Phylogenetics documentation

Details

Capabilities

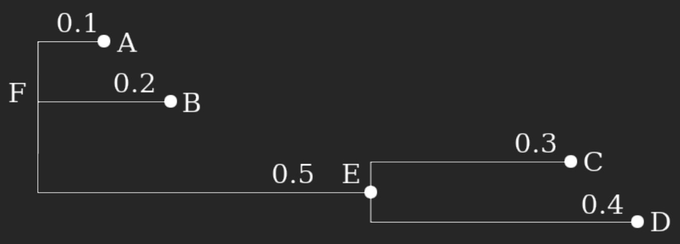
The `Phylogenetics`` package can:

- 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 evolutionary 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
```

Set some options for plotting trees.

```
In[3]:= Quiet@SetOptions[#, VertexLabelStyle → White, EdgeLabelStyle → Orange,
              EdgeStyle → Thick, FrameStyle → Gray, LabelStyle → Gray] & /@
           {Dendrogram, Cladogram, Graph, TreeGraph, TreeToGraph, KaryTree};
        SetOptions[DendrogramPlot, LeafLabels → (Style[#, White] &), PlotStyle → White];
```

Available functions within the `Phylogenetics`` package.

```
In[5]:= 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	

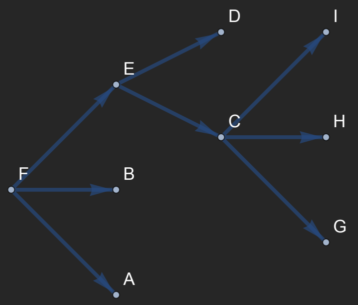
Trees

`NewickToTree` parses a Newick string and returns a `Tree` object.

```
In[6]:= 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[6]= Tree[F, {A, B, Tree[E, {Tree[C, {G, H, I}], D}]}]
```

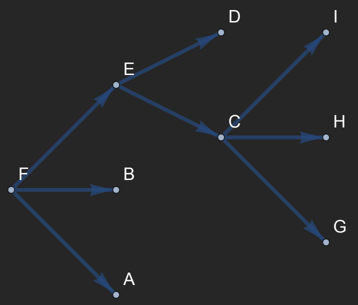
Visualize the tree.

```
In[7]:= TreeToGraph[tree, ImageSize → Small]
Out[7]=
```



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

```
In[8]:= TreeToGraph[Tree["F",
          {"A", "B", Tree["E", {Tree["C", {"G", "H", "I"}], "D"}]}], ImageSize → Small]
Out[8]=
```



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

```
In[9]:= {VertexList[tree], LeafList@tree, NodeList[tree]}
Out[9]= {{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[10]:= {BranchLength[tree], BranchLength[tree, {"F", "B", "C"}]}
Out[10]= {{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[11]:= {VertexList[Subtree[tree, "C"]],
          VertexList[Subtree[tree, "A"]], VertexList[Subtree[tree, "F"]]}
Out[11]= {{C, G, H, I}, {A}, {F, A, B, E, C, G, H, I, D}}
```

Query the edge list of a tree.

```
In[12]:= EdgeList[tree]
Out[12]= {F → A, F → B, F → E, E → C, E → D, C → G, C → H, C → I}
```

Display tree as a table of nodes.

```
In[13]:= FullVertexList[tree] // Dataset
```

Out[13]=

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[14]:= DistanceIndex[NewickToTree["(((A:1)B:2)C:3)D:4;"]]
Out[14]= <|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[15]:= DistanceIndex[NewickToTree["(((A:1)B:2)C:3)D;"]]
Out[15]= <|D → 1, C → 4, B → 6, A → 7|>
```

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

```
In[16]:= DistanceIndex[NewickToTree["(((A:1)B:2)C:3)D:0;"]]
Out[16]= <|D → 0, C → 3, B → 5, A → 6|>
```

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

```
In[17]:= 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[18]= <|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[19]:= AncestorIndex[tree]
Out[19]= <|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[20]:= ChildIndex[tree]

Out[20]= <|F → {A, B, E}, E → {C, D}, D → {}, C → {G, H, I}, I → {}, H → {}, G → {}, B → {}, A → {}|>
```

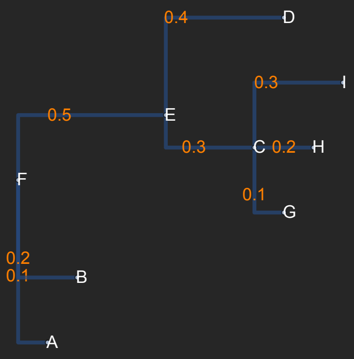
`BranchLengthIndex` lists the branch length values for each vertex (i.e. the distance of the vertex to its parent node).

```
In[21]:= BranchLengthIndex[tree]

Out[21]= <|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[22]:= 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[23]=
```



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

```
In[24]:= TreeDistanceMatrix[tree] // MatrixForm

Out[24]//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.

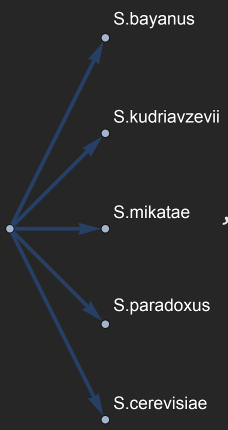
```
In[25]:= v = {"F", "F", "A", "H"};
         MatrixForm[TreeDistanceMatrix[tree, v], TableHeadings → {v, v}]

Out[26]//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

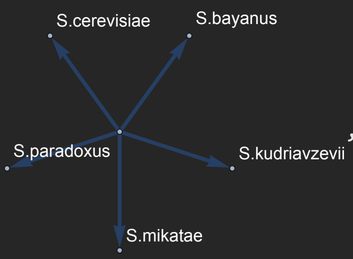
Display trees as graphs subject to different layouts.

```
In[27]:= 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}}]
         }
```

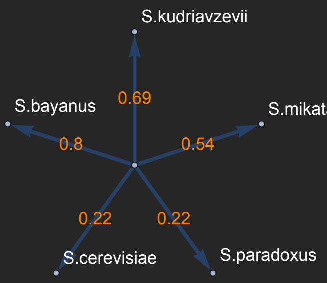
Out[28]= {



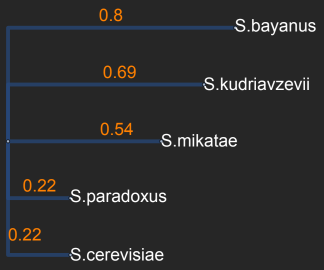
,



,



,



}

`Cladogram` is a graph object.

```
In[29]:= Head[Cladogram[tree]]
Out[29]= Graph
```

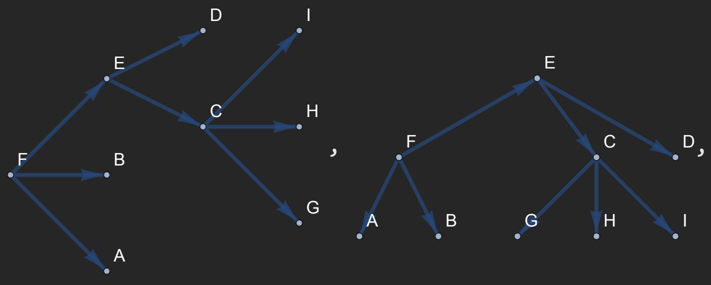
A `Cladogram` is a directed graph, with directed edges.

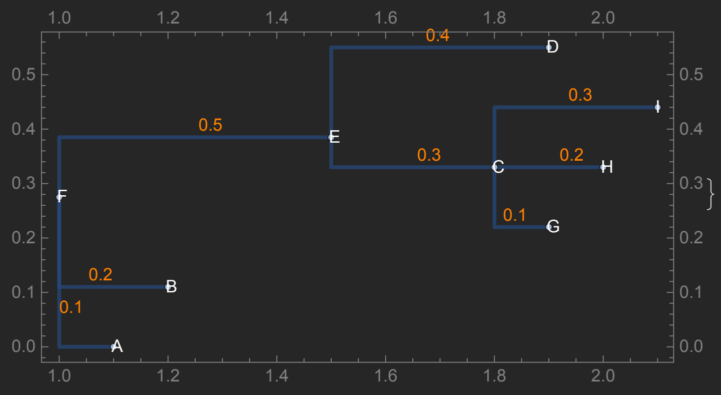
```
In[30]:= DirectedGraphQ[Cladogram[tree]]
         Cladogram[tree] // EdgeList
Out[30]= True
Out[31]= {1 ⧴ S.cerevisiae, 1 ⧴ S.paradoxus, 1 ⧴ S.mikatae, 1 ⧴ S.kudriavzevii, 1 ⧴ S.bayanus}
```

Display the tree in different layouts.

```
In[32]:= 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[33]= {





}

Conversion to/from Newick format

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

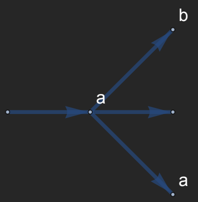
```
In[34]:= old = "(((a,(,,),a3)b))d:1.4;";
         new = TreeToNewick[NewickToTree[old]]
         old === new

Out[35]= (((a,(,,),a3)b))d:1.4;

Out[36]= True
```

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

```
In[37]:= tree = NewickToTree["((a,,b)a)"]

Out[37]= Tree[1, {Tree[a, {2, 3, b}]}]
```

Labels are nevertheless retained and can be accessed by `VertexLabels`.

```
In[38]:= g = TreeToGraph[tree, ImageSize → Tiny]
Out[38]=
```



```
In[39]:= PropertyValue[g, VertexLabels]
Out[39]= {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.

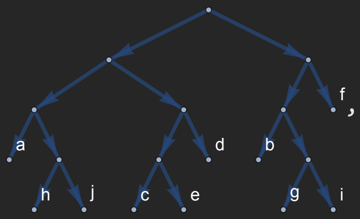
```
In[40]:= cluster = PhylogeneticData["ExampleCluster1"]
         tree = ClusterToTree[cluster]
Out[40]= Cluster[Cluster[Cluster[a, Cluster[h, j, 1.52217, 1, 1], 28.8538, 1, 2],
           Cluster[Cluster[c, e, 10.1371, 1, 1], d, 22.0063, 2, 1], 47.1129, 3, 3],
          Cluster[Cluster[b, Cluster[g, i, 2.5374, 1, 1], 5.73533, 1, 2], f, 13.6197, 3, 1],
          64.5168, 6, 4]
Out[41]= 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.

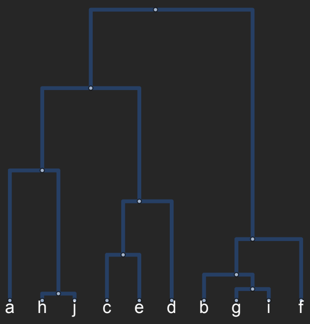
```
In[42]:= BinaryTreeQ[tree]
Out[42]= True
```

Visualize the tree.

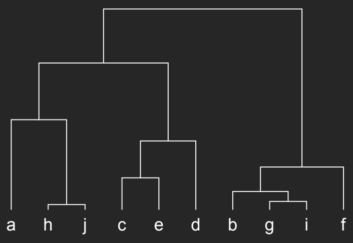
```
In[43]:= {TreeToGraph[tree, GraphLayout → {"LayeredEmbedding", Orientation → Top}],
          Cladogram[tree, Orientation → Top, ImageSize → Small, ImagePadding → 10],
          DendrogramPlot[cluster, Orientation → Top]}
Out[43]= {
```



,



,



}

Convert back to cluster format.

```
In[44]:= new = TreeToCluster[tree]

Out[44]= 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[45]:= {cluster === new, cluster == new}

Out[45]= {False, True}
```

Subtree

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

```
In[46]:= 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[46]= Tree[F, {A, B, Tree[E, {Tree[C, {G, H, I}], D}]}]

Out[47]= 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[48]:= Subtree[tree, "H"]

Out[48]= Tree[H, {}]
```

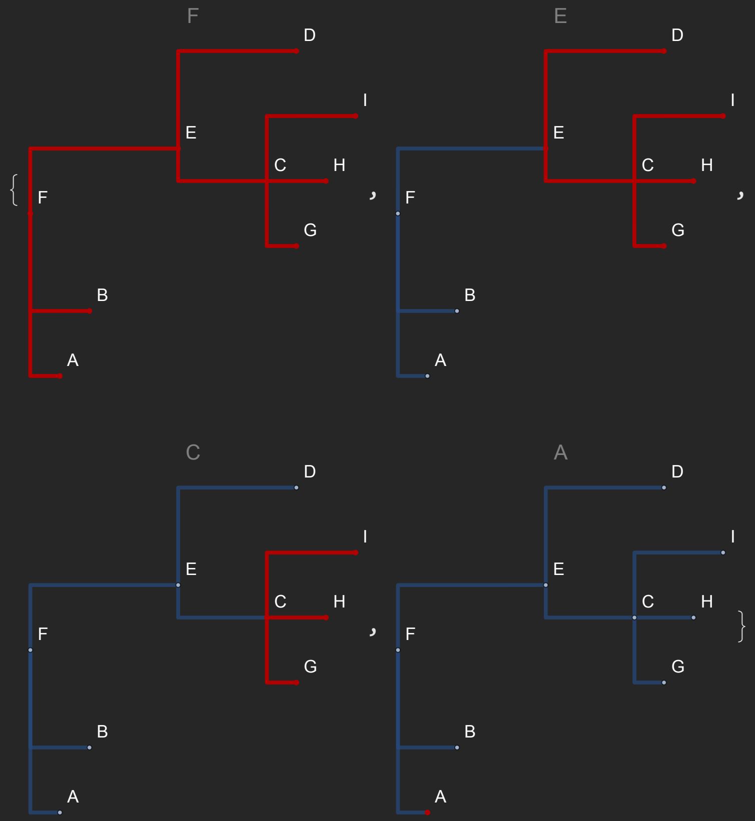
If the vertex is not present in the tree, `Missing` is returned.

```
In[49]:= Subtree[tree, "X"]

Out[49]= Missing[NotFound]
```

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

```
In[50]:= Cladogram[tree, GraphHighlight → Subtree[tree, #],
            PlotLabel → #, VertexLabels → "Name"] & /@ {"F", "E", "C", "A"}
```

Out[50]=



Supertree

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

```
In[51]:= 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[51]= Tree[F, {A, B, Tree[E, {Tree[C, {G, H, I}], D}]}]

In[52]:= Supertree[tree, {"H", "I"}]

Out[52]= Tree[C, {G, H, I}]
```

`Supertree` works with any vertex, leaf or internal node.

```
In[53]:= Supertree[tree, {"H", "A", "C"}]

Out[53]= 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[54]:= Supertree[tree, "G"]

Out[54]= Tree[G, {}]
```

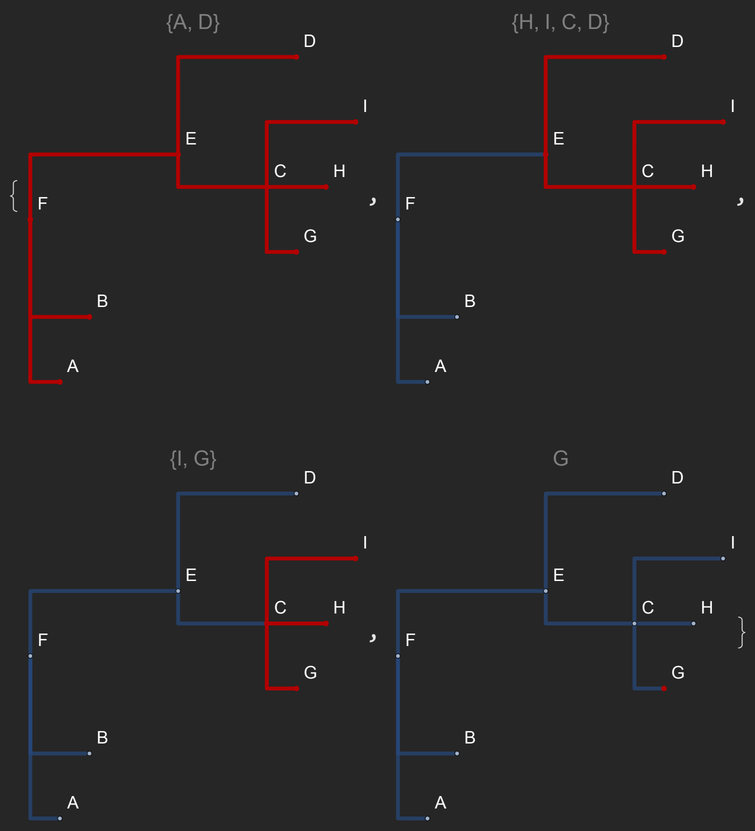
An unknown vertex does not have a supertree.

```
In[55]:= Supertree[tree, "X"]

Out[55]= Missing[NotFound]
```

Visualize supertrees as highlighted subgraphs.

```
In[56]:= Cladogram[tree, ImageSize → Small, GraphHighlight → Supertree[tree, #], PlotLabel → #,
            VertexLabels → "Name"] & /@ {{"A", "D"}, {"H", "I", "C", "D"}, {"I", "G"}, "G"}
```

Out[56]=



CommonAncestor

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

```
In[57]:= 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[57]= Tree[F, {A, B, Tree[E, {Tree[C, {G, H, I}], D}]}]

Out[58]= C
```

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

```
In[59]:= CommonAncestor[tree, {"H", "G", "D"}]

Out[59]= E
```

A common ancestor of a single node is itself.

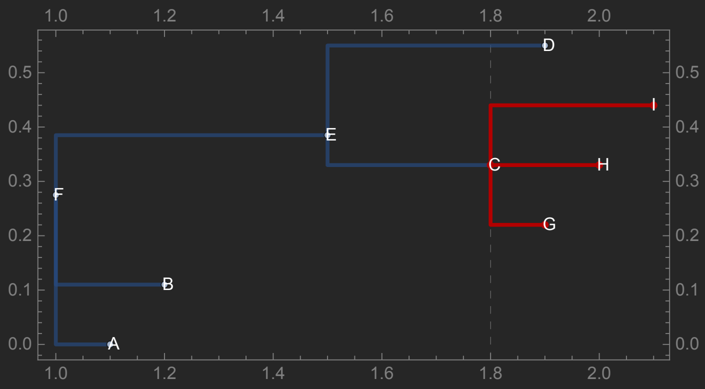
```
In[60]:= CommonAncestor[tree, "E"]

Out[60]= 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[61]:= 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[61]= Tree[F, {A, B, Tree[E, {Tree[C, {G, H, I}], D}]}]

Out[62]= 1.8
```

```
In[63]:= Cladogram[tree, Frame → True, LayerSizeFunction → (1 # &),
          GridLines → {{{d, Dashed}}, {}}, GraphHighlight → Supertree[tree, {"H", "G"}]]
```

Out[63]=


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[64]:= DivergenceTime[tree, {"H", "G", "D"}]

Out[64]= 1.5
```

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

```
In[65]:= DivergenceTime[tree, "E"]

Out[65]= 1.5
```

Paths and distances

The path between two nodes is always the shortest path.

```
In[66]:= 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[67]= {C → G, E → C, E → D}
```

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

```
In[68]:= TreePath[tree, {"G", "D"}]

Out[68]= {C → G, E → C, E → D}
```

A single vertex returns a path from the root.

```
In[69]:= TreePath[tree, "G"]
Out[69]= {C → G, E → C, F → E}
```

A path between a vertex and itself returns the vertex.

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

The path to an unknown vertex is an empty list.

```
In[71]:= TreePath[tree, "G", "X"]
Out[71]= { }
```

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

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

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

```
In[73]:= TreeDistance[tree, {"G", "D"}]
Out[73]= 0.8
```

A single vertex returns its absolute distance from the root.

```
In[74]:= TreeDistance[tree, "G"]
Out[74]= 0.9
```

A distance between a vertex and itself is always 0.

```
In[75]:= TreeDistance[tree, "G", "G"]
Out[75]= 0
```

Distance to an unknown vertex is infinite.

```
In[76]:= TreeDistance[tree, "G", "X"]
Out[76]= ∞
```

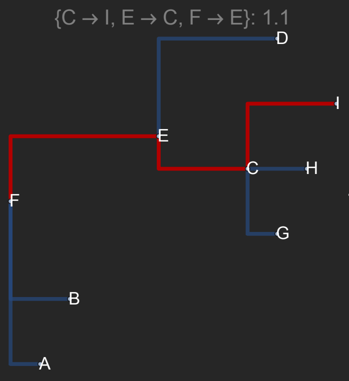
The distance of two nodes is always a nonnegative value, regardless of the direction.

```
In[77]:= {TreeDistance[tree, {"F", "I"}], TreeDistance[tree, {"I", "F"}]}
Out[77]= {1.1, 1.1}
```

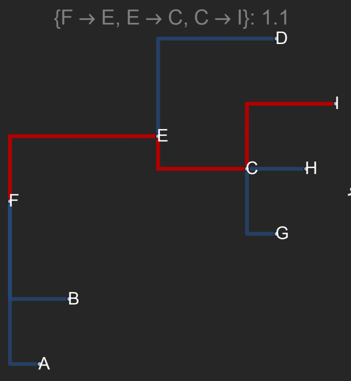
Examples.

```
In[78]:= 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}]
```

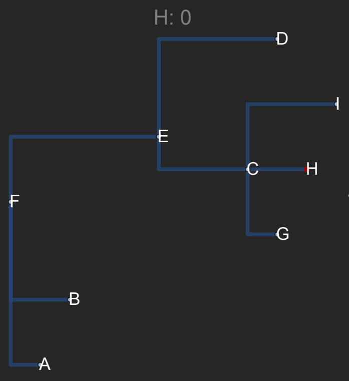
Out[81]= {



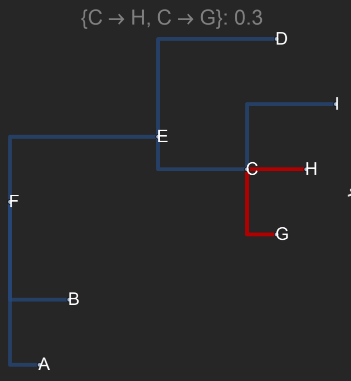
,



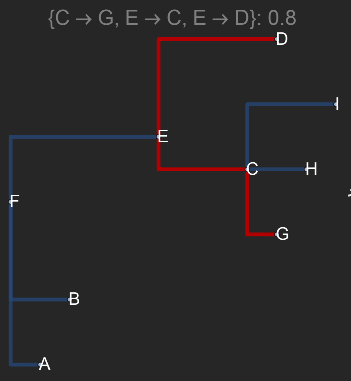
,



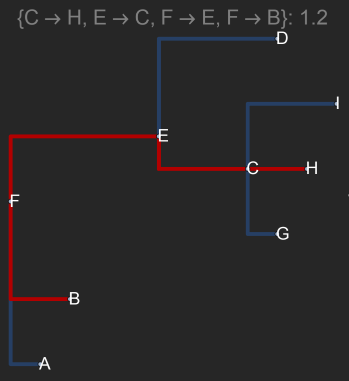
,



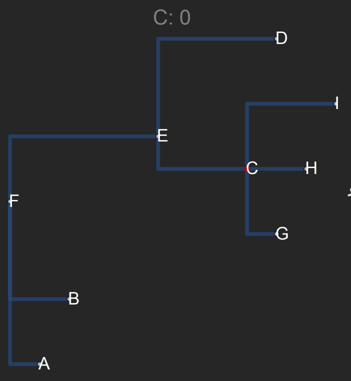
,



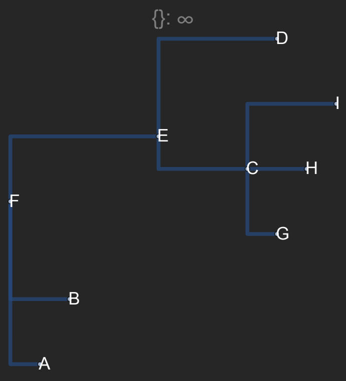
,



,



,



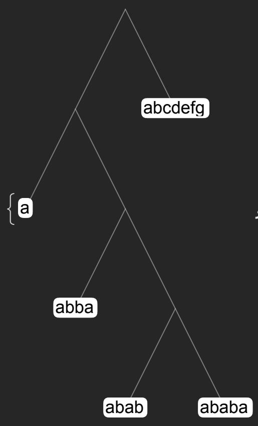
}

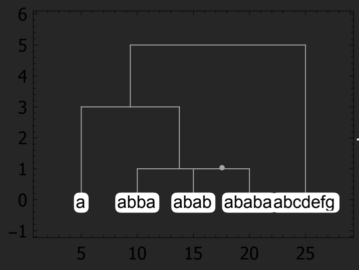
Clusters

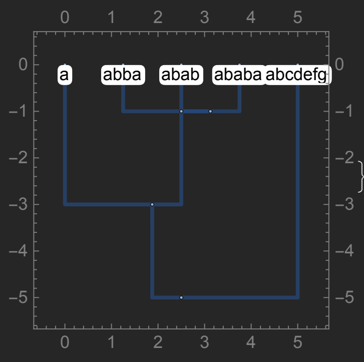
Clusterize a list of strings based on their similarity. `Cladogram` cannot work with such lists directly, but can accept the graph produced by `ClusteringTree`. Since `Dendrogram` returns a static `Graphics` object instead of a rich `Graph`, `Cladogram` cannot deal with it.

```
In[82]:= 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[83]= { , , }



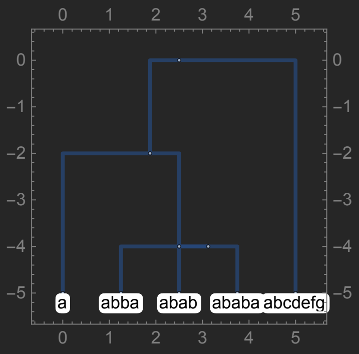




Since `Cladogram` measures distances from the root instead of (dis)similarity between vertices, the result is a mirrored plot. To display it correctly, invert the vertex weights that represent the vertical absolute distance for each node.

```
In[84]:= w = PropertyValue[c, VertexWeight]
         Cladogram[g, VertexWeight → (Max@w - w), Orientation → Top,
          Frame → True, ImagePadding → All, ImageSize → Small,
          PlotRangePadding → Scaled[.1], VertexLabelStyle → Black]

Out[84]= {5., 3., 0, 1., 0, 1., 0, 0, 0}
```

Out[85]=

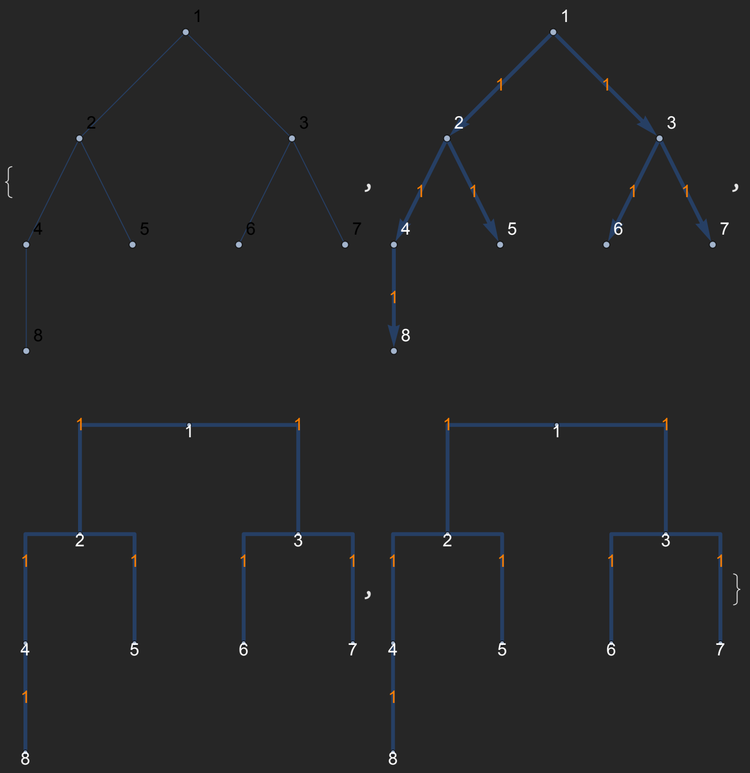


Graphs

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

```
In[86]:= 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[87]= Tree[1, {Tree[2, {Tree[4, {8}], 5}], Tree[3, {6, 7}]}]
```

Out[88]= {

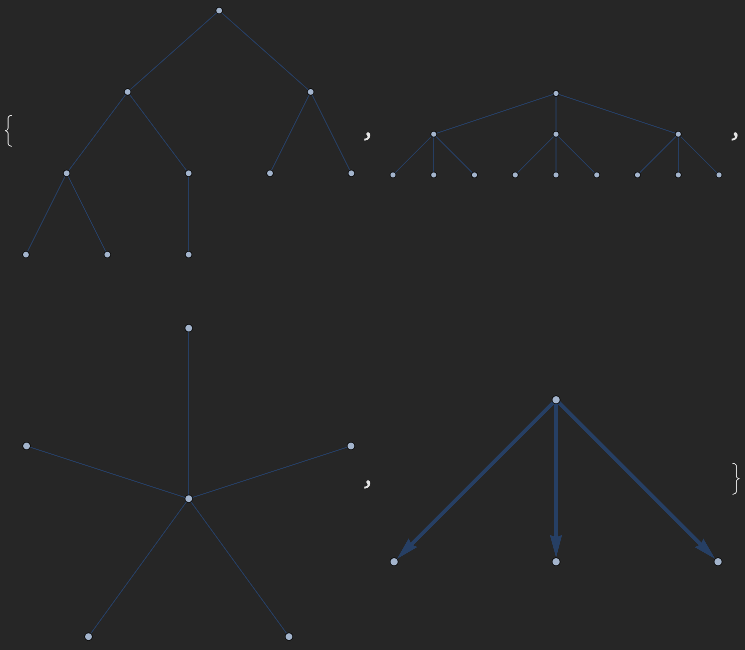


}

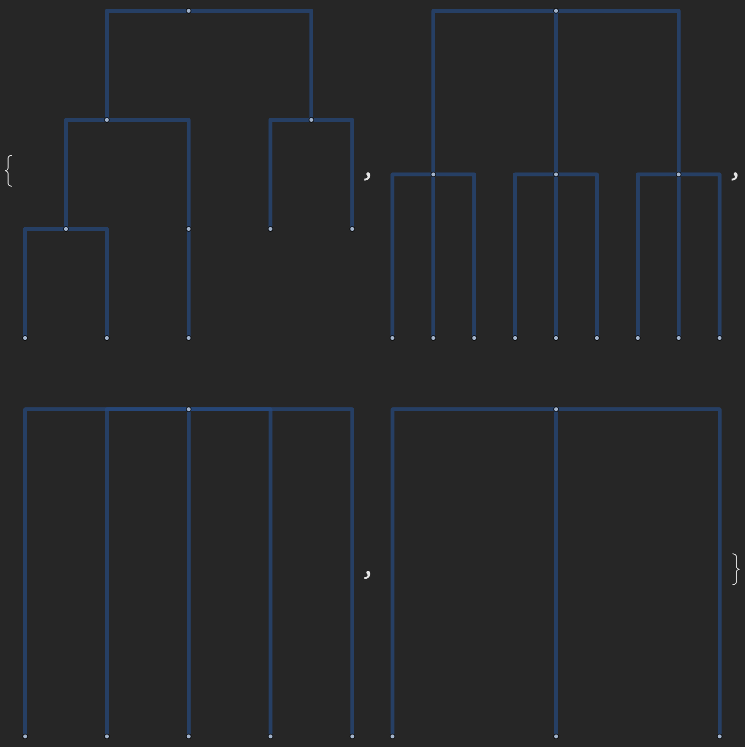
Display some example graphs as cladograms.

```
In[89]:= graphs = {KaryTree[10], CompleteKaryTree[3, 3],
           StarGraph[6], TreeGraph[{1 → 2, 1 → 3, 1 → 4}]}
         Cladogram[#, Orientation → Top] & /@ graphs
```

Out[89]=



Out[90]=



Phylogenetic data

List the available phylogenetic examples stored within `PhylogeneticData`.

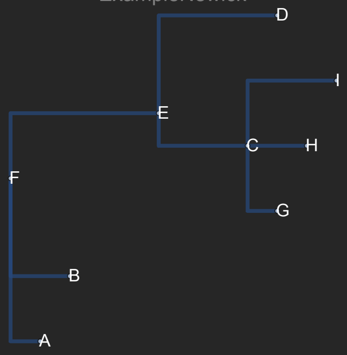
```
In[91]:= all = PhylogeneticData["Names"]

Out[91]= {ExampleNewick, ExampleNewick1, ExampleNewick2, ExampleNewick3, ExampleNewick6,
          ExampleNewick7, ExampleNewick8, ExampleNewick9, ExampleNewickEmpty1,
          ExampleNewickEmpty2, ExampleNewickEmpty3, ExampleNewickEmpty4,
          ExampleCluster1, ExampleCluster2, NewickMammal1, NewickMammal2,
          NewickMammal3, NewickHominoid, NewickSaccharomyces, NewickEukarya}
```

Display example data.

```
In[92]:= Cladogram[If[StringMatchQ[#, "*Newick*"], NewickToTree, ClusterToTree]@
             PhylogeneticData[#], ImageSize → Small, PlotLabel → #] & /@ Most[all]
```

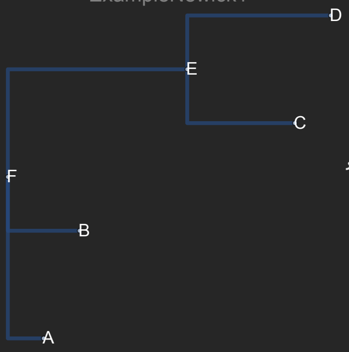
Out[92]= {

ExampleNewick



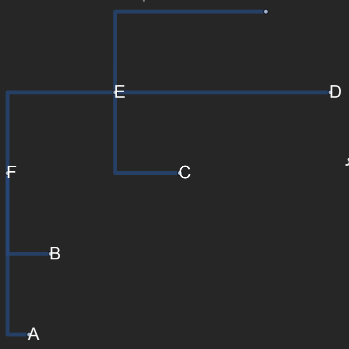
,

ExampleNewick1



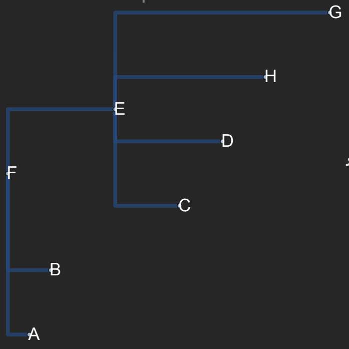
,

ExampleNewick2



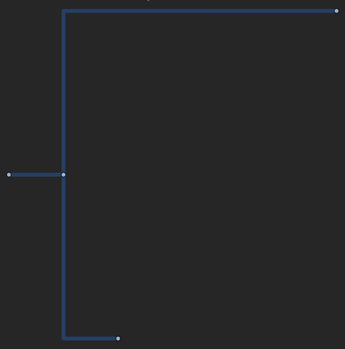
,

ExampleNewick3



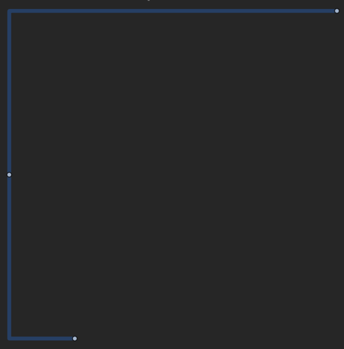
,

ExampleNewick6



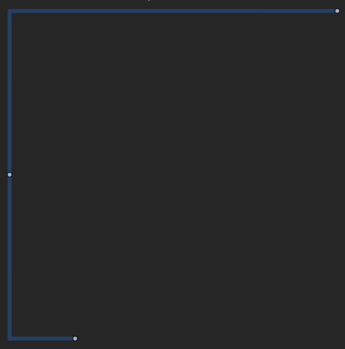
,

ExampleNewick7

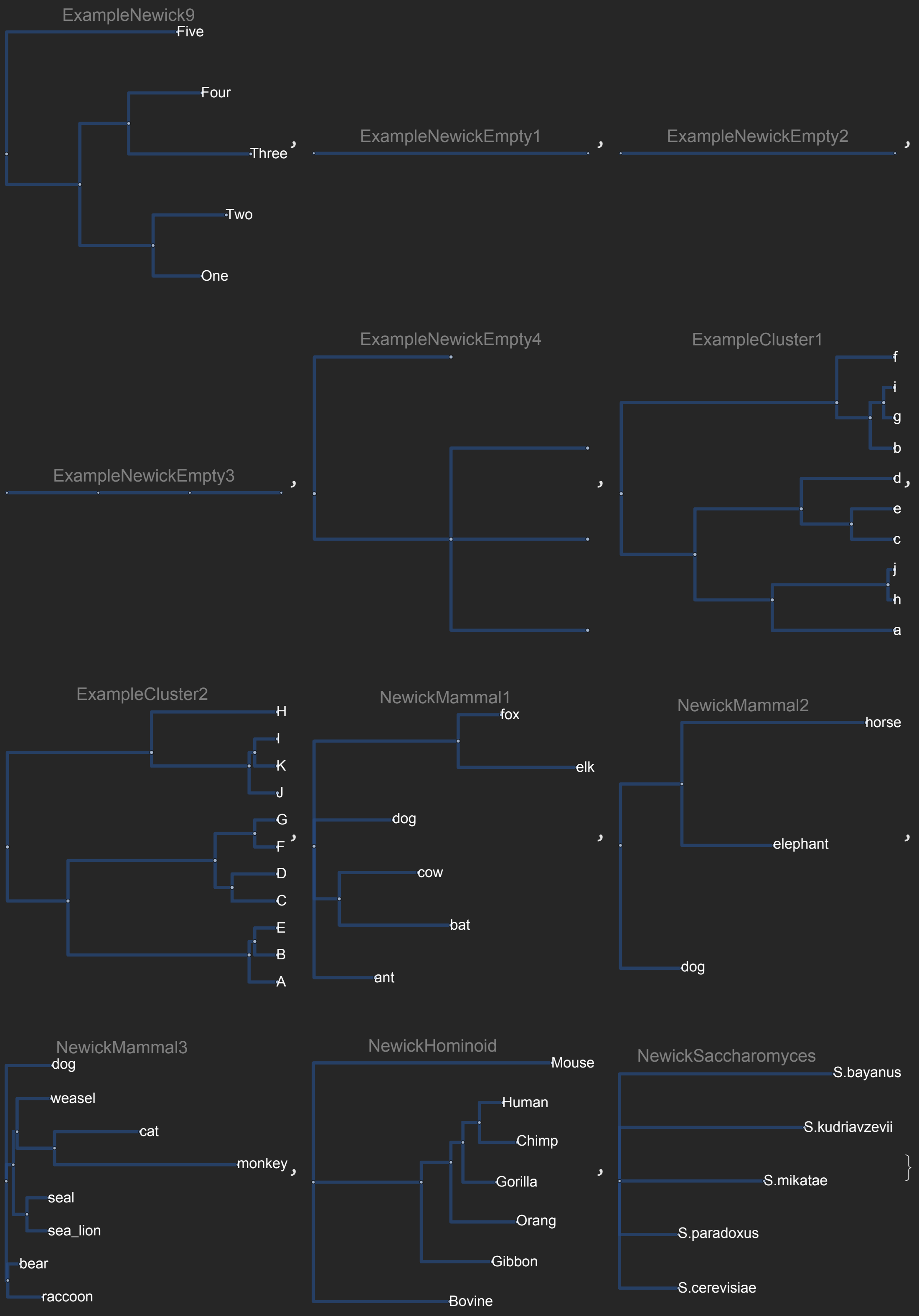


,

ExampleNewick8



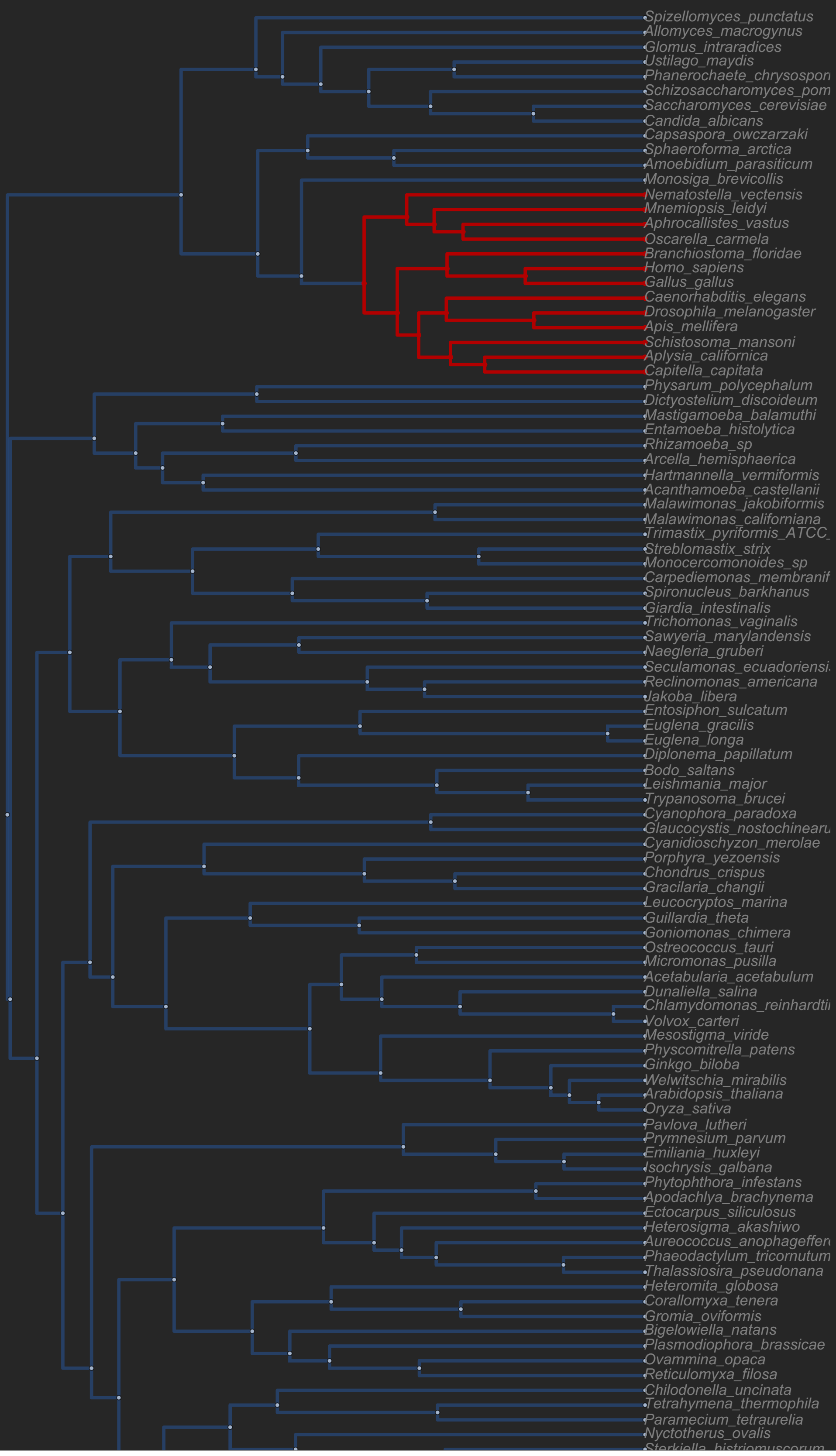
,

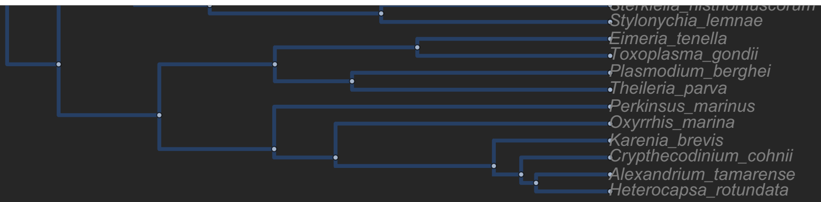


A complex tree, showing the diversificaion of Eukarya, with Metazoa highlighted (from *Parfrey et al. 2011*).

In[93]:=
```
tree = NewickToTree[PhylogeneticData["NewickEukarya"]];
Cladogram[tree, ImageSize → 500,
 ImagePadding → {{1, 100}, {1, 1}}, LayerSizeFunction → (1 / 5 # &),
 VertexSize → .3, VertexLabelStyle → Directive[Gray, 9, Italic],
 GraphHighlight → Supertree[tree, {"Nematostella_vectensis", "Homo_sapiens"}]]
```

Out[94]=




Stylonychia_lemnae
Eimeria_tenella
Toxoplasma_gondii
Plasmodium_berghei
Theileria_parva
Perkinsus_marinus
Oxyrrhis_marina
Karenia_brevis
Crypthecodinium_cohnii
Alexandrium_tamarense
Heterocapsa_rotundata