Phylogenetics documentation

Details

Capabilities

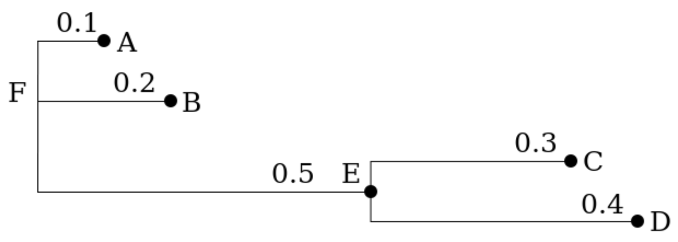
The Phylogenetics` package can:

- parse a Newick tree string to a *Mathematica*-usable Tree object and write a Tree object to a Newick string;
- parse a Cluster object of the HierarchicalClustering` package to a Tree and convert a Tree to Cluster representation;
- test properties of Tree objects;
- extract vertices, edges, internal nodes, leafs, sub- and supertrees of Tree objects;
- directly calculate various properties of Tree objects like paths and distances between vertices;
- convert a Tree object to Graph, and a tree Graph to Tree; any tree graph (TreeGraph, ClusteringTree, etc.) for which TreeGraphQ returns True can be converted to Tree;
- convert a Tree object or a Graph tree to Cladogram graph,
- provide example trees in Newick and Cluster format.

Tree specification

A phylogenetic tree can be represented in many ways. The Newick format is a minimal comma-bracket string descriptor of a directed tree (see the Newick Standard and this site for syntax details, here is an online tree visualizer). A simple Newick string example is resolved as follows:

```
"(A:0.1,B:0.2,(C:0.3,D:0.4)E:0.5)F;"
```



Where numbers indicate branch lengths between connected parent-child nodes. The unit $(b_1, b_2, \ldots)\,n$ represents a tree, rooted at node n with branches b_i in parenthesis. Branch b_i can be a tree itself or a leaf node. Any node can have a label *lbl* and a distance *d* to its parent in the form *lbl* : *d*. Multiple, independent trees can be separated by a semicolon (;). The Newick format is rather flexible, as is apparent from the wiki. Both the label and the distance can be omitted for nodes, hence a pure comma-bracket string like ((,), ,) is a valid tree with unnamed nodes and unit distances.

The above Newick string is converted to a Tree object that basically inherits the Newick structure, but is a full representation where nothing is omitted:

```
Tree[F, {A, B, Tree[E, {C, D}]}]
```

which is represented in StandardForm output as:

```
Tree[F, {A, B, Tree[E, {C, D}]}]
```

A Tree object is a description of a tree graph, a directed, acyclic graph, that always has a root vertex with name *n* and can contain branches b_i (polytomies, singular branches or no branches are allowed at any branching):

```
Tree[n, {b₁, b₂, …}]
```

The root of a tree is always a vertex, but a branch can be a vertex or a Tree. Vertices (internal nodes and leafs) are represented as associations, containing the following information:

```
<|"Name" → F, "Label" → F, "BranchLength" → 1, "Type" → "Node"|>
```

"Name" specifies the unique name of the vertex, just like in Graph.
"Label" specifies the label of the vertex that might appear when the tree is displayed as a graph.
"BranchLength" specifies the distance of the vertex to its parent vertex.
"Type" specifies the type of the vertex: Node indicates an internal node, Leaf indicates a terminal node.
Additional vertex-specific properties can be easily added, by editing the vertex-specification ($TreeVertexProperties).

Further notes

- A Newick tree is right associative: "(b)a" is the valid specification instead of "a(b)". The latter is not recognized correctly.
- Polytomies are allowed in a Newick string (and in a Tree) at any level in the form of "(a,b,c)d".
- If no distance is specified for a node to its parent, it is assumed to be 1.
- If the distance of a node is 0, when visualized, it can appear on top of its parent node, possibly covering it.
- If a node label is not specified, it is assumed to be None. When converting to Tree format (and to Graph or Cladogram), all nodes are resolved to have unique names, and labels are added as VertexLabels.
- TreeToGraph and Cladogram return Graph objects, with all the Graph functionalities.
- TreeGraph represent the distance of nodes along the branching dimension (i.e. the *y* coordinate) by vertex weights using VertexWeight; Cladogram does the same.
- Cladogram stores the stem lengths as EdgeWeight.
- Since the vertex positions represent important information of the graph (e.g. similarity or evolution-ary distance of vertices), they must be explicitly represented and not distorted by e.g. AspectRatio.
- Conversion from Graph to Tree and back does not necessarily result in the same vertex indices as a Tree is always parsed from root to leaves while a graph may not be parsed the same way by *Mathematica* when indexing vertices.

Sources

- The Newick Standard: http://evolution.genetics.washington.edu/phylip/newicktree.html
- Wikipedia: Newick format: https://en.wikipedia.org/wiki/Newick_format
- Newick tree formats: http://marvin.cs.uidaho.edu/Teaching/CS515/newickFormat.html
- An online Newick tree visualizer: http://etetoolkit.org/treeview/
- Orthogonal EdgeShapeFunction by Vitaliy Kaurov: http://community.wolfram.com/groups/-/m/t/241376

Package initialization

Load the Phylogenetics` package and check its version number.

```
In[1]:= Needs["Phylogenetics`"];
        $PhylogeneticsVersionNumber
Out[2]= 1.1
```

Available functions within the Phylogenetics` package.

```
In[4]:= Information["Phylogenetics`*"]
```

▼ Phylogenetics`

AncestorIndex	DescendantIndex	NewickToTree	Tree	TreeToNewick
BinaryTreeQ	DistanceIndex	NewickTreeQ	TreeDistance	TreeTop
BranchLength	DivergenceTime	Node	TreeDistanceMatrix	UniqueVerticesQ
BranchLengthIndex	FullVertexList	NodeList	TreePath	$PhylogeneticsVersionNumber
ChildIndex	GraphToTree	NodeQ	TreeQ	$TreeVertexProperties
Cladogram	Leaf	PhylogeneticData	TreeRoot	
ClusterToTree	LeafList	Subtree	TreeToCluster	
CommonAncestor	LeafQ	Supertree	TreeToGraph	

Trees

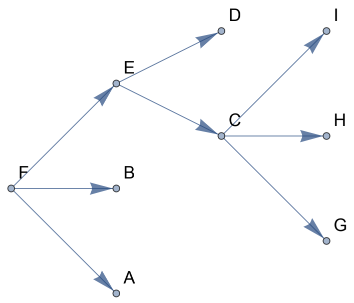
NewickToTree parses a Newick string and returns a Tree object.

```
In[5]:= tree = NewickToTree["(A:0.1,B:0.2,((G:.1,H:0.2,I:0.3)C:0.3,D:0.4)E:0.5)F;"]
Out[5]= Tree[F, {A, B, Tree[E, {Tree[C, {G, H, I}], D}]}]
```

Visualize the tree.

```
In[6]:= TreeToGraph[tree, ImageSize → Small]
```

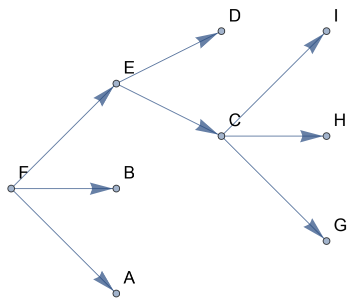
Out[6]=



Select and copy the full output Tree object and use it as input. Make sure, that you select the whole expression.

```
In[7]:= TreeToGraph[Tree["F",
          {"A", "B", Tree["E", {Tree["C", {"G", "H", "I"}], "D"}]}], ImageSize → Small]
```

Out[7]=



Query all vertices, leaf or internal node vertices of the tree.

```
In[8]:= {VertexList[tree], LeafList@tree, NodeList[tree]}

Out[8]= {{F, A, B, E, C, G, H, I, D}, {A, B, G, H, I, D}, {F, E, C}}
```

Query the branch length of all nodes or for a selected set of nodes.

```
In[9]:= {BranchLength[tree], BranchLength[tree, {"F", "B", "C"}]}

Out[9]= {{1, 0.1, 0.2, 0.5, 0.3, 0.1, 0.2, 0.3, 0.4}, {1, 0.2, 0.3}}
```

Return the vertices of subtrees rooted at specific nodes.

```
In[10]:= {VertexList[Subtree[tree, "C"]],
          VertexList[Subtree[tree, "A"]], VertexList[Subtree[tree, "F"]]}

Out[10]= {{C, G, H, I}, {A}, {F, A, B, E, C, G, H, I, D}}
```

Query the edge list of a tree.

```
In[11]:= EdgeList[tree]

Out[11]= {F → A, F → B, F → E, E → C, E → D, C → G, C → H, C → I}
```

Display tree as a table of nodes.

```
In[12]:= FullVertexList[tree] // Dataset
```

Out[12]=

Name	Label	BranchLength	Type
F	F	1	Node
A	A	0.1	Leaf
B	B	0.2	Leaf
E	E	0.5	Node
C	C	0.3	Node
G	G	0.1	Leaf
H	H	0.2	Leaf
I	I	0.3	Leaf
D	D	0.4	Leaf

Indices

DistanceIndex lists the absolute distance of each node from the absolute root at distance 0. Note, that the explicit root *D* has a distance of 4 from the unspecified implicit root.

```
In[13]:= DistanceIndex[NewickToTree["(((A:1)B:2)C:3)D:4;"]]

Out[13]= <|D → 4, C → 7, B → 9, A → 10|>
```

If no distance is specified for the explicit root vertex (*D*), it is assumed to be 1.

```
In[14]:= DistanceIndex[NewickToTree["(((A:1)B:2)C:3)D;"]]

Out[14]= <|D → 1, C → 4, B → 6, A → 7|>
```

Specify a node to be the absolute root by assigning a branch length of 0.

```
In[15]:= DistanceIndex[NewickToTree["(((A:1)B:2)C:3)D:0;"]]

Out[15]= <|D → 0, C → 3, B → 5, A → 6|>
```

DescendantIndex lists all the vertices below each vertex in topological (depth-first) order.

```
In[16]:= tree = NewickToTree["(A:0.1,B:0.2,((G:.1,H:0.2,I:0.3)C:0.3,D:0.4)E:0.5)F:0.3;"];
         DescendantIndex[tree]

Out[17]= <|F → {A, B, E, C, G, H, I, D}, E → {C, G, H, I, D},
          D → {}, C → {G, H, I}, I → {}, H → {}, G → {}, B → {}, A → {}|>
```

AncestorIndex returns all the direct ancestors of vertices in the tree in topological order, starting from the root.

```
In[18]:= AncestorIndex[tree]

Out[18]= <|F → {}, A → {F}, B → {F}, G → {F, E, C},
          C → {F, E}, E → {F}, H → {F, E, C}, I → {F, E, C}, D → {F, E}|>
```

ChildIndex lists all the vertices immediately below each vertex.

```
In[19]:= ChildIndex[tree]

Out[19]= ⟨|F → {A, B, E}, E → {C, D}, D → {}, C → {G, H, I}, I → {}, H → {}, G → {}, B → {}, A → {}|⟩
```

BranchLengthIndex lists the branch length values for each vertex (i.e. the distance of the vertex to its parent node).

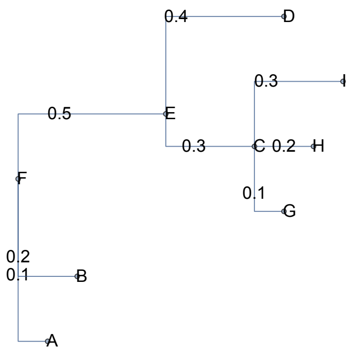
```
In[20]:= BranchLengthIndex[tree]

Out[20]= ⟨|F → 0.3, A → 0.1, B → 0.2, E → 0.5, C → 0.3, G → 0.1, H → 0.2, I → 0.3, D → 0.4|⟩
```

TreeDistanceMatrix

Display parent-child node distances stored in a tree as edgelabels in a cladogram.

```
In[21]:= tree = NewickToTree["(A:0.1,B:0.2,((G:.1,H:0.2,I:0.3)C:0.3,D:0.4)E:0.5)F;"];
         Cladogram[tree, ImageSize → Small, EdgeLabels → "EdgeWeight"]
```

Out[22]=



Return all the pairwise distances of the tree as a matrix.

```
In[23]:= TreeDistanceMatrix[tree] // MatrixForm

Out[23]//MatrixForm=
 0    0.1  0.2  0.5  0.8  0.9  1.   1.1  0.9
 0.1  0    0.3  0.6  0.9  1.   1.1  1.2  1.
 0.2  0.3  0    0.7  1.   1.1  1.2  1.3  1.1
 0.5  0.6  0.7  0    0.3  0.4  0.5  0.6  0.4
 0.8  0.9  1.   0.3  0    0.1  0.2  0.3  0.7
 0.9  1.   1.1  0.4  0.1  0    0.3  0.4  0.8
 1.   1.1  1.2  0.5  0.2  0.3  0    0.5  0.9
 1.1  1.2  1.3  0.6  0.3  0.4  0.5  0    1.
 0.9  1.   1.1  0.4  0.7  0.8  0.9  1.   0
```

Query the distance matrix of a specific subset of vertices.

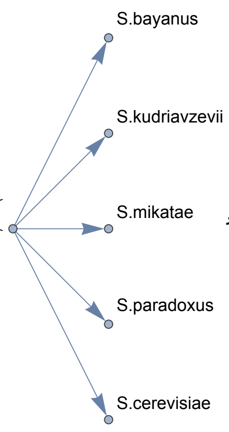
```
In[24]:= v = {"F", "F", "A", "H"};
         MatrixForm[TreeDistanceMatrix[tree, v], TableHeadings → {v, v}]

Out[25]//MatrixForm=
   |  F    F    A    H
 F |  0    0    0.1  1.
 F |  0    0    0.1  1.
 A |  0.1  0.1  0    1.1
 H |  1.   1.   1.1  0
```

Layout

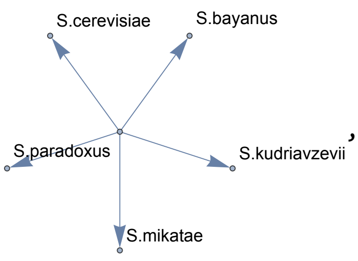
Display trees as graphs subject to different layouts.

```
In[26]:= tree = NewickToTree[PhylogeneticData["NewickSaccharomyces"]];
         {
          TreeToGraph[tree],
          TreeToGraph[tree, GraphLayout → "RadialDrawing"],
          TreeToGraph[tree, EdgeLabels → Automatic,
           GraphLayout → {"StarEmbedding", "Center" → Automatic}, EdgeLabels → "EdgeWeight"],
          Cladogram[tree, EdgeLabels → Automatic, ImagePadding → {{5, 50}, {5, 10}}]
         }
```

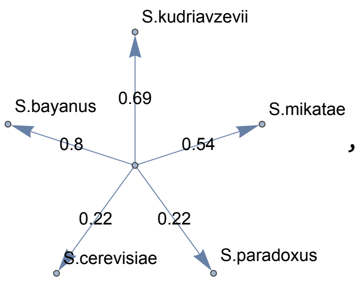
Out[27]= {



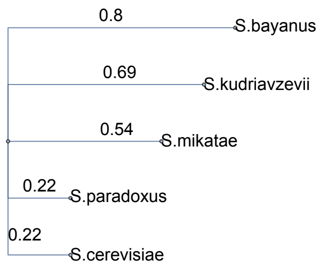
,



,



,



}

Cladogram is a graph object.

```
In[28]:= Head[Cladogram[tree]]

Out[28]= Graph
```

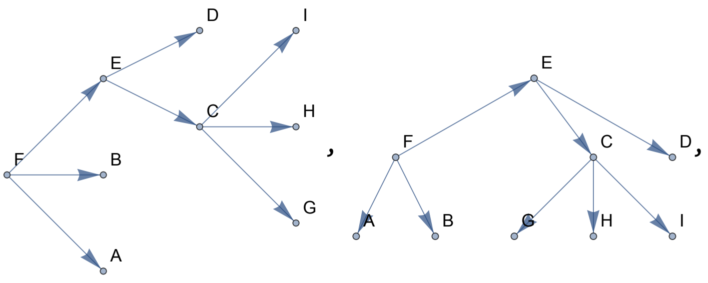
A Cladogram is a directed graph, with directed edges.

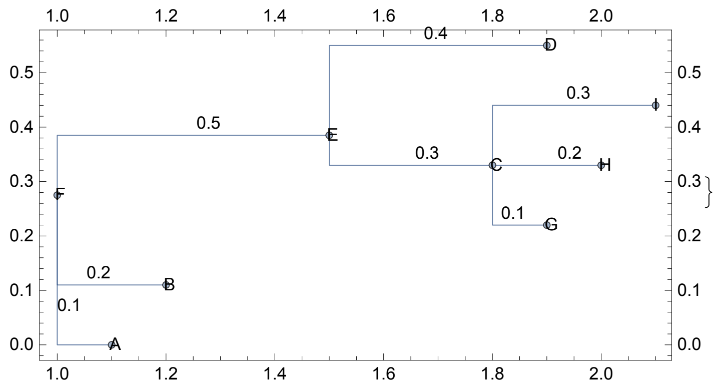
```
In[29]:= DirectedGraphQ[Cladogram[tree]]
         Cladogram[tree] // EdgeList

Out[29]= True

Out[30]= {1 ↦ S.cerevisiae, 1 ↦ S.paradoxus, 1 ↦ S.mikatae, 1 ↦ S.kudriavzevii, 1 ↦ S.bayanus}
```

Display the tree in different layouts.

```
In[31]:= tree = NewickToTree[PhylogeneticData["ExampleNewick"]];
         {
          TreeToGraph[tree, ImagePadding → 10],
          TreeGraph[VertexList[tree], EdgeList[tree], VertexLabels → "Name"],
          Cladogram[tree, EdgeLabels → Automatic,
           Frame → True, LayerSizeFunction → (1 # &), ImageSize → Medium]
         }
```

Out[32]= {





}

Conversion to and from Newick format

Converting from Newick string to tree and back should yield the same string.

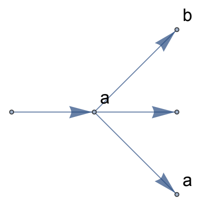
```
In[33]:= old = "(((a,(,,),a3)b))d:1.4;";
         new = TreeToNewick[NewickToTree[old]]
         old === new

Out[34]= (((a,(,,),a3)b))d:1.4;

Out[35]= True
```

Unnamed or identical nodes are assigned unique indices to avoid collision.

```
In[36]:= tree = NewickToTree["((a,,b)a)"]

Out[36]= Tree[1, {Tree[a, {2, 3, b}]}]
```

Labels are nevertheless retained and can be accessed by VertexLabels.

```
In[37]:= g = TreeToGraph[tree, ImageSize → Tiny]
```

Out[37]=


```
In[38]:= PropertyValue[g, VertexLabels]

Out[38]= {3 → None, b → b, a → a, 2 → a, 1 → None}
```

Conversion to and from hierarchical cluster format

Convert a hierarchical clustering structure to a tree.

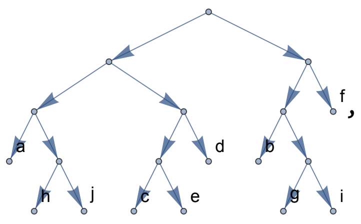
```
In[39]:= cluster = PhylogeneticData["ExampleCluster1"]
         tree = ClusterToTree[cluster]

Out[39]= Cluster[Cluster[Cluster[a, Cluster[h, j, 1.52217, 1, 1], 28.8538, 1, 2],
           Cluster[Cluster[c, e, 10.1371, 1, 1], d, 22.0063, 2, 1], 47.1129, 3, 3],
          Cluster[Cluster[b, Cluster[g, i, 2.5374, 1, 1], 5.73533, 1, 2], f, 13.6197, 3, 1],
          64.5168, 6, 4]

Out[40]= Tree[1,
          {Tree[2, {Tree[4, {a, Tree[7, {h, j}]}], Tree[5, {Tree[8, {c, e}], d}]}],
           Tree[3, {Tree[6, {b, Tree[9, {g, i}]}], f}]}]
```

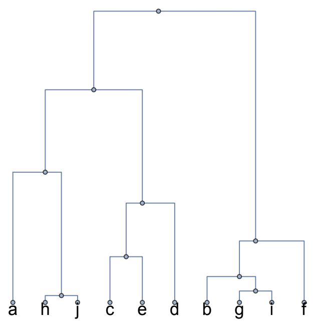
A hierarchical clustering structure is always binary, therefore if it is converted to Tree, the result will always be strictly binary.

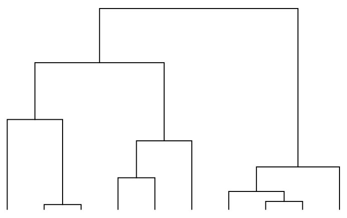
```
In[41]:= BinaryTreeQ[tree]

Out[41]= True
```

Visualize the tree.

```
In[42]:= {TreeToGraph[tree, GraphLayout → {"LayeredEmbedding", Orientation → Top}],
          Cladogram[tree, Orientation → Top, ImageSize → Small, ImagePadding → 10],
          DendrogramPlot[cluster, Orientation → Top]}
```

Out[42]= {


,


,

}

Convert back to cluster format.

```
In[43]:= new = TreeToCluster[tree]

Out[43]= Cluster[Cluster[Cluster[a, Cluster[h, j, 1.52217, 1, 1], 28.8538, 1, 2],
           Cluster[Cluster[c, e, 10.1371, 1, 1], d, 22.0063, 2, 1], 47.1129, 3, 3],
          Cluster[Cluster[b, Cluster[g, i, 2.5374, 1, 1], 5.73533, 1, 2], f, 13.6197, 3, 1],
          64.5168, 6, 4]
```

Conversion from and to cluster format should result in the same tree topology and values ***up to numerical precision***. Therefore, instead of SameQ, Equal should be used when comparing objects.

```
In[44]:= {cluster === new, cluster == new}

Out[44]= {False, True}
```

Subtree

Subtree returns the tree that is rooted at the requested node.

```
In[45]:= tree = NewickToTree["(A:0.1,B:0.2,((G:.1,H:0.2,I:0.3)C:0.3,D:0.4)E:0.5)F;"]
         Subtree[tree, "C"]

Out[45]= Tree[F, {A, B, Tree[E, {Tree[C, {G, H, I}], D}]}]

Out[46]= Tree[C, {G, H, I}]
```

If the vertex is a leaf, it is returned wrapped in Tree (with zero branches), so that it can be converted to Graph more easily by other functions.

```
In[47]:= Subtree[tree, "H"]

Out[47]= Tree[H, {}]
```

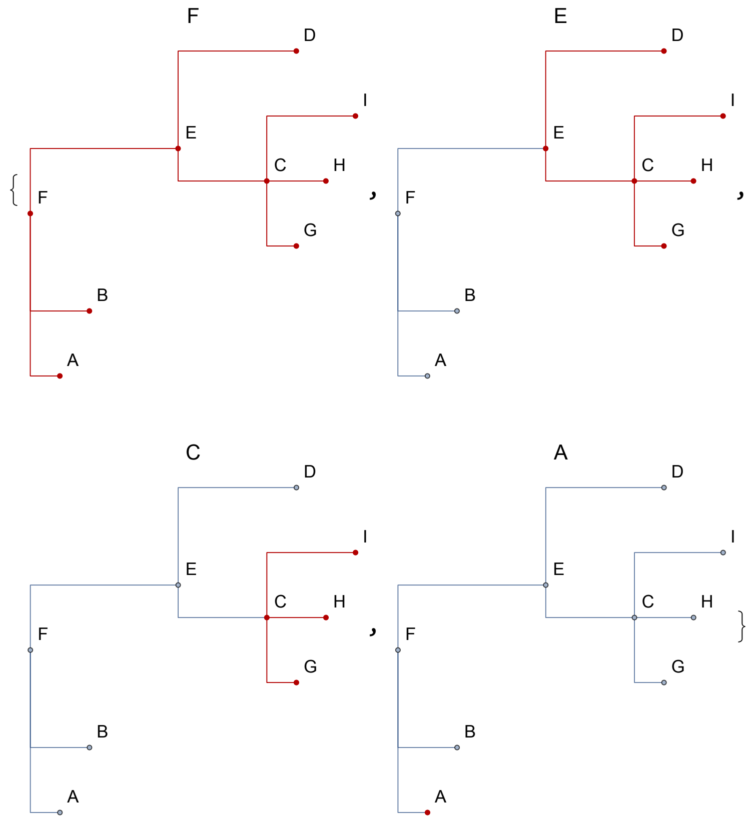
If the vertex is not present in the tree, Missing is returned.

```
In[48]:= Subtree[tree, "X"]

Out[48]= Missing[NotFound]
```

GraphHighlight, besides graphs, edges and vertices, works also with Tree objects in Cladogram.

```
In[49]:= Cladogram[tree, GraphHighlight → Subtree[tree, #],
            PlotLabel → #, VertexLabels → "Name"] & /@ {"F", "E", "C", "A"}
```

Out[49]=



Supertree

Supertree returns the most compact subtree of a tree that includes all listed vertices.

```
In[50]:= tree = NewickToTree["(A:0.1,B:0.2,((G:.1,H:0.2,I:0.3)C:0.3,D:0.4)E:0.5)F;"]

Out[50]= Tree[F, {A, B, Tree[E, {Tree[C, {G, H, I}], D}]}]

In[51]:= Supertree[tree, {"H", "I"}]

Out[51]= Tree[C, {G, H, I}]
```

Supertree works with any vertex, leaf or internal node.

```
In[52]:= Supertree[tree, {"H", "A", "C"}]

Out[52]= Tree[F, {A, B, Tree[E, {Tree[C, {G, H, I}], D}]}]
```

The supertree of a leaf vertex consists only itself, wrapped in Tree.

```
In[53]:= Supertree[tree, "G"]

Out[53]= Tree[G, {}]
```

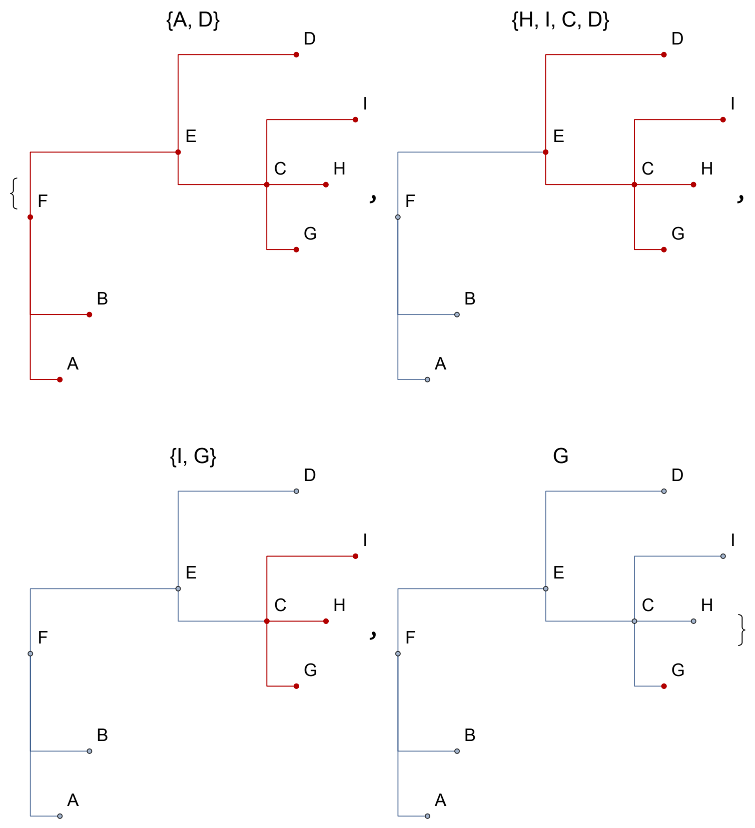
An unknown vertex does not have a supertree.

```
In[54]:= Supertree[tree, "X"]
Out[54]= Missing[NotFound]
```

Visualize supertrees as highlighted subgraphs.

```
In[55]:= Cladogram[tree, ImageSize → Small, GraphHighlight → Supertree[tree, #], PlotLabel → #,
            VertexLabels → "Name"] & /@ {{"A", "D"}, {"H", "I", "C", "D"}, {"I", "G"}, "G"}
```

Out[55]=



CommonAncestor

The common ancestor of two leaves is the internal node of the tree where they branch off.

```
In[56]:= tree = NewickToTree["(A:0.1,B:0.2,((G:.1,H:0.2,I:0.3)C:0.3,D:0.4)E:0.5)F;"]
         CommonAncestor[tree, {"H", "G"}]
Out[56]= Tree[F, {A, B, Tree[E, {Tree[C, {G, H, I}], D}]}]
Out[57]= C
```

The common ancestor of multiple nodes is the root of the smallest subtree inclusive of all listed vertices.

```
In[58]:= CommonAncestor[tree, {"H", "G", "D"}]
Out[58]= E
```

A common ancestor of a single node is itself.

```
In[59]:= CommonAncestor[tree, "E"]
Out[59]= E
```

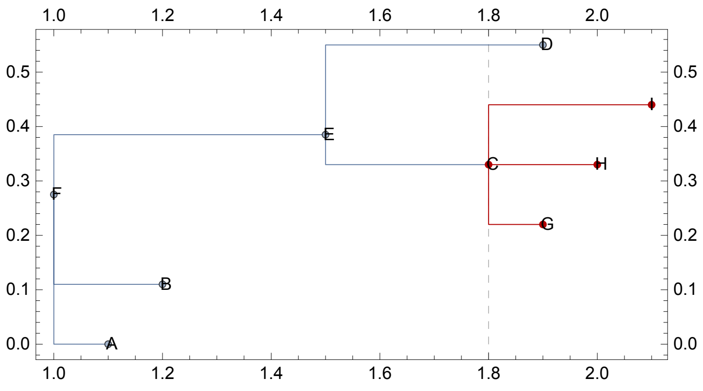
DivergenceTime

The divergence time of two vertices is the absolute distance of their common ancestor from the absolute, implicit root node (at distance 0).

```
In[60]:= tree = NewickToTree["(A:0.1,B:0.2,((G:.1,H:0.2,I:0.3)C:0.3,D:0.4)E:0.5)F;"]
         d = DivergenceTime[tree, {"H", "G"}]

Out[60]= Tree[F, {A, B, Tree[E, {Tree[C, {G, H, I}], D}]}]

Out[61]= 1.8
```

```
In[62]:= Cladogram[tree, Frame → True, LayerSizeFunction → (1 # &),
          GridLines → {{{d, Dashed}}, {}}, GraphHighlight → Supertree[tree, {"H", "G"}]]
```

Out[62]=



The divergence time of multiple vertices is the absolute time of the internal node that is the root of the smallest subtree containing all vertices.

```
In[63]:= DivergenceTime[tree, {"H", "G", "D"}]

Out[63]= 1.5
```

The divergence time of a single vertex is its absolute distance.

```
In[64]:= DivergenceTime[tree, "E"]

Out[64]= 1.5
```

Paths and distances

The path between two nodes is always the shortest path.

```
In[65]:= tree = NewickToTree["(A:0.1,B:0.2,((G:.1,H:0.2,I:0.3)C:0.3,D:0.4)E:0.5)F;"];
         TreePath[tree, "G", "D"]

Out[66]= {C → G, E → C, E → D}
```

The pair of vertices can be also provided as a list.

```
In[67]:= TreePath[tree, {"G", "D"}]

Out[67]= {C → G, E → C, E → D}
```

A single vertex returns a path from the root.

```
In[68]:= TreePath[tree, "G"]
Out[68]= {C → G, E → C, F → E}
```

A path between a vertex and itself returns the vertex.

```
In[69]:= TreePath[tree, "G", "G"]
Out[69]= G
```

The path to an unknown vertex is an empty list.

```
In[70]:= TreePath[tree, "G", "X"]
Out[70]= { }
```

The distance between two nodes is the summed branch lengths of the shortest path.

```
In[71]:= TreeDistance[tree, "G", "D"]
Out[71]= 0.8
```

The pair of vertices can be also provided as a list.

```
In[72]:= TreeDistance[tree, {"G", "D"}]
Out[72]= 0.8
```

A single vertex returns its absolute distance from the root.

```
In[73]:= TreeDistance[tree, "G"]
Out[73]= 0.9
```

A distance between a vertex and itself is always 0.

```
In[74]:= TreeDistance[tree, "G", "G"]
Out[74]= 0
```

Distance to an unknown vertex is infinite.

```
In[75]:= TreeDistance[tree, "G", "X"]
Out[75]= ∞
```

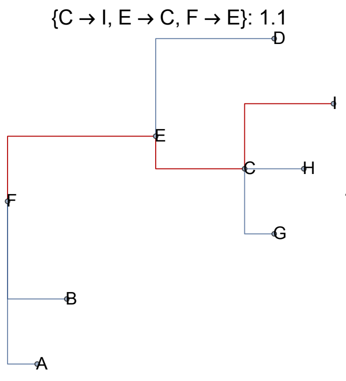
The distance of two nodes is always a nonnegative value, regardless of the direction.

```
In[76]:= {TreeDistance[tree, {"F", "I"}], TreeDistance[tree, {"I", "F"}]}
Out[76]= {1.1, 1.1}
```

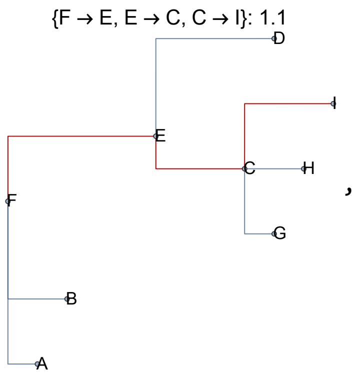
Examples.

```
In[77]:= list =
           {{"I", "F"}, {"F", "I"}, {"H", "H"}, {"H", "G"}, {"G", "D"}, {"H", "B"}, {"C"}, {"X"}};
         path = TreePath[tree, #] & /@ list;
         dist = TreeDistance[tree, #] & /@ list;
         MapThread[Cladogram[tree, GraphHighlight → #1,
             ImageSize → Small, PlotLabel → Row@{#1, ": ", #2}] &, {path, dist}]
```

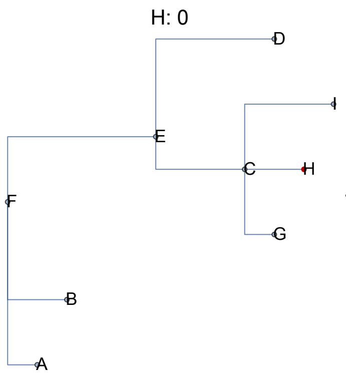
Out[80]= {



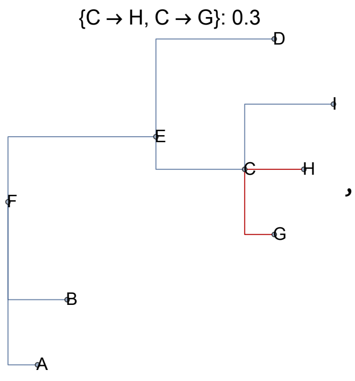
,



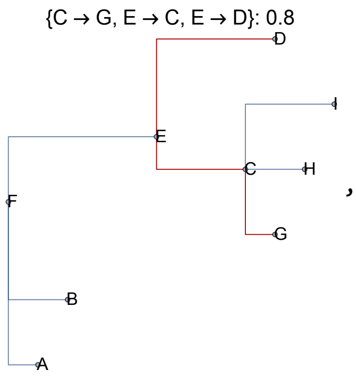
,



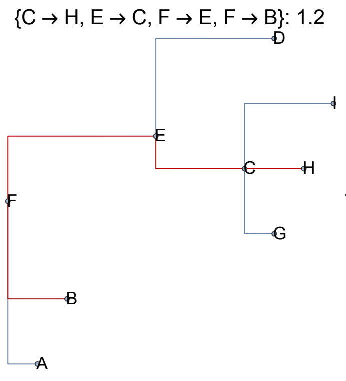
,



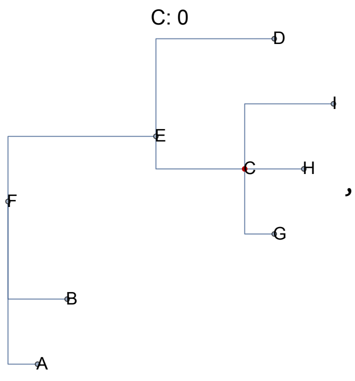
,



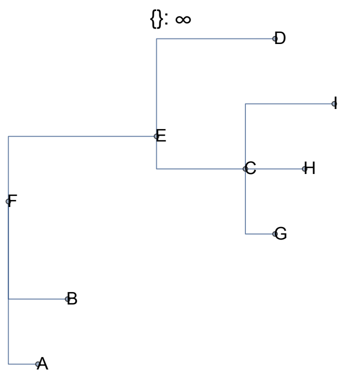
,



,



,



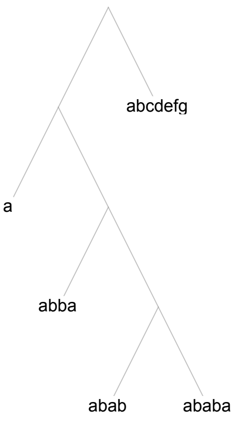
}

Clusters

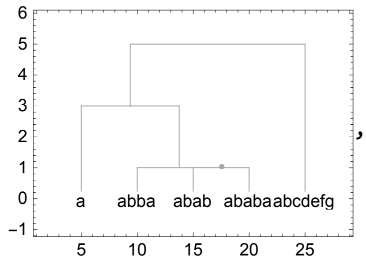
Clusterize a list of strings based on their similarity. Cladogram cannot work with such lists directly, but can accept the graph produced by ClusteringTree. Since Dendrogram returns a static Graphics object instead of a rich Graph, Cladogram cannot deal with it.

```
In[81]:= list = {"a", "abba", "abab", "ababa", "abcdefg"};
         {
          g = ClusteringTree[list, VertexLabelStyle → Black],
          Dendrogram[list, Frame → True, PlotRangePadding → Scaled[.15]],
          c = Cladogram[g, Orientation → Top, Frame → True,
            ImagePadding → All, PlotRangePadding → Scaled[.1], VertexLabelStyle → Black]
         }
```

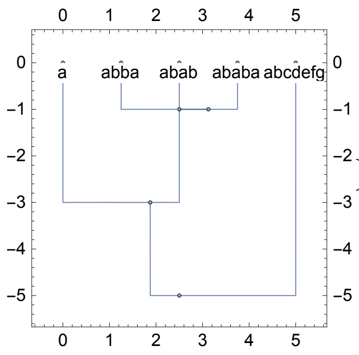
Out[82]= {



,



,



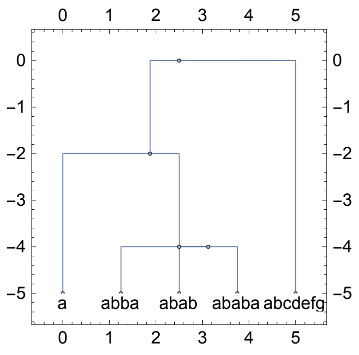
}

Since Cladogram measures distances from the root instead of (dis)similarity between vertices, the result is a mirrored plot. To display it correctly, invert the vertex weights that represent the vertical absolute distance for each node.

```
In[83]:= w = PropertyValue[c, VertexWeight]
         Cladogram[g, VertexWeight → (Max@w - w), Orientation → Top,
          Frame → True, ImagePadding → All, ImageSize → Small,
          PlotRangePadding → Scaled[.1], VertexLabelStyle → Black]
```

```
Out[83]= {5., 3., 0, 1., 0, 1., 0, 0, 0}
```

Out[84]=

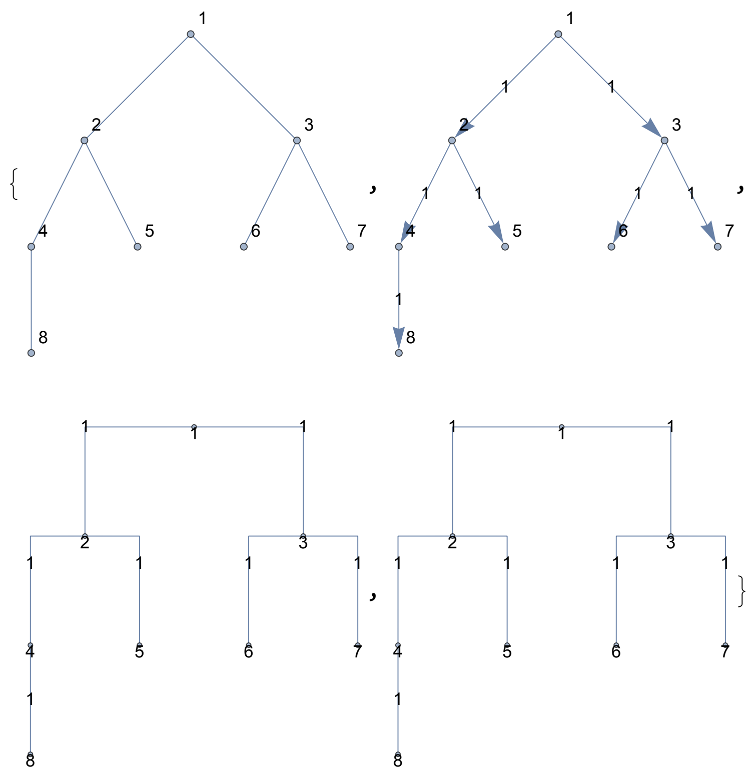


Graphs

Display a tree as a graph in various ways and layouts.

In[85]:=
```
g = KaryTree[8, DirectedEdges → False, VertexLabels → "Index"];
tree = GraphToTree[g]
{
 g,
 TreeToGraph[tree,
  GraphLayout → {"LayeredEmbedding", "Orientation" → Top, "RootVertex" → 1},
  EdgeLabels → "EdgeWeight"],
 Cladogram[tree, Orientation → Top, EdgeLabels → "EdgeWeight"],
 Cladogram[g, Orientation → Top, EdgeLabels → "EdgeWeight"]
}
```

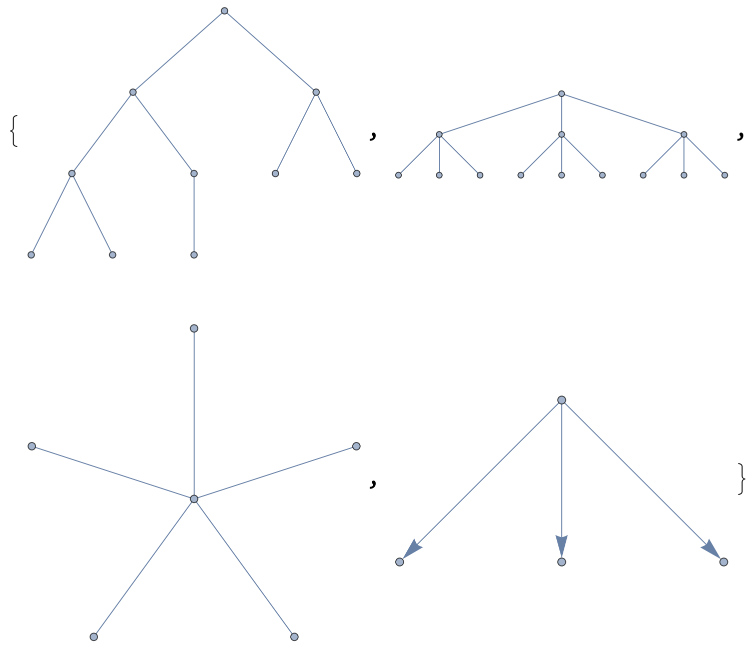
Out[86]=
```
Tree[1, {Tree[2, {Tree[4, {8}], 5}], Tree[3, {6, 7}]}]
```

Out[87]=


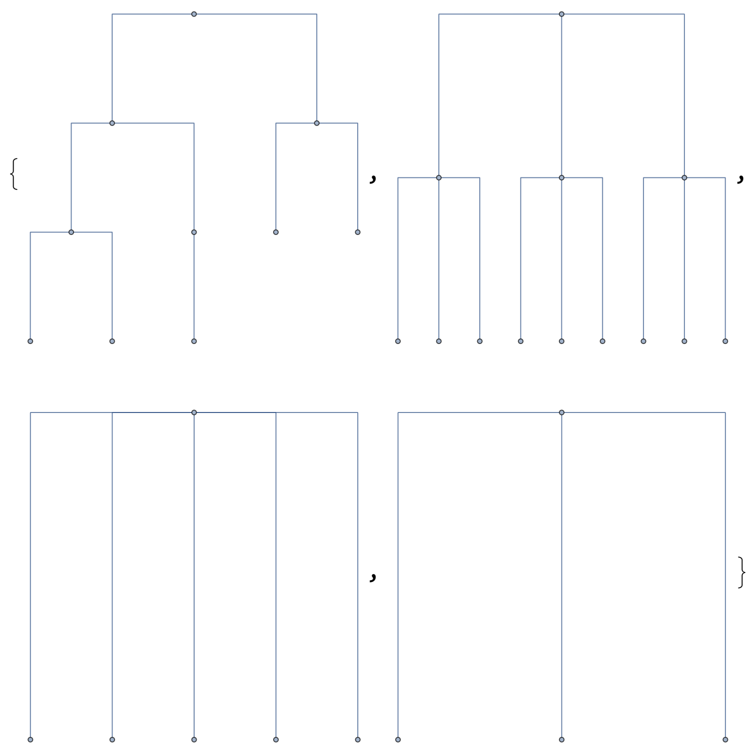
Display some example graphs as cladograms.

In[88]:=
```
graphs = {KaryTree[10], CompleteKaryTree[3, 3],
   StarGraph[6], TreeGraph[{1 → 2, 1 → 3, 1 → 4}]}
Cladogram[#, Orientation → Top] & /@ graphs
```

Out[88]=



Out[89]=



Phylogenetic data

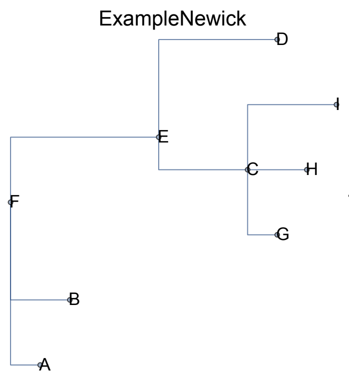
List the available phylogenetic examples stored within PhylogeneticData.

```
In[90]:= all = PhylogeneticData["Names"]

Out[90]= {ExampleNewick, ExampleNewick1, ExampleNewick2, ExampleNewick3, ExampleNewick6,
          ExampleNewick7, ExampleNewick8, ExampleNewick9, ExampleNewickEmpty1,
          ExampleNewickEmpty2, ExampleNewickEmpty3, ExampleNewickEmpty4,
          ExampleCluster1, ExampleCluster2, NewickMammal1, NewickMammal2,
          NewickMammal3, NewickHominoid, NewickSaccharomyces, NewickEukarya}
```

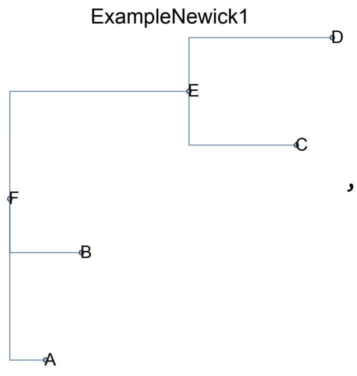
Display example data.

```
In[91]:= Cladogram[If[StringMatchQ[#, "*Newick*"], NewickToTree, ClusterToTree]@
             PhylogeneticData[#], ImageSize → Small, PlotLabel → #] & /@ Most[all]
```

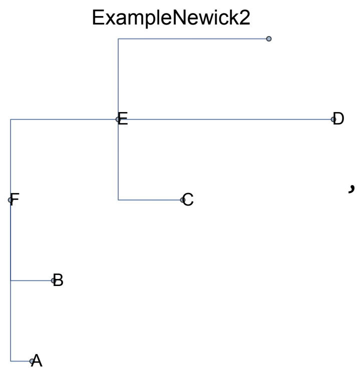
Out[91]= {



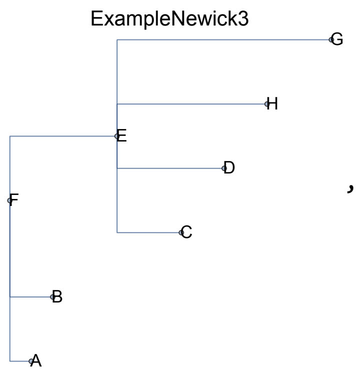
,



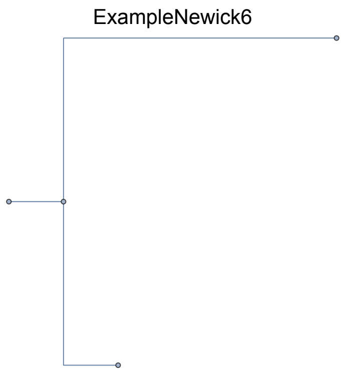
,



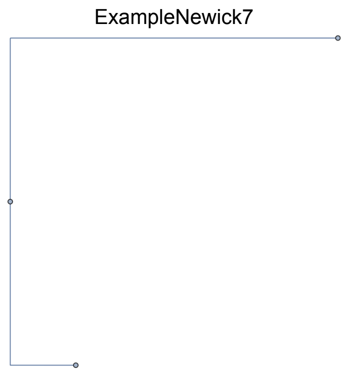
,



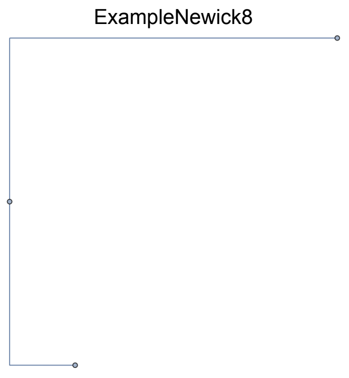
,



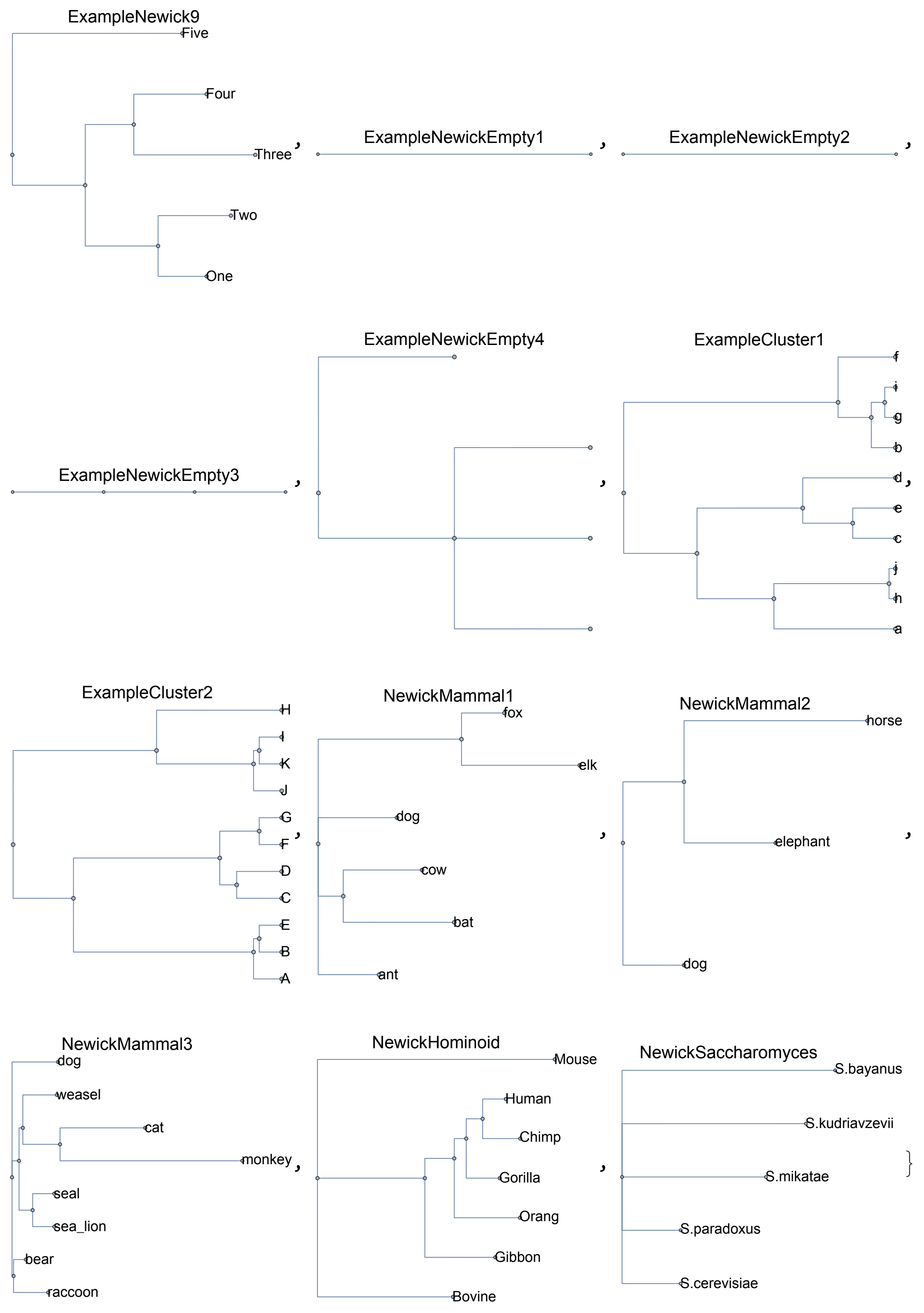
,



,



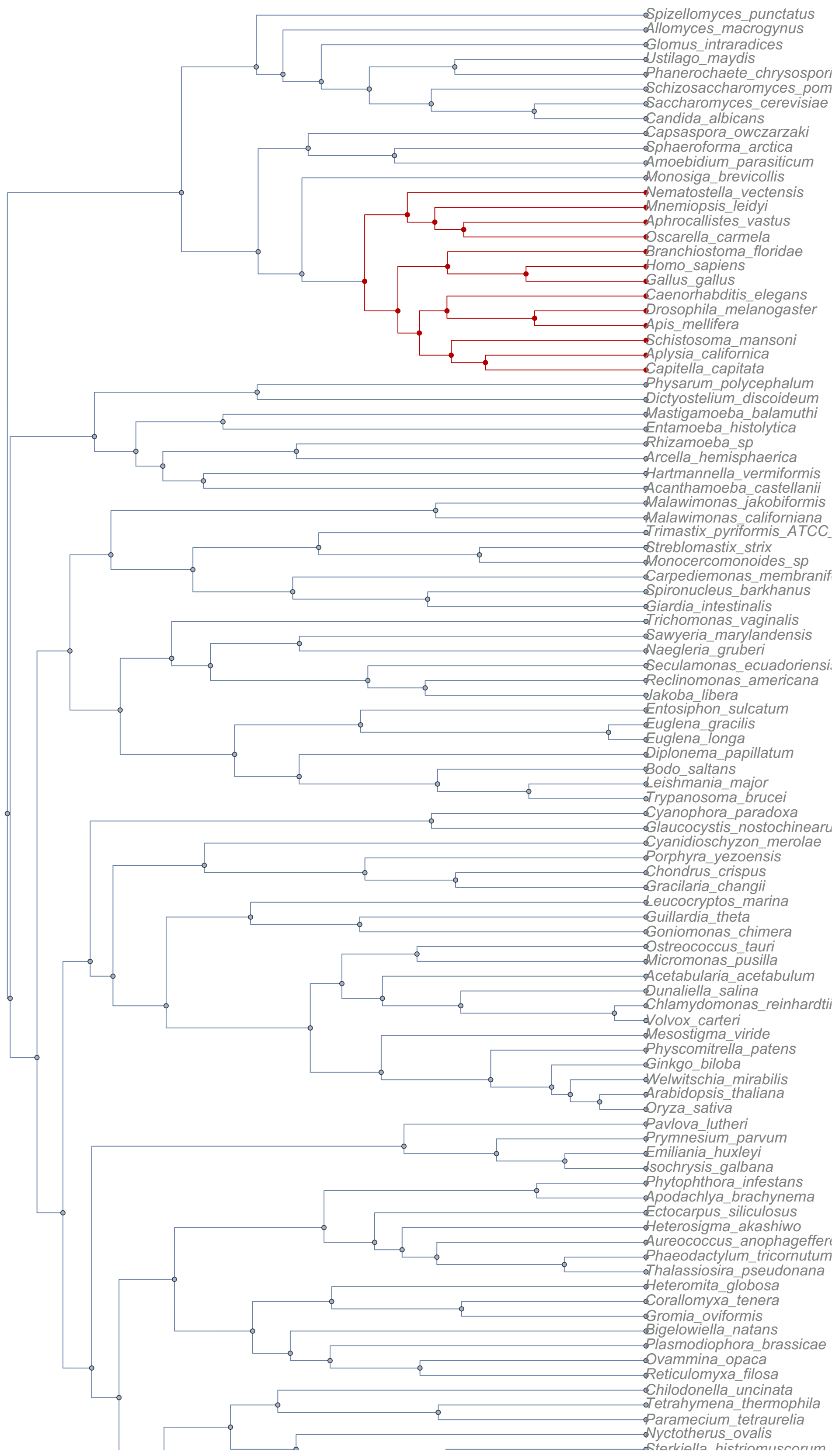
,

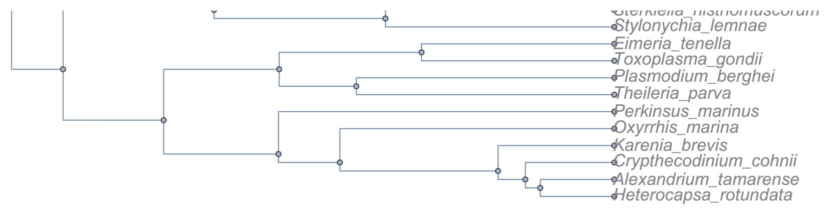


A complex tree, showing the diversificaion of Eukarya, with Metazoa highlighted (from *Parfrey et al. 2011*).

In[92]:=
```
tree = NewickToTree[PhylogeneticData["NewickEukarya"]];
Cladogram[tree, ImageSize → 500,
 ImagePadding → {{1, 100}, {1, 1}}, LayerSizeFunction → (1/5 # &),
 VertexSize → .3, VertexLabelStyle → Directive[Gray, Italic, 9],
 GraphHighlight → Supertree[tree, {"Nematostella_vectensis", "Homo_sapiens"}]]
```

Out[93]=




Sterkiella_histriomuscorum
Stylonychia_lemnae
Eimeria_tenella
Toxoplasma_gondii
Plasmodium_berghei
Theileria_parva
Perkinsus_marinus
Oxyrrhis_marina
Karenia_brevis
Crypthecodinium_cohnii
Alexandrium_tamarense
Heterocapsa_rotundata