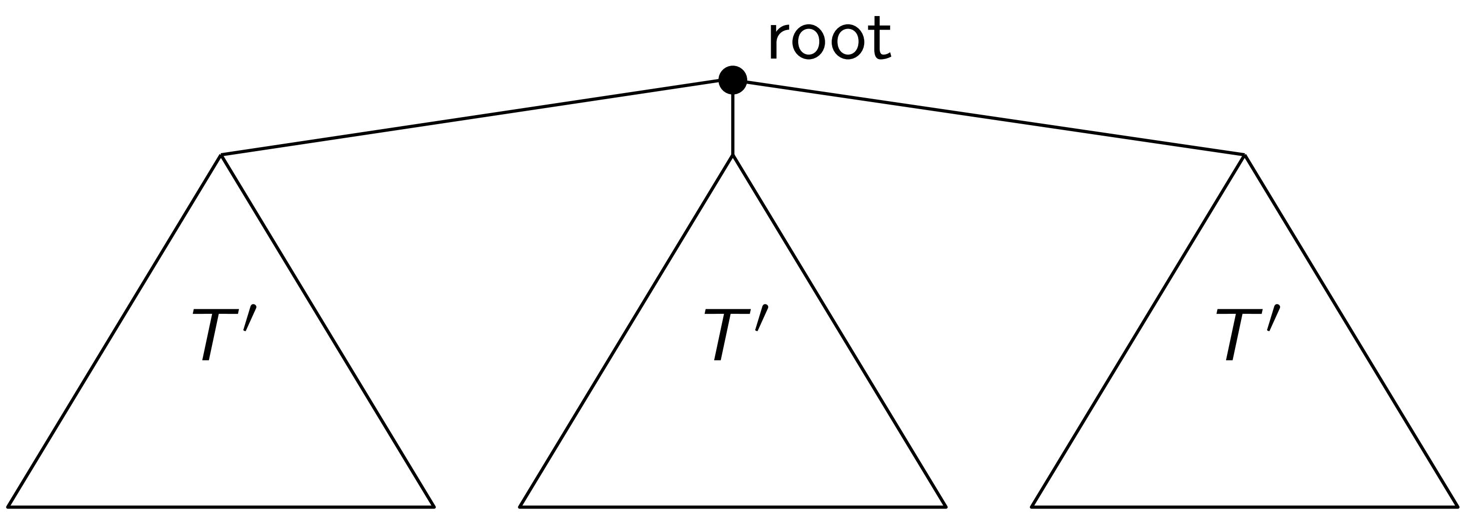
- Sibling subtrees of T share exactly one vertex of G

Total Amount of Layers

- Sibling subtrees of T share exactly one vertex of G
  - holds at every step of DFS over T (invariant)

Total Amount of Layers

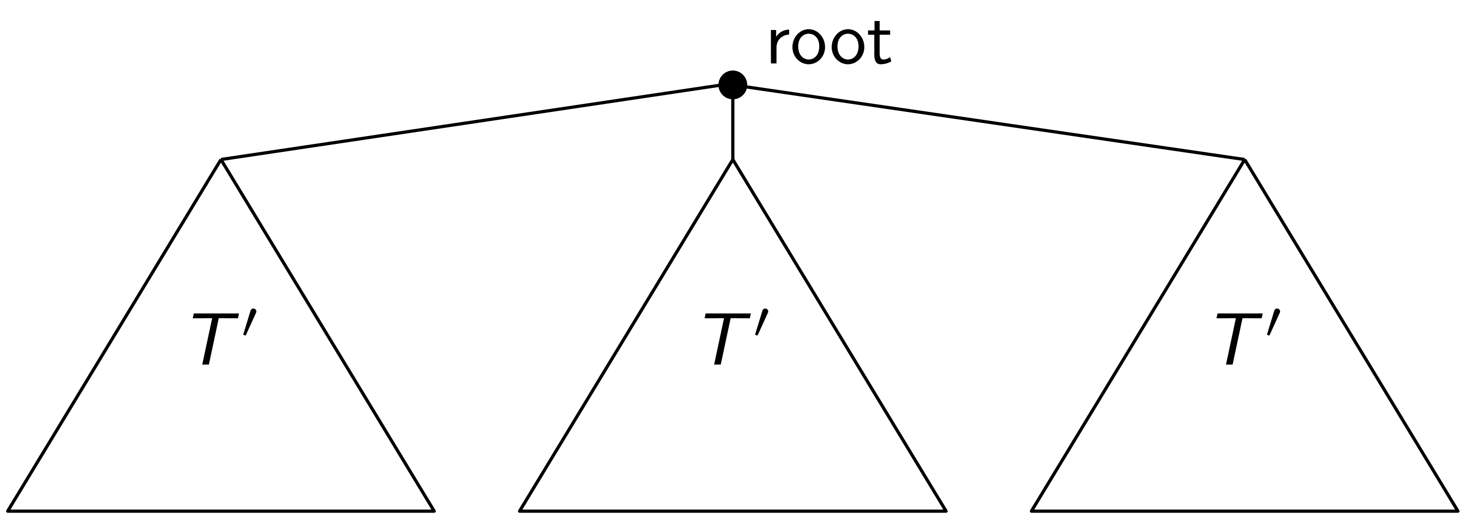
- Sibling subtrees of T share exactly one vertex of G
  - holds at every step of DFS over T (invariant)



  - If one subtree T' is fully drawn, constant amount of layers are still occupied and $\mathcal{O}(\log n)$ layers become free

Total Amount of Layers

- Sibling subtrees of T share exactly one vertex of G
  - holds at every step of DFS over T (invariant)



- If one subtree T' is fully drawn, constant amount of layers are still occupied and $\mathcal{O}(\log n)$ layers become free
- Height of drawing of T is bound by the height of largest subtree

Worst Case Subtree Structure

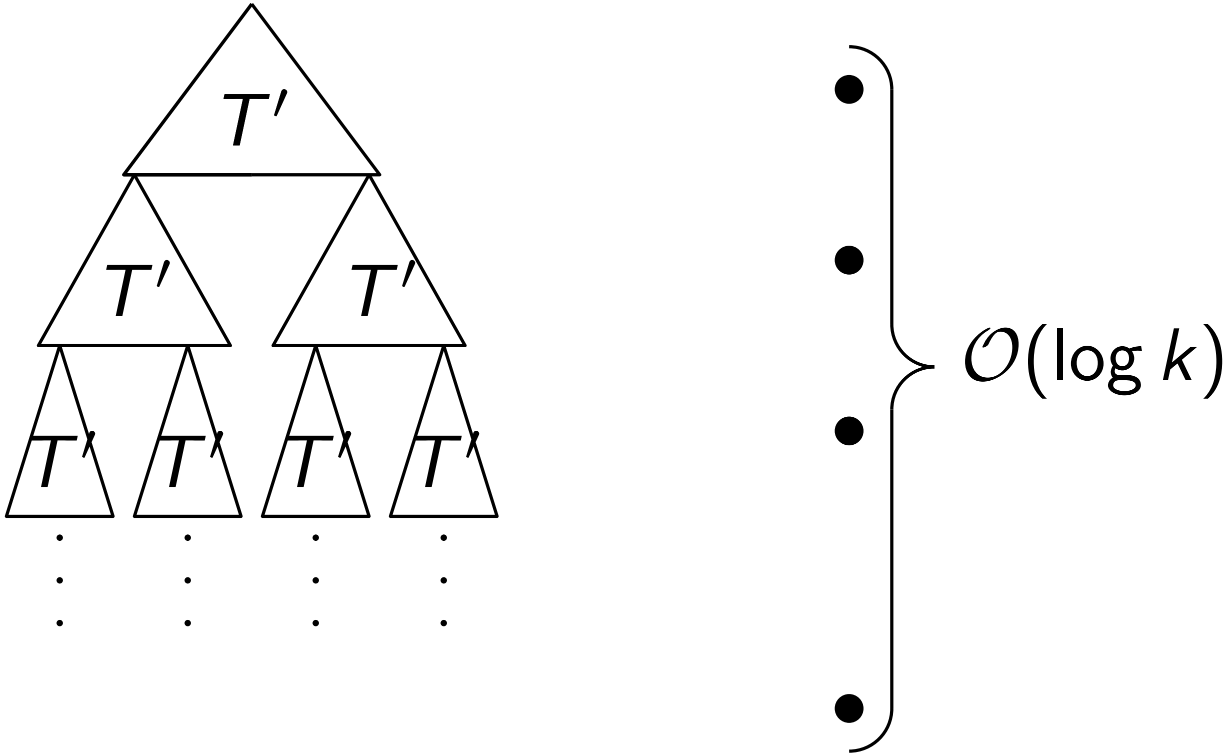
- T' complete binary tree with height h'

Worst Case Subtree Structure

- T' complete binary tree with height h'
- k times T' appended in binary structure

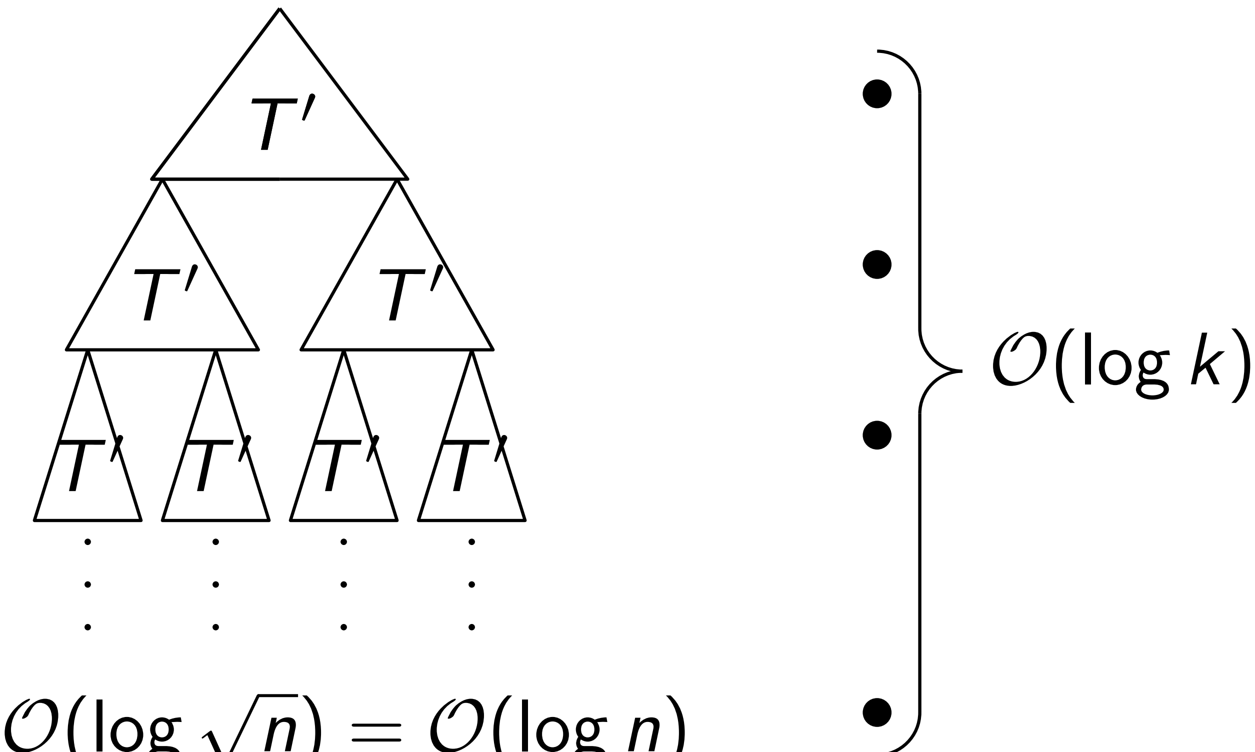
Worst Case Subtree Structure

- T' complete binary tree with height h'
- k times T' appended in binary structure
- $h' \in \mathcal{O}\left(\log \frac{n}{k}\right)$



Worst Case Subtree Structure

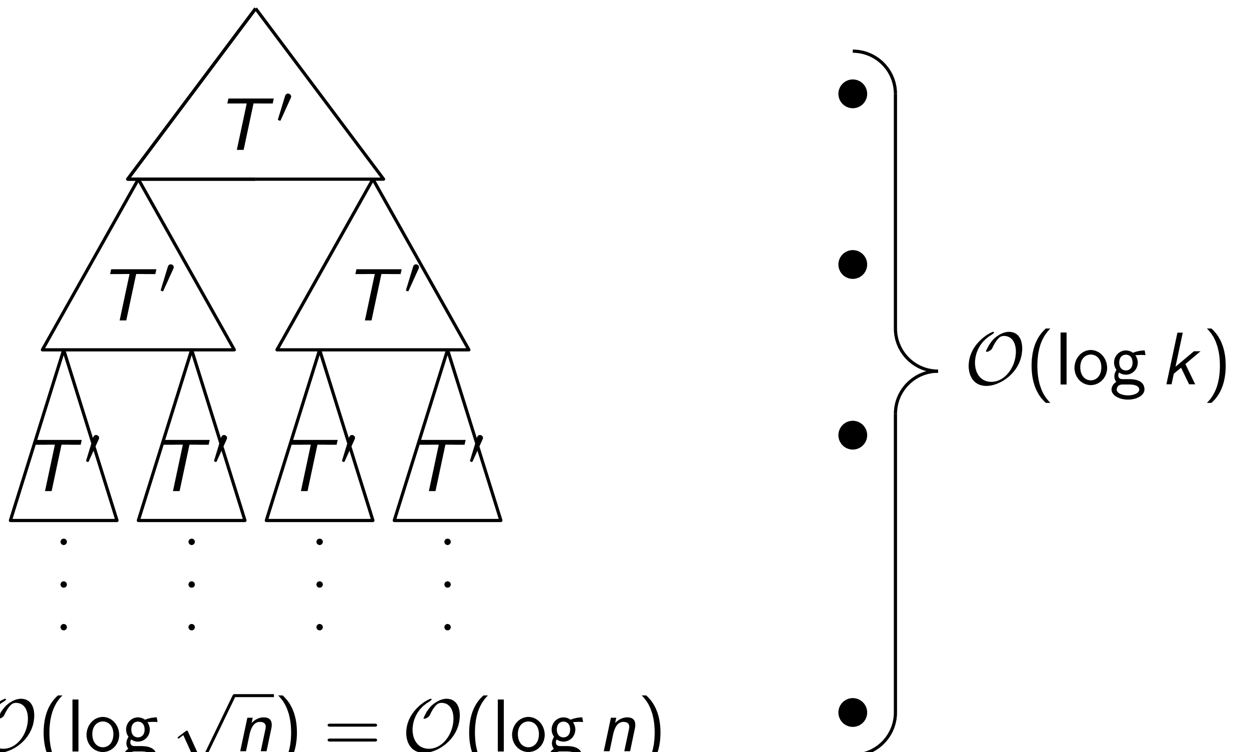
- T' complete binary tree with height h'
- k times T' appended in binary structure
- $h' \in \mathcal{O}\left(\log \frac{n}{k}\right)$



- $k := \sqrt{n} \Rightarrow h' \in \mathcal{O}(\log \sqrt{n}) = \mathcal{O}(\log n)$

Worst Case Subtree Structure

- T' complete binary tree with height h'
- k times T' appended in binary structure
- $h' \in \mathcal{O}\left(\log \frac{n}{k}\right)$



- $k := \sqrt{n} \Rightarrow h' \in \mathcal{O}(\log \sqrt{n}) = \mathcal{O}(\log n)$
- Total height in $\mathcal{O}(\log n) \cdot \mathcal{O}(\log n) = \mathcal{O}(\log^2 n)$

Results For Outerplanar Graphs

- For every edge there is a column in the box drawing

Results For Outerplanar Graphs

- For every edge there is a column in the box drawing
  - Width of drawing w is bounded by $\mathcal{O}(m) = \mathcal{O}(n)$

Results For Outerplanar Graphs

- For every edge there is a column in the box drawing
  - Width of drawing w is bounded by $\mathcal{O}(m) = \mathcal{O}(n)$
- Height of box drawing $h = \delta \cdot \#\text{layers}$

Results For Outerplanar Graphs

- For every edge there is a column in the box drawing
  - Width of drawing w is bounded by $\mathcal{O}(m) = \mathcal{O}(n)$
- Height of box drawing $h = \delta \cdot \#\text{layers}$
  - $\delta \cdot \mathcal{O}(\log^2 n)$ in the worst case due to DFS strategy

Results For Outerplanar Graphs

- For every edge there is a column in the box drawing
  - Width of drawing w is bounded by $\mathcal{O}(m) = \mathcal{O}(n)$
- Height of box drawing $h = \delta \cdot \#$layers
  - $\delta \cdot \mathcal{O}(\log^2 n)$ in the worst case due to DFS strategy
- Box drawing transferable to polyline drawing with two bends in asymptotically same area bounds

Results For Outerplanar Graphs

- For every edge there is a column in the box drawing
  - Width of drawing w is bounded by $\mathcal{O}(m) = \mathcal{O}(n)$
- Height of box drawing $h = \delta \cdot \#\text{layers}$
  - $\delta \cdot \mathcal{O}(\log^2 n)$ in the worst case due to DFS strategy
- Box drawing transferable to polyline drawing with two bends in asymptotically same area bounds
- If $\delta := n$, then the height is in $\mathcal{O}(n \log^2 n)$

Results For Outerplanar Graphs

- For every edge there is a column in the box drawing
  - Width of drawing w is bounded by $\mathcal{O}(m) = \mathcal{O}(n)$
- Height of box drawing $h = \delta \cdot \#\text{layers}$
  - $\delta \cdot \mathcal{O}(\log^2 n)$ in the worst case due to DFS strategy
- Box drawing transferable to polyline drawing with two bends in asymptotically same area bounds
- If $\delta := n$, then the height is in $\mathcal{O}(n \log^2 n)$
  - Longest polyline with two bends is bounded by $\max\{w, h\} = \mathcal{O}(n \log^2 n)$

Results For Outerplanar Graphs

- For every edge there is a column in the box drawing
  - Width of drawing w is bounded by $\mathcal{O}(m) = \mathcal{O}(n)$
- Height of box drawing $h = \delta \cdot \#\text{layers}$
  - $\delta \cdot \mathcal{O}(\log^2 n)$ in the worst case due to DFS strategy
- Box drawing transferable to polyline drawing with two bends in asymptotically same area bounds
- If $\delta := n$, then the height is in $\mathcal{O}(n \log^2 n)$
  - Longest polyline with two bends is bounded by $\max\{w, h\} = \mathcal{O}(n \log^2 n)$
  - $r \in \mathcal{O}(\frac{n \log^2 n}{n}) = \mathcal{O}(\log^2 n)$

Results For Outerplanar Graphs

- For every edge there is a column in the box drawing
  - Width of drawing w is bounded by $\mathcal{O}(m) = \mathcal{O}(n)$
- Height of box drawing $h = \delta \cdot \#\text{layers}$
  - $\delta \cdot \mathcal{O}(\log^2 n)$ in the worst case due to DFS strategy
- Box drawing transferable to polyline drawing with two bends in asymptotically same area bounds
- If $\delta := n$, then the height is in $\mathcal{O}(n \log^2 n)$
  - Longest polyline with two bends is bounded by $\max\{w, h\} = \mathcal{O}(n \log^2 n)$
  - $r \in \mathcal{O}(\frac{n \log^2 n}{n}) = \mathcal{O}(\log^2 n)$
- Any outerplanar graph G admits a polyline drawing in $\mathcal{O}(n^2 \log^2 n)$ with $r \in \mathcal{O}(\log^2 n)$ and two bends per edge

Series-Parallel Graphs

- Approach II is applicable to series-parallel graphs!

Series-Parallel Graphs

- Approach II is applicable to series-parallel graphs!
- Degree of tree decomposition T arbitrary
- In T, find largest maximal outerplanar subgraph G', T' is tree of degree 3

Series-Parallel Graphs

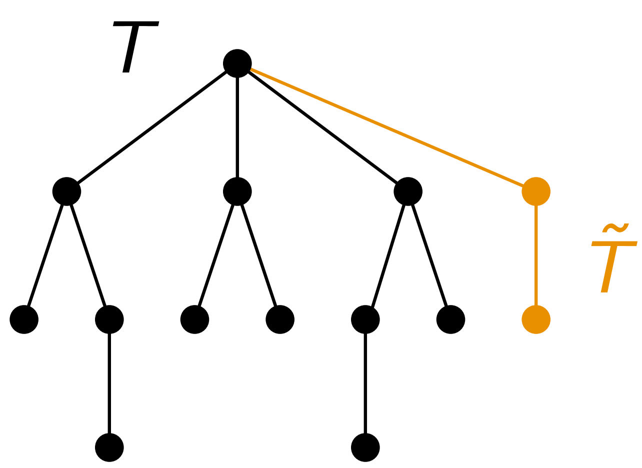
- Approach II is applicable to series-parallel graphs!
- Degree of tree decomposition T arbitrary
- In T, find largest maximal outerplanar subgraph G', T' is tree of degree 3
- Traverse T' and draw a box drawing $B_{G'}$ as before

Series-Parallel Graphs

- Approach II is applicable to series-parallel graphs!
- Degree of tree decomposition T arbitrary
- In T, find largest maximal outerplanar subgraph G', T' is tree of degree 3
- Traverse T' and draw a box drawing $B_{G'}$ as before
- Insert the missing subtrees out of $T \setminus T'$ into $B_{G'}$

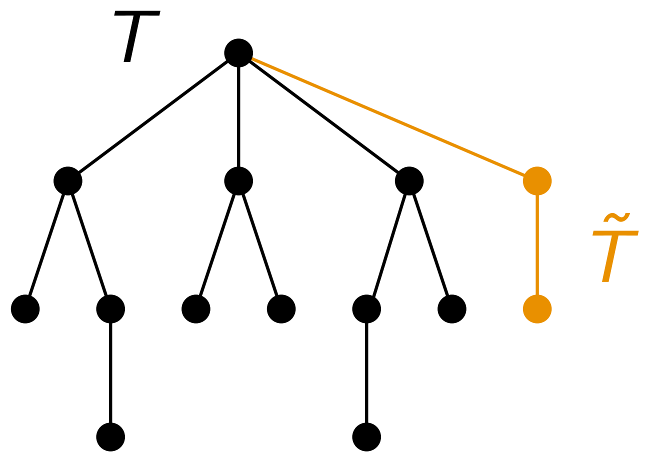
Series-Parallel Graphs

- Approach II is applicable to series-parallel graphs!
- Degree of tree decomposition T arbitrary
- In T, find largest maximal outerplanar subgraph G', T' is tree of degree 3
- Traverse T' and draw a box drawing $B_{G'}$ as before
- Insert the missing subtrees out of $T \setminus T'$ into $B_{G'}$



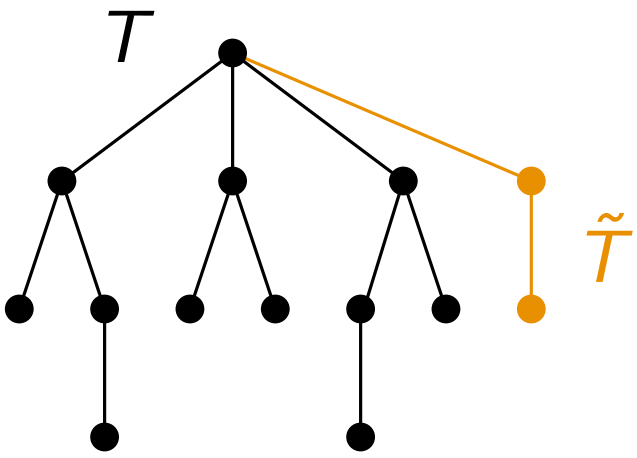
Subtree Insertions out of $T \setminus T'$ into $B_{G'}$

- Let $\tilde{T}$ be out of $T \setminus T'$



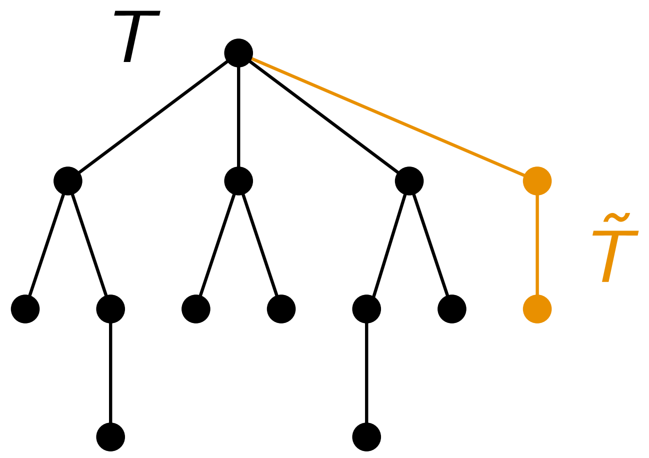
Subtree Insertions out of $T \setminus T'$ into $B_{G'}$

- Let $\tilde{T}$ be out of $T \setminus T'$
- Parent of $\tilde{T}$ is in T' and already drawn in $B_{G'}$
- Root of $\tilde{T}$ shares exactly two vertices a, b with its parent in T'

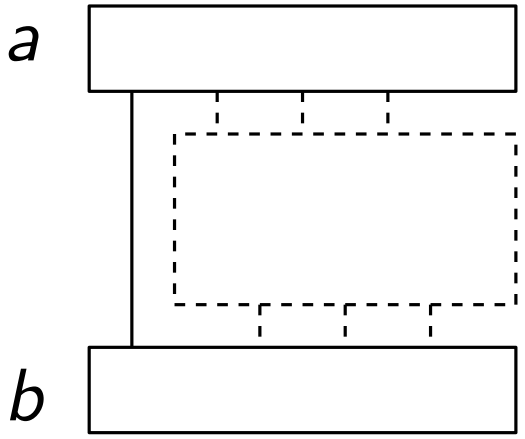


Subtree Insertions out of $T \setminus T'$ into $B_{G'}$

- Let $\tilde{T}$ be out of $T \setminus T'$
- Parent of $\tilde{T}$ is in T' and already drawn in $B_{G'}$
- Root of $\tilde{T}$ shares exactly two vertices a, b with its parent in T'

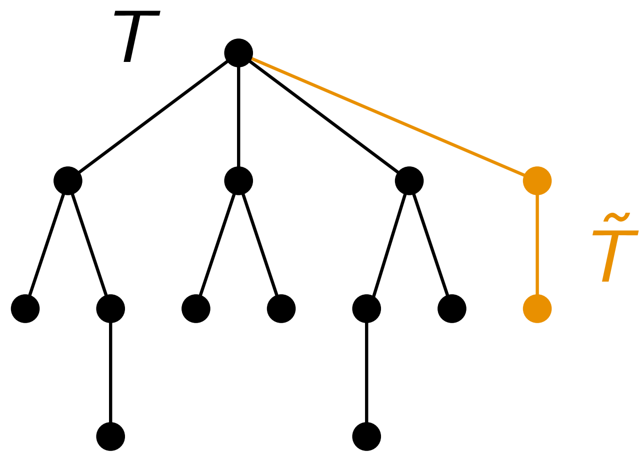


- Since G' is the largest maximal outerplanar subgraph, there already exists a drawn subtree between a and b

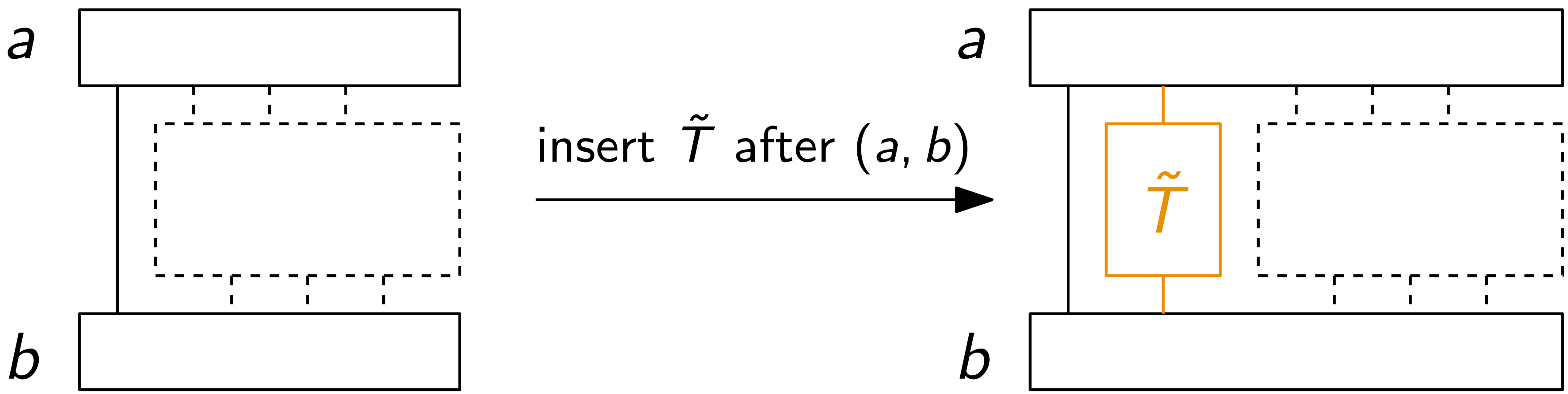


Subtree Insertions out of $T \setminus T'$ into $B_{G'}$

- Let $\tilde{T}$ be out of $T \setminus T'$
- Parent of $\tilde{T}$ is in T' and already drawn in $B_{G'}$
- Root of $\tilde{T}$ shares exactly two vertices a, b with its parent in T'



- Since G' is the largest maximal outerplanar subgraph, there already exists a drawn subtree between a and b

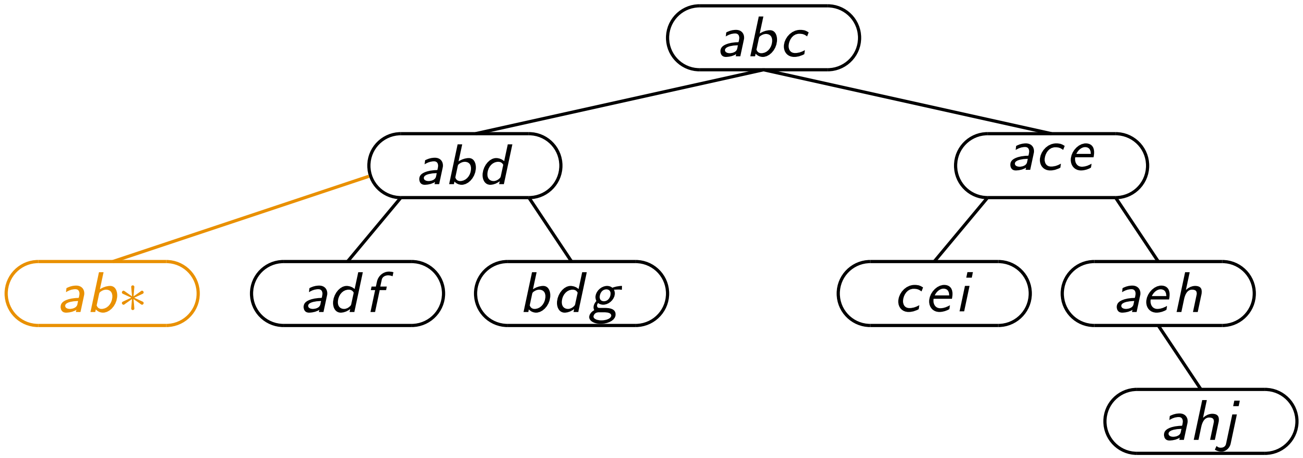


- Does not alter the area bounds of $B_{G'}$ asymptotically!

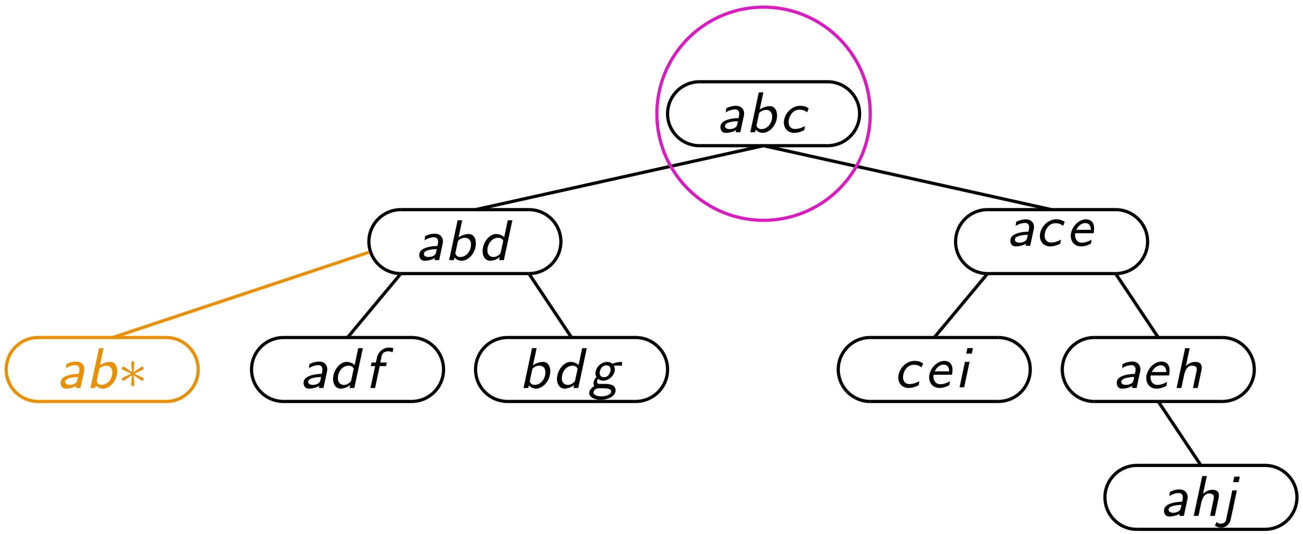
Results for Series-Parallel Graphs

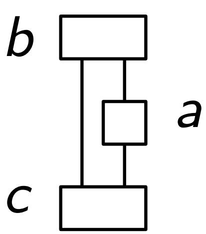
- Since the area bounds stay the same, the results of outerplanar graphs also hold for series-parallel graphs
- Any series-parallel graph G admits a polyline drawing in $\mathcal{O}(n^2 \log^2 n)$ with $r \in \mathcal{O}(\log^2 n)$ and two bends per edge

Example drawing

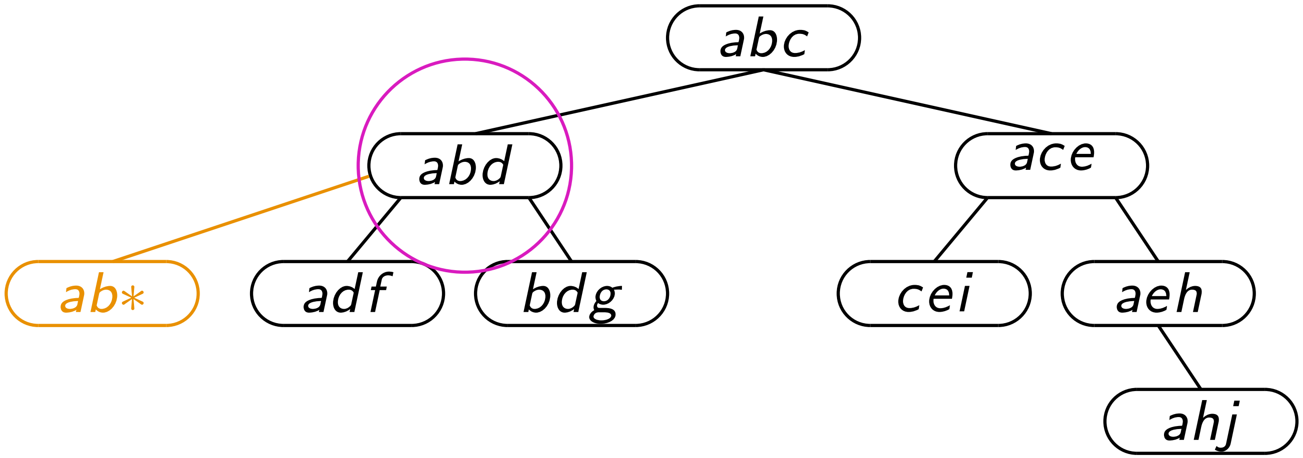


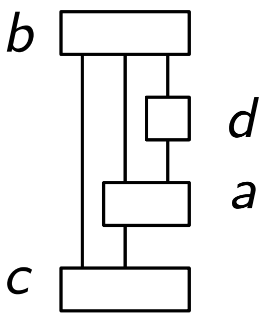
Example drawing



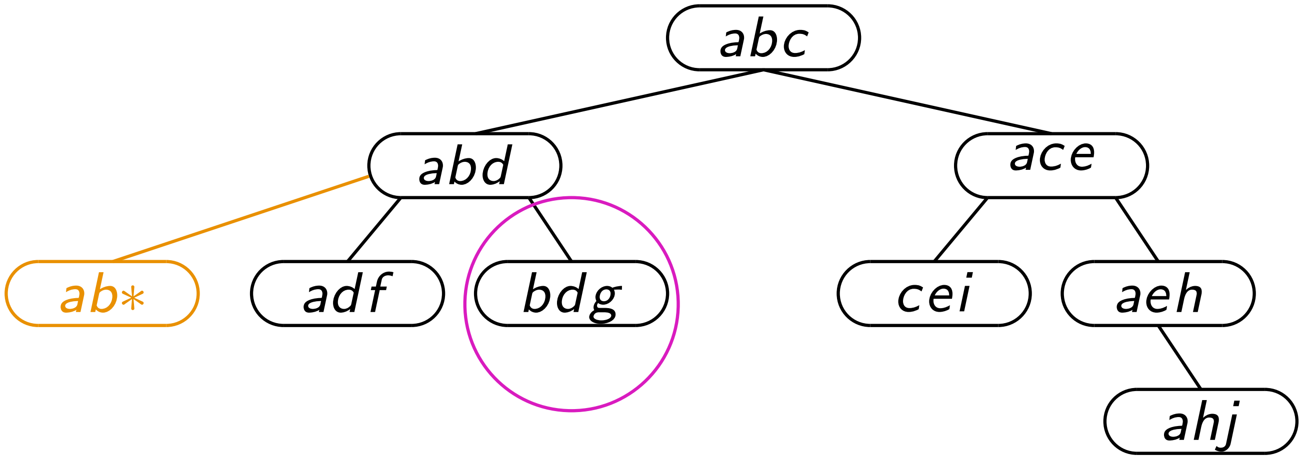


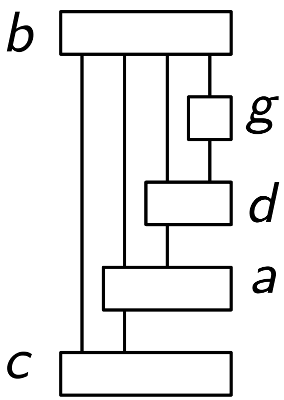
Example drawing



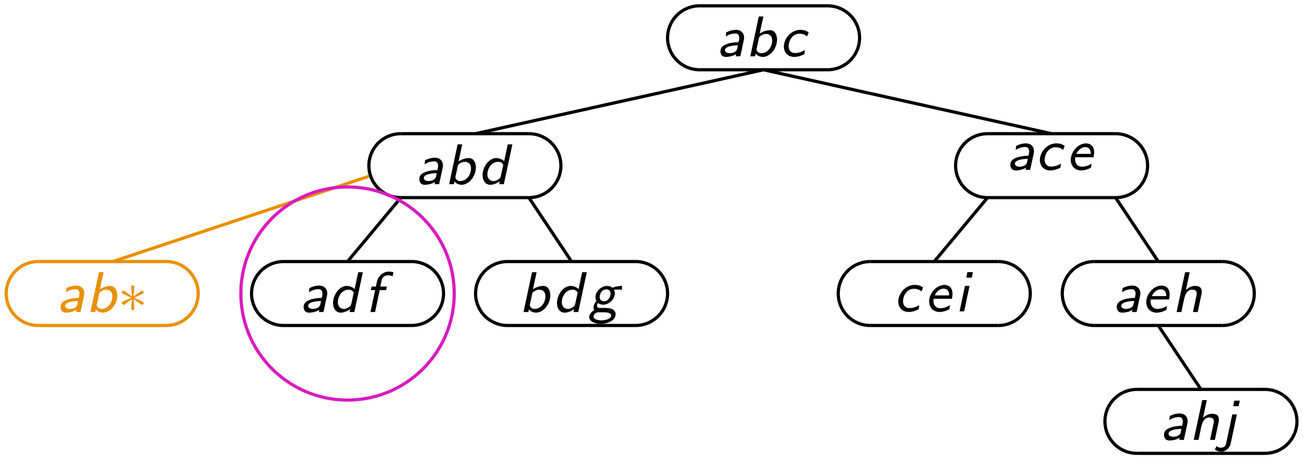


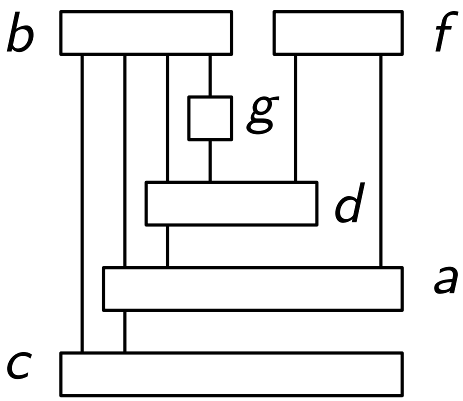
Example drawing



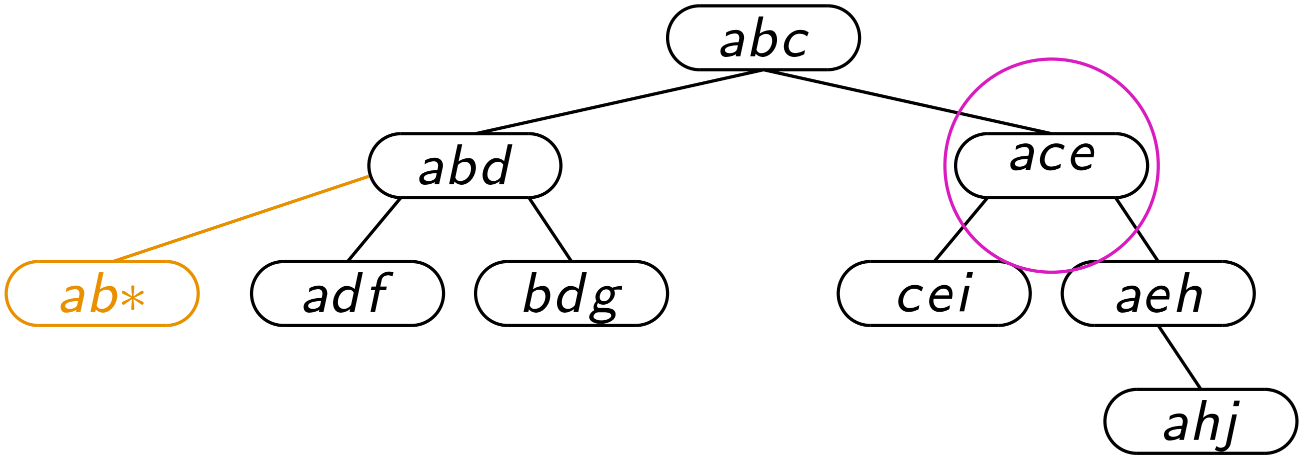


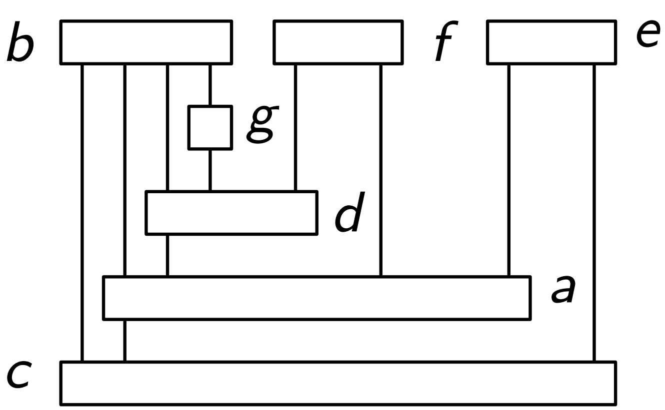
Example drawing



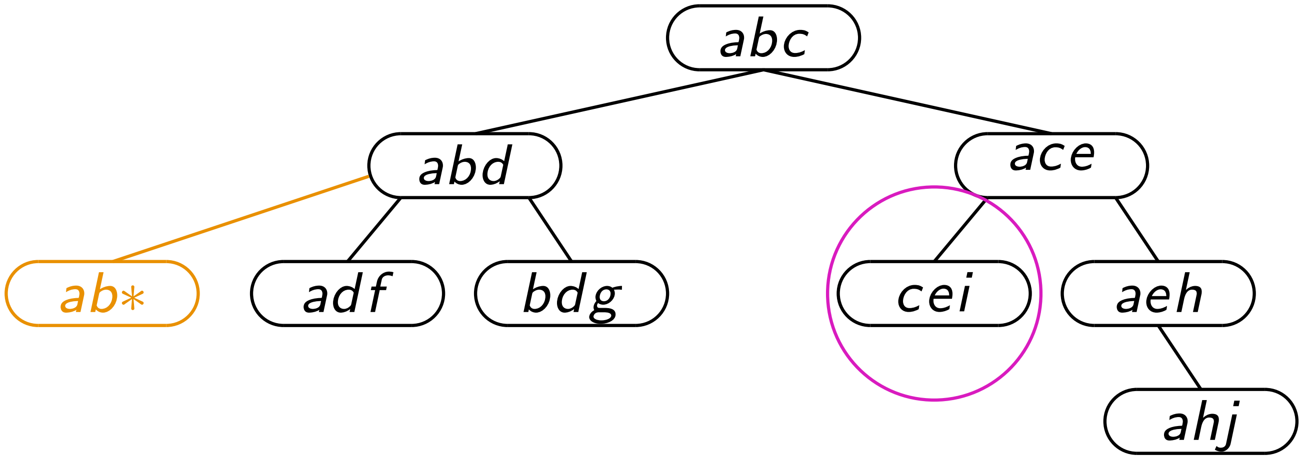


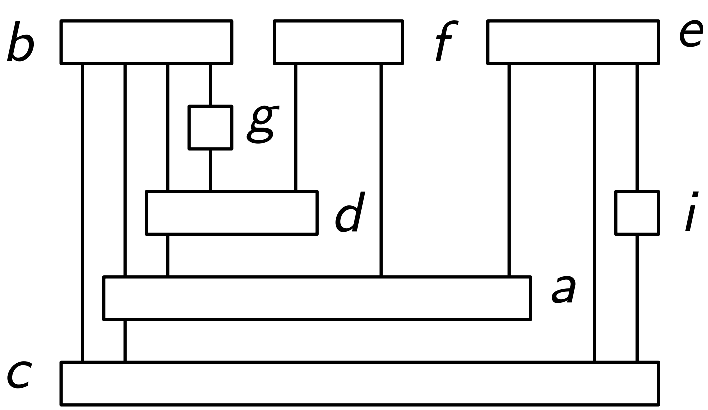
Example drawing



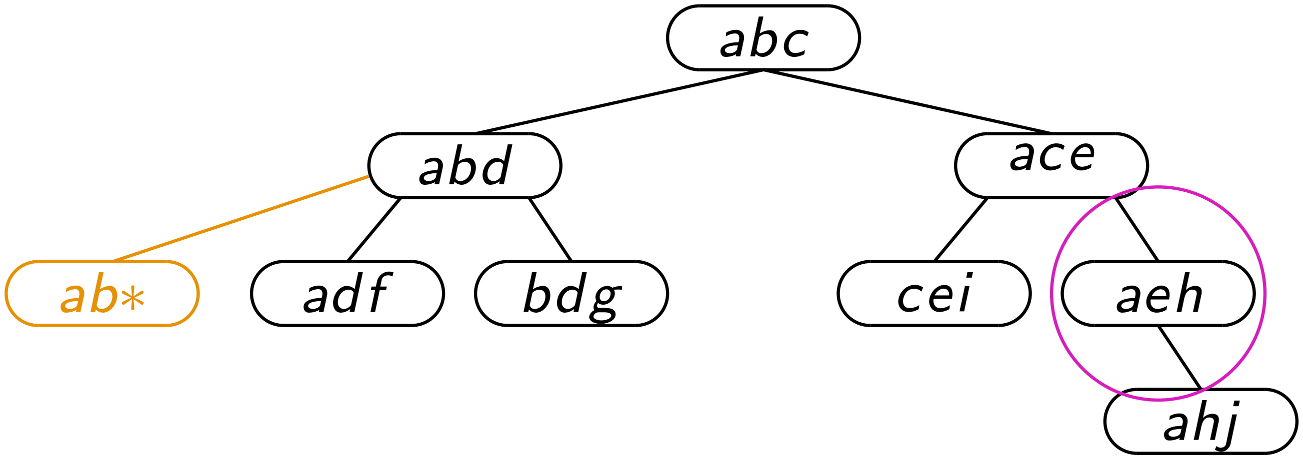


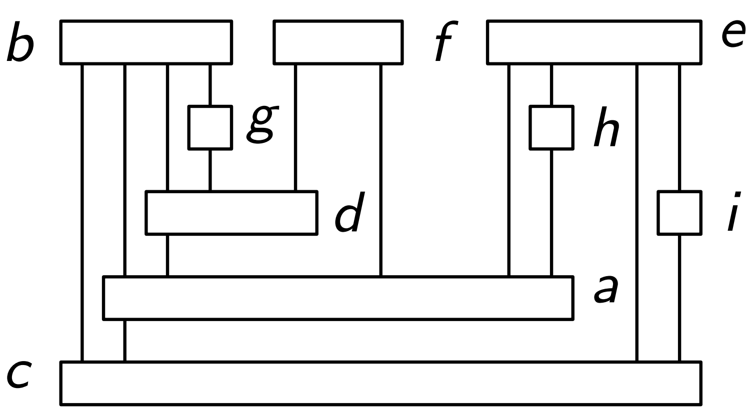
Example drawing



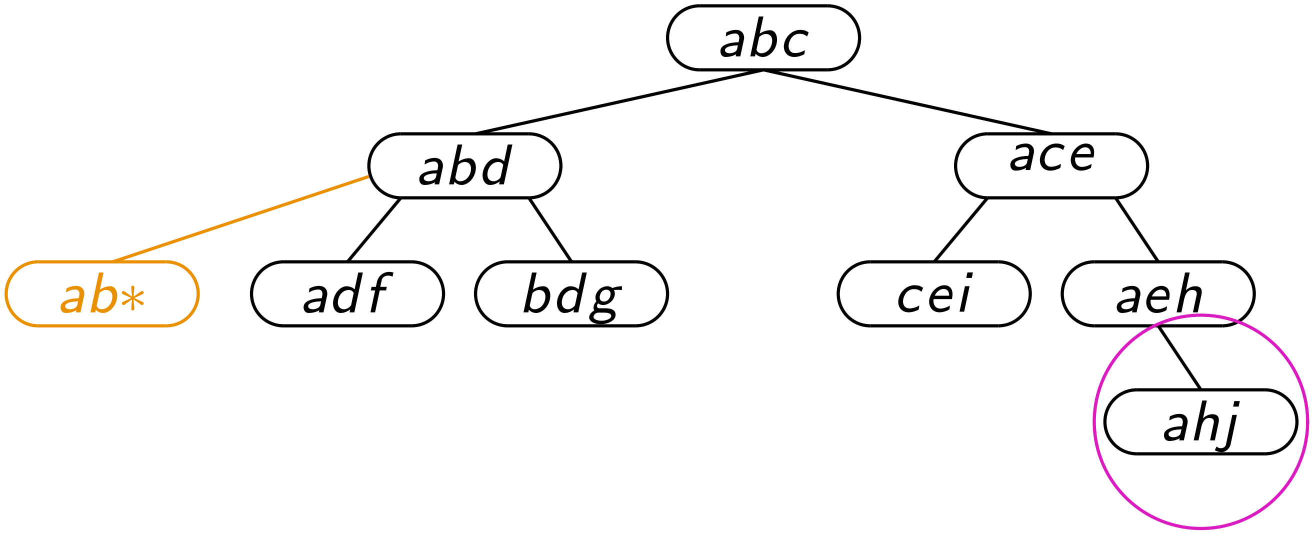


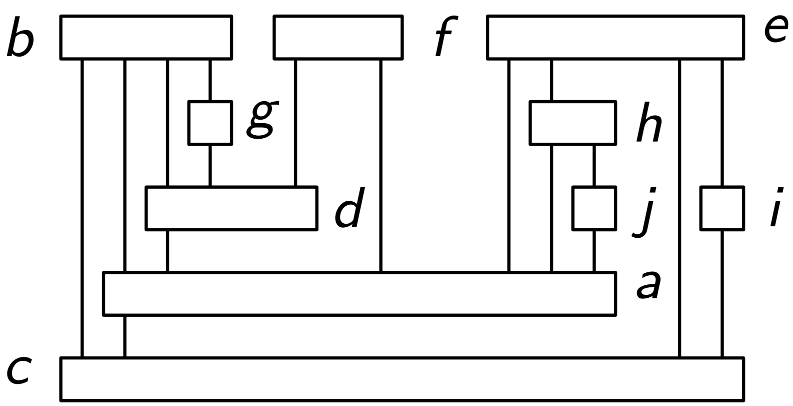
Example drawing



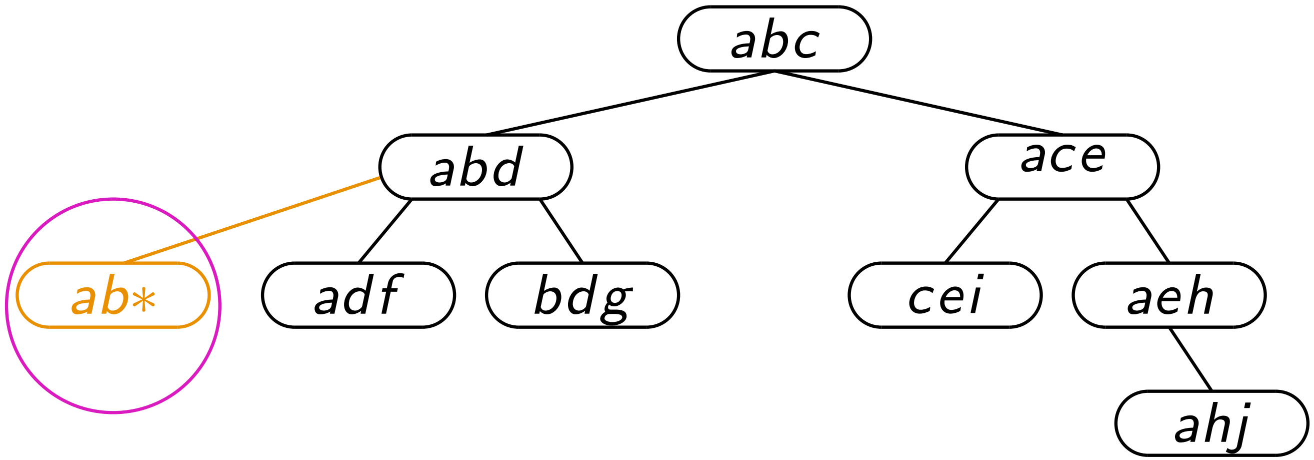


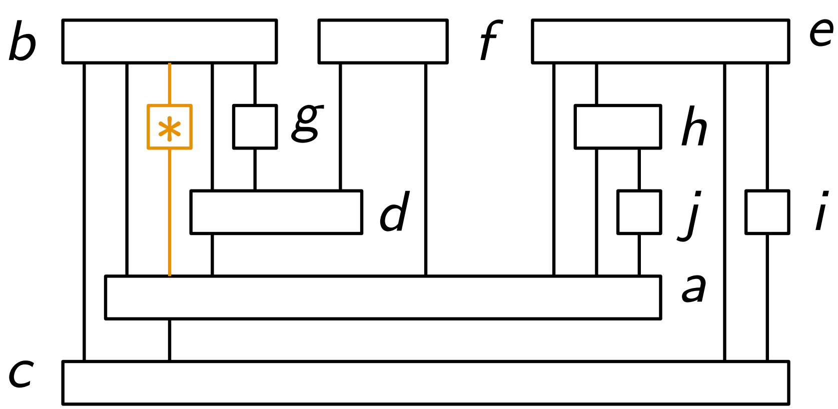
Example drawing



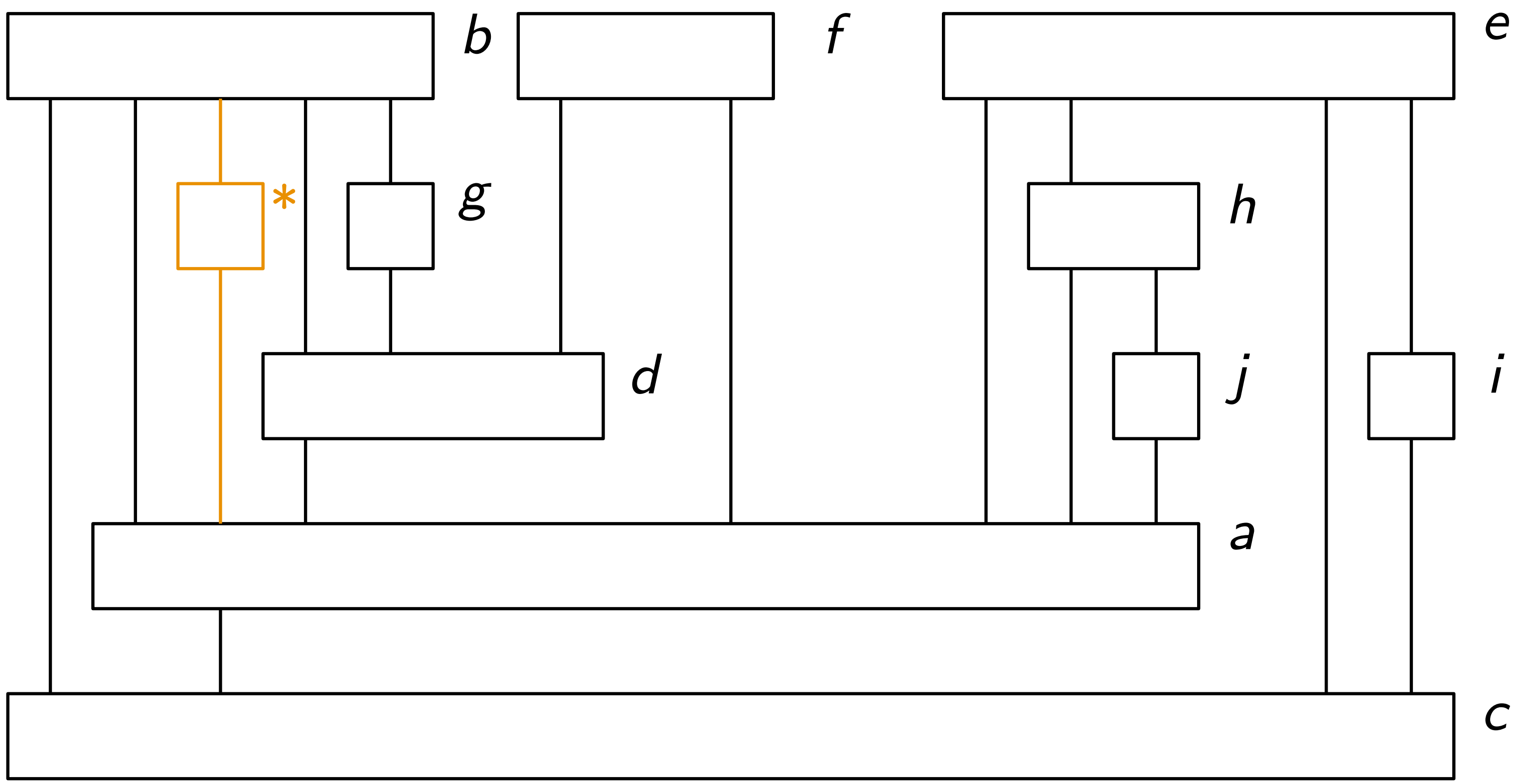


Example drawing

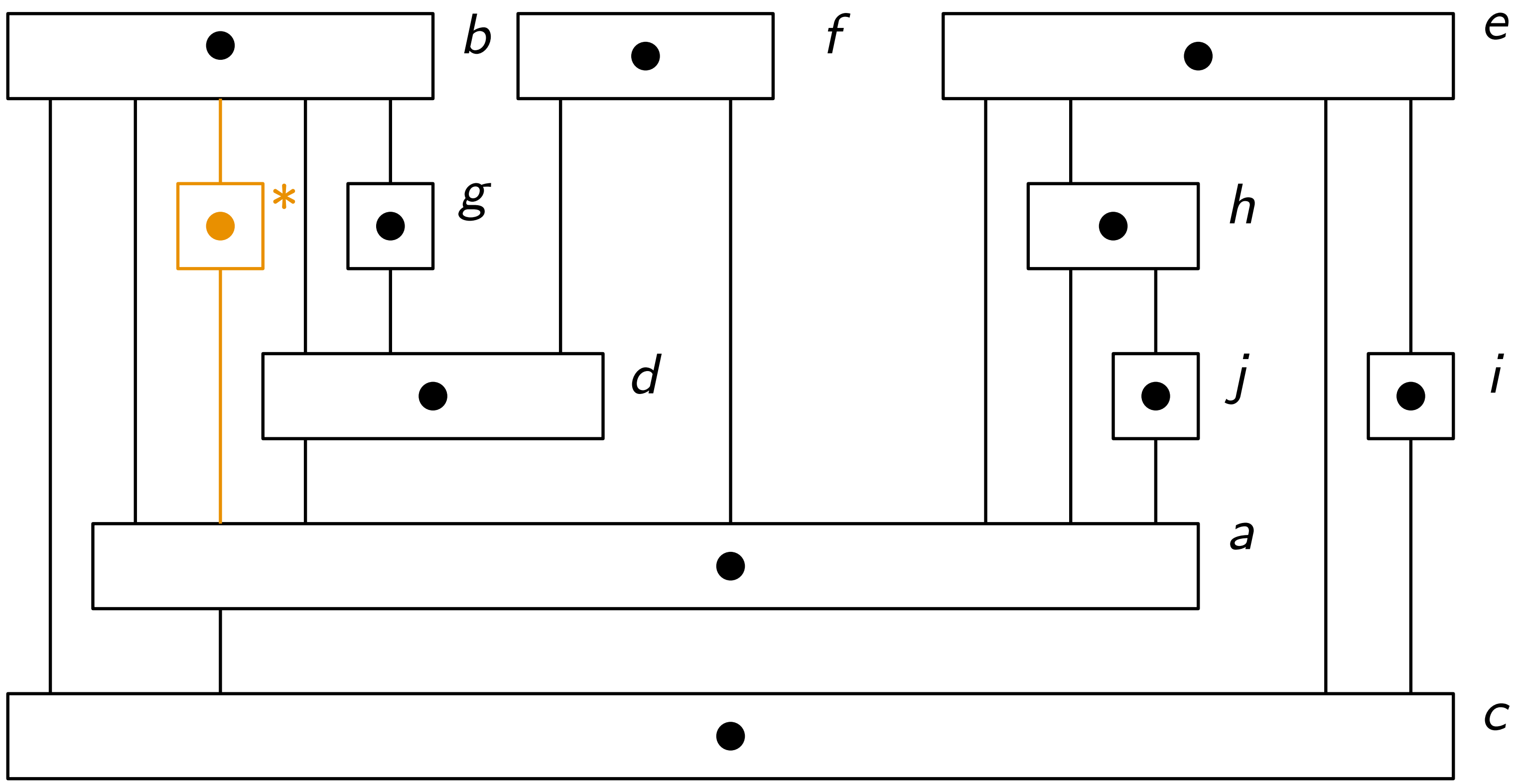




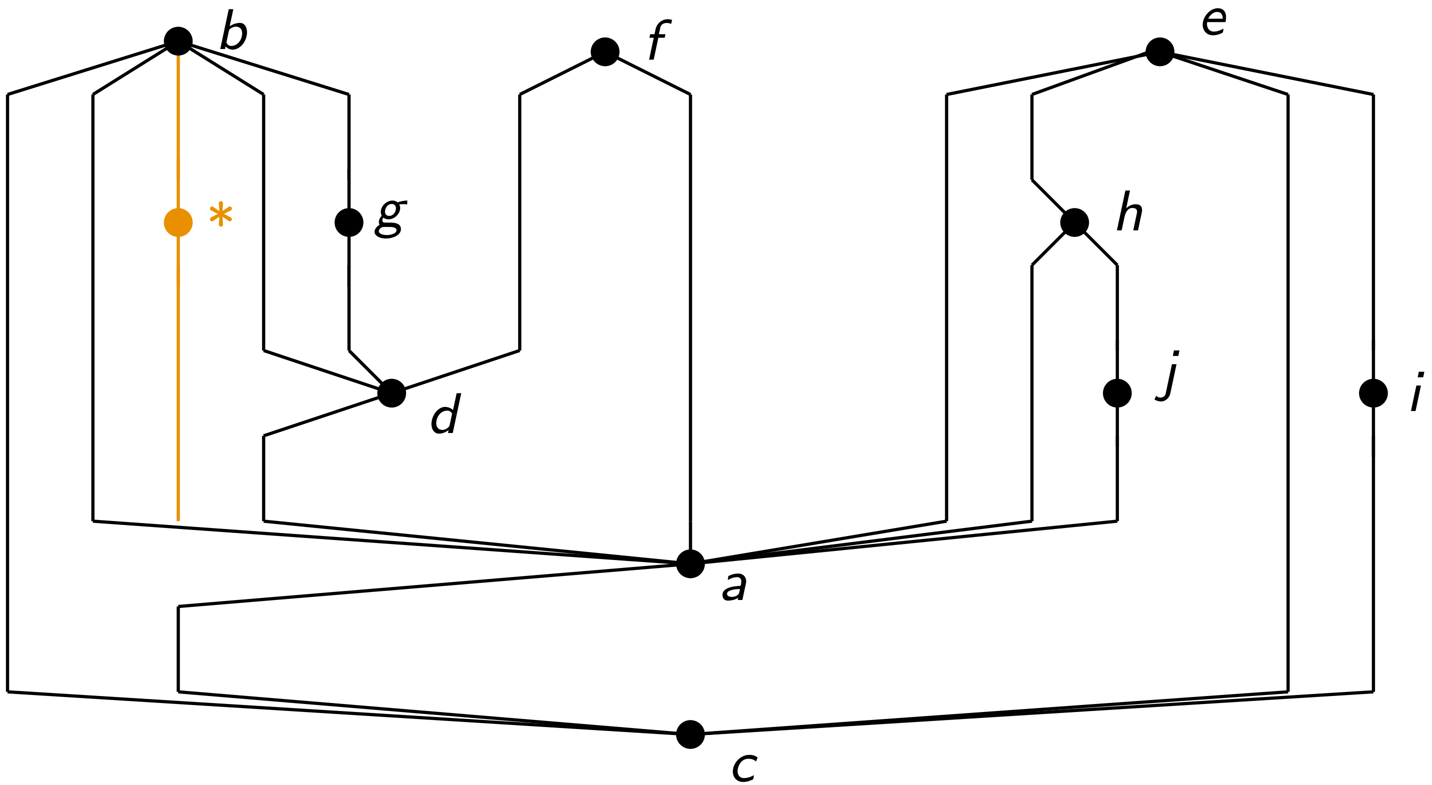
Example drawing



Example drawing



Example drawing



Overall Results

Graph Class	Ratio r	Area	Bends per Edge

Overall Results

Graph Class	Ratio r	Area	Bends per Edge
Complete k-ary Trees	$1 + \varepsilon$	$\mathcal{O}(n^2 \log n)$	-

Overall Results

Graph Class	Ratio r	Area	Bends per Edge
Complete k-ary Trees	$1 + \varepsilon$	$\mathcal{O}(n^2 \log n)$	-
General Trees	$1 + \varepsilon$	Exponential	-

Overall Results

Graph Class	Ratio r	Area	Bends per Edge
Complete k-ary Trees	$1+\varepsilon$	$\mathcal{O}(n^2 \log n)$	-
General Trees	$1+\varepsilon$	Exponential	-
Outerplanar Graphs	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n^2 \log^2 n)$	2

Overall Results

Graph Class	Ratio r	Area	Bends per Edge
Complete k-ary Trees	$1+\varepsilon$	$\mathcal{O}(n^2 \log n)$	-
General Trees	$1+\varepsilon$	Exponential	-
Outerplanar Graphs	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n^2 \log^2 n)$	2
Series-parallel Graphs	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(n^2 \log^2 n)$	2

Questions?

Thank you for participating!

:)

Planar 3-trees

Complete *k*-ary Tree Implementation

- Implemented in `python` $\geq$ `3.8`
- `networkx` library used for a graph structure representation
- `matplotlib` library used for plotting
- Coordinate computation can be found in thesis