Package ‘TreeTools’

October 6, 2021

Title Create, Modify and Analyse Phylogenetic Trees
Version 1.5.1
License GPL (>= 3)
Copyright Incorporates C/C++ code from 'ape' by Emmanuel Paradis
<doi:10.1093/bioinformatics/bty633>

Description Efficient implementations of functions for the creation, modification and analysis of phylogenetic trees. Applications include:
generation of trees with specified shapes;
tree rearrangement;
analysis of tree shape;
rooting of trees and extraction of subtrees;
calculation and depiction of split support;
calculation of ancestor-descendant relationships,
artificial extinction (Asher & Smith, 2021) <doi:10.1093/sysbio/syab072>;
and analysis of splits and cladistic information.


BugReports https://github.com/ms609/TreeTools/issues/

SystemRequirements C++14

Depends R (>= 3.4.0), ape (>= 5.0),
Imports bit64, colorspace, fastmatch (>= 1.1.3), phangorn (>= 2.2.1), R.cache, Rdpack (>= 0.7),
Suggests knitr, Rcpp, markdown, shiny, testthat (>= 3.0), vdiffr (>= 1.0.0),
Config/Needs/coverage covr
Config/Needs/memcheck devtools, rcmdcheck
R topics documented:

- Config/Needs/metadata: codemetaR
- Config/Needs/revdeps: revdepcheck
- Config/Needs/website: pkgdown
- Config/testthat/parallel: false
- Config/testthat/edition: 3
- LinkingTo: Rcpp
- RdMacros: Rdpack
- LazyData: true
- ByteCompile: true
- Encoding: UTF-8
- Language: en-GB
- VignetteBuilder: knitr
- RoxygenNote: 7.1.2
- NeedsCompilation: yes

Author: Martin R. Smith [aut, cre, cph]
Emmanuel Paradis [cph]

Maintainer: Martin R. Smith <martin.smith@durham.ac.uk>

Repository: CRAN

Date/Publication: 2021-10-06 12:00:12 UTC

R topics documented:

- AddTip .......................................................... 4
- ApeTime ......................................................... 6
- ArtificialExtinction ......................................... 6
- as.multiPhylo .................................................. 9
- as.Newick ....................................................... 10
- brewer .......................................................... 11
- CharacterInformation ....................................... 11
- CladeSizes ..................................................... 12
- CladisticInfo .................................................. 13
- CollapseNode ............................................... 14
- Consensus ..................................................... 16
- ConsensusWithout .......................................... 17
- ConstrainedNJ ............................................... 18
- DescendantEdges ............................................ 19
- DoubleFactorial ............................................. 20
- doubleFactorials ........................................... 21
- DropTip ........................................................ 22
- EdgeAncestry ................................................ 23
- EdgeDistances ............................................... 24
<table>
<thead>
<tr>
<th>R topics documented</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>EndSentence</td>
<td>25</td>
</tr>
<tr>
<td>EnforceOutgroup</td>
<td>25</td>
</tr>
<tr>
<td>GenerateTree</td>
<td>26</td>
</tr>
<tr>
<td>ImposeConstraint</td>
<td>28</td>
</tr>
<tr>
<td>LabelSplits</td>
<td>29</td>
</tr>
<tr>
<td>LeafLabelInterchange</td>
<td>30</td>
</tr>
<tr>
<td>ListAncestors</td>
<td>31</td>
</tr>
<tr>
<td>Lobo.data</td>
<td>33</td>
</tr>
<tr>
<td>logDoubleFactorials</td>
<td>33</td>
</tr>
<tr>
<td>MakeTreeBinary</td>
<td>34</td>
</tr>
<tr>
<td>match</td>
<td>35</td>
</tr>
<tr>
<td>MorphoBankDecode</td>
<td>36</td>
</tr>
<tr>
<td>MRCA</td>
<td>37</td>
</tr>
<tr>
<td>MSTEdges</td>
<td>38</td>
</tr>
<tr>
<td>N1Spr</td>
<td>39</td>
</tr>
<tr>
<td>NDescendants</td>
<td>40</td>
</tr>
<tr>
<td>NewickTree</td>
<td>41</td>
</tr>
<tr>
<td>NJTree</td>
<td>41</td>
</tr>
<tr>
<td>NodeDepth</td>
<td>42</td>
</tr>
<tr>
<td>NodeOrder</td>
<td>43</td>
</tr>
<tr>
<td>NPartitionPairs</td>
<td>44</td>
</tr>
<tr>
<td>NRooted</td>
<td>45</td>
</tr>
<tr>
<td>nRootedShapes</td>
<td>47</td>
</tr>
<tr>
<td>NSplits</td>
<td>48</td>
</tr>
<tr>
<td>NTip</td>
<td>49</td>
</tr>
<tr>
<td>PairwiseDistances</td>
<td>50</td>
</tr>
<tr>
<td>PolarizeSplits</td>
<td>51</td>
</tr>
<tr>
<td>print.TreeNumber</td>
<td>52</td>
</tr>
<tr>
<td>ReadCharacters</td>
<td>52</td>
</tr>
<tr>
<td>ReadTntTree</td>
<td>54</td>
</tr>
<tr>
<td>Renumber</td>
<td>57</td>
</tr>
<tr>
<td>RenumberTips</td>
<td>58</td>
</tr>
<tr>
<td>RightmostCharacter</td>
<td>59</td>
</tr>
<tr>
<td>RootNode</td>
<td>60</td>
</tr>
<tr>
<td>RootTree</td>
<td>61</td>
</tr>
<tr>
<td>sapply64</td>
<td>62</td>
</tr>
<tr>
<td>SingleTaxonTree</td>
<td>63</td>
</tr>
<tr>
<td>SortTree</td>
<td>64</td>
</tr>
<tr>
<td>SplitFrequency</td>
<td>65</td>
</tr>
<tr>
<td>SplitInformation</td>
<td>67</td>
</tr>
<tr>
<td>SplitMatchProbability</td>
<td>69</td>
</tr>
<tr>
<td>Splits</td>
<td>70</td>
</tr>
<tr>
<td>SplitsInBinaryTree</td>
<td>71</td>
</tr>
<tr>
<td>Stemwardness</td>
<td>73</td>
</tr>
<tr>
<td>StringToPhyDat</td>
<td>75</td>
</tr>
<tr>
<td>Subsplit</td>
<td>77</td>
</tr>
<tr>
<td>Subtree</td>
<td>78</td>
</tr>
<tr>
<td>SupportColour</td>
<td>79</td>
</tr>
</tbody>
</table>
**AddTip**

Add a tip to a phylogenetic tree

**Description**

AddTip() adds a tip to a phylogenetic tree at a specified location.

**Usage**

AddTip(
  tree,
  where = sample.int(tree$Nnode * 2 + 2L, size = 1) - 1L,
  label = "New tip",
  edgeLength = 0,
  lengthBelow = NULL,
  nTip = NTip(tree),
  nNode = tree$Nnode,
  rootNode = RootNode(tree)
)

AddTipEverywhere(tree, label = "New tip", includeRoot = FALSE)

**Arguments**

- **tree** A tree of class phylo.
- **where** The node or tip that should form the sister taxon to the new node. To add a new tip at the root, use where = 0. By default, the new tip is added to a random edge.
- **label** Character string providing the label to apply to the new tip.
- **edgeLength** Numeric specifying length of new edge
**AddTip**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lengthBelow</td>
<td>Numeric specifying length below neighbour at which to graft new edge. Values greater than the length of the edge will result in negative edge lengths. If NULL, the default, the new tip will be added at the midpoint of the broken edge. If inserting at the root (where = 0), a new edge of length lengthBelow will be inserted.</td>
</tr>
<tr>
<td>nTip, nNode, rootNode</td>
<td>Optional integer vectors specifying number of tips and nodes in tree, and index of root node. Not checked for correctness: specifying values here trades code safety for a nominal speed increase.</td>
</tr>
<tr>
<td>includeRoot</td>
<td>Logical; if TRUE, each position adjacent to the root edge is considered to represent distinct edges; if FALSE, they are treated as a single edge.</td>
</tr>
</tbody>
</table>

**Details**

AddTip() extends `bind.tree`, which cannot handle single-taxon trees.

AddTipEverywhere() adds a tip to each edge in turn.

**Value**

AddTip() returns a tree of class `phylo` with an additional tip at the desired location.

AddTipEverywhere() returns a list of class `multiPhylo` containing the trees produced by adding label to each edge of `tree` in turn.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Add one tree to another: `bind.tree()`

Other tree manipulation: `CollapseNode()`, `ConsensusWithout()`, `DropTip()`, `EnforceOutgroup()`, `ImposeConstraint()`, `LeafLabelInterchange()`, `MakeTreeBinary()`, `RenumberTips()`, `RenumberTree()`, `Renumber()`, `RootTree()`, `SingleTaxonTree()`, `SortTree()`, `Subtree()`

**Examples**

```r
eroot <- BalancedTree(10)
ape::nodelabels()
ape::nodelabels(15, 15, bg='green')

plot(AddTip(root, 15, "NEW_TIP"))
oldPar <- par(mfrow = c(2, 4), mar = rep(0.3, 4), cex = 0.9)

backbone <- BalancedTree(4)
# Treating the position of the root as instructive:
additions <- AddTipEverywhere(backbone, includeRoot = TRUE)
xx <- lapply(additions, plot)
```
par(mfrow=c(2, 3))
# Don't treat root edges as distinct:
additions <- AddTipEverywhere(backbone, includeRoot = FALSE)
xx <- lapply(additions, plot)
par(oldPar)

---

### ApeTime

*Read modification time from 'ape' Nexus file*

**Description**

ApeTime() reads the time that a tree written with 'ape' was modified, based on the comment in the Nexus file.

**Usage**

```
ApeTime(filepath, format = "double")
```

**Arguments**

- `filepath` Character string specifying path to the file.
- `format` Format in which to return the time: 'double' as a sortable numeric; any other value to return a string in the format YYYY-MM-DD hh:mm:ss.

**Value**

ApeTime() returns the time that the specified file was created by ape, in the format specified by format.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

---

### ArtificialExtinction

*Artificial Extinction*

**Description**

Remove tokens that do not occur in a fossil 'template' taxon from a living taxon, to simulate the process of fossilization in removing data from a phylogenetic dataset.
Usage

ArtificialExtinction(
  dataset,
  subject,
  template,
  replaceAmbiguous = "ambig",
  replaceCoded = "original",
  replaceAll = TRUE,
  sampleFrom = NULL
)

## S3 method for class 'matrix'
ArtificialExtinction(
  dataset,
  subject,
  template,
  replaceAmbiguous = "ambig",
  replaceCoded = "original",
  replaceAll = TRUE,
  sampleFrom = NULL
)

## S3 method for class 'phyDat'
ArtificialExtinction(
  dataset,
  subject,
  template,
  replaceAmbiguous = "ambig",
  replaceCoded = "original",
  replaceAll = TRUE,
  sampleFrom = NULL
)

ArtEx(
  dataset,
  subject,
  template,
  replaceAmbiguous = "ambig",
  replaceCoded = "original",
  replaceAll = TRUE,
  sampleFrom = NULL
)

Arguments

dataset Phylogenetic dataset of class phyDat or matrix.
subject Vector identifying subject taxa, by name or index.
template Character or integer identifying taxon to use as a template.
replaceAmbiguous, replaceCoded

Character specifying whether tokens that are ambiguous (?) or coded (not ?) in
the fossil template should be replaced with:

- original: Their original value; i.e. no change;
- ambiguous: The ambiguous token, ?;
- binary: The tokens 0 or 1, with equal probability;
- uniform: One of the tokens present in sampleFrom, with equal probability;
- sample: One of the tokens present in sampleFrom, sampled according to
  their frequency.

replaceAll

Logical: if TRUE, replace all tokens in a subject; if FALSE, leave any ambiguous
tokens (?) ambiguous.

sampleFrom

Vector identifying a subset of characters from which to sample replacement to-
kens. If NULL, replacement tokens will be sampled from the initial states of all
taxa not used as a template (including the subjects).

Details

Further details are provided in Asher and Smith (2021).

Note: this simple implementation does not account for character contingency, e.g. characters whose
absence imposes inapplicable or absent tokens on dependent characters.

Value

A dataset with the same class as dataset in which entries that are ambiguous in template are made
ambiguous in subject.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References


Examples

```r
set.seed(1)
dataset <- matrix(c(sample(0:2, 4 * 8, TRUE),
  '0', '0', rep('?', 6)), nrow = 5,
  dimnames = list(c(LETTERS[1:4], 'FOSSIL'),
                  paste('char', 1:8)), byrow = TRUE)
artex <- ArtificialExtinction(dataset, c('A', 'C', 'FOSSIL'))
```
as.multiPhylo

Convert object to multiPhylo class

Description

Converts representations of phylogenetic trees to an object of the 'ape' class multiPhylo.

Usage

as.multiPhylo(x)

## S3 method for class 'phylo'
as.multiPhylo(x)

## S3 method for class 'list'
as.multiPhylo(x)

## S3 method for class 'phyDat'
as.multiPhylo(x)

## S3 method for class 'Splits'
as.multiPhylo(x)

Arguments

x Object to be converted

Value

as.multiPhylo returns an object of class multiPhylo

as.multiPhylo.phyDat() returns a list of trees, each corresponding to the partitions implied by each non-ambiguous character in x.

Examples

as.multiPhylo(BalancedTree(8))
as.multiPhylo(list(BalancedTree(8), PectinateTree(8)))
data('Lobo')
as.multiPhylo(Lobo.phy)
Write a phylogenetic tree in Newick format

Description

as.Newick() creates a character string representation of a phylogenetic tree, in the Newick format, using R’s internal tip numbering. Use RenumberTips() to ensure that the internal numbering follows the order you expect.

Usage

as.Newick(x)

## S3 method for class 'phylo'
as.Newick(x)

## S3 method for class 'list'
as.Newick(x)

## S3 method for class 'multiPhylo'
as.Newick(x)

Arguments

x Object to convert to Newick format. See Usage section for supported classes.

Value

as.Newick() returns a character string representing tree in Newick format.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

• Retain leaf labels: NewickTree()
• Change R’s internal numbering of leaves: RenumberTips()
• Write tree to text or file: ape::write.tree()

Examples

trees <- list(BalancedTree(1:8), PectinateTree(8:1))
trees <- lapply(trees, RenumberTips, 1:8)
as.Newick(trees)
## Brewer palettes

### Description
A list of eleven Brewer palettes containing one to eleven colours that are readily distinguished by colourblind viewers, followed by a twelfth 12-colour palette adapted for colour blindness.

### Usage
```r
brewer
```

### Format
An object of class `list` of length 12.

### Source
- ColourBrewer2.org
- Martin Krzywinski

### Examples
```r
data("brewer", package="TreeTools")
plot(0, type="n", xlim=c(1, 12), ylim=c(12, 1),
  xlab = 'Colour', ylab='Palette')
for (i in seq_along(brewer)) text(seq_len(i), i, col=brewer[[i]])
```

## CharacterInformation

### Description
`CharacterInformation()` calculates the phylogenetic information content of a given character.

### Usage
```r
CharacterInformation(tokens)
```

### Arguments
- **tokens**: Character vector specifying the tokens assigned to each taxon for a character. Example: `c(0,0,0,1,1,1,'?', '-', ')'. Note that ambiguous tokens such as (01) are not supported, and should be replaced with `?`.
CladeSizes

Value

CharacterInformation() returns a numeric specifying the phylogenetic information content of the character (sensu Steel & Penny 2006), in bits.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References


See Also

Other split information functions: SplitInformation(), SplitMatchProbability(), TreesMatchingSplit(), UnrootedTreesMatchingSplit()

CladeSizes

Clade sizes

Description

CladeSizes() reports the number of nodes in each clade in a tree.

Usage

CladeSizes(tree, internal = FALSE, nodes = NULL)

Arguments

tree         A tree of class phylo.
internal     Logical specifying whether internal nodes should be counted towards the size of each clade.
nodes        Integer specifying indices of nodes at the base of clades whose sizes should be returned. If unspecified, counts will be provided for all nodes (including leaves).

Value

CladeSizes() returns the number of nodes (including leaves) that are descended from each node, not including the node itself.

See Also

Other tree navigation: AncestorEdge(), DescendantEdges(), EdgeAncestry(), EdgeDistances(), ListAncestors(), MRCA(), NDescendants(), NodeDepth(), NodeOrder(), NonDuplicateRoot(), RootNode()
CladisticInfo

Examples

```r
tree <- BalancedTree(6)
plot(tree)
ape::nodelabels()
CladeSizes(tree, nodes = c(1, 8, 9))
```

---

**CladisticInfo**

*Cladistic information content of a tree*

**Description**


**Usage**

CladisticInfo(x)

PhylogeneticInfo(x)

```r
## S3 method for class 'phylo'
CladisticInfo(x)
```

```r
## S3 method for class 'Splits'
CladisticInfo(x)
```

```r
## S3 method for class 'list'
CladisticInfo(x)
```

```r
## S3 method for class 'multiPhylo'
CladisticInfo(x)
```

PhylogeneticInformation(x)

CladisticInformation(x)

**Arguments**

*x* Tree of class *phylo*, or a list thereof.

**Details**

The CIC is the logarithm of the number of binary trees that include the specified topology. A base two logarithm gives an information content in bits.

The CIC was originally proposed by Rohlf (1982), and formalised, with an information-theoretic justification, by Thorley *et al.* (1998). Steel and Penny (2006) term the equivalent quantity 'phylogenetic information content' in the context of individual characters.
The number of binary trees consistent with a cladogram provides a more satisfactory measure of the resolution of a tree than simply counting the number of edges resolved (Page, 1992).

**Value**

CladisticInfo() returns a numeric giving the cladistic information content of the input tree(s), in bits. If passed a Splits object, it returns the information content of each split in turn.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**References**


**See Also**

Other tree information functions: NRooted(), TreesMatchingTree()
Other tree characterization functions: Consensus(), Stemwardness, TotalCopheneticIndex()

---

**CollapseNode**

*Collapse nodes on a phylogenetic tree*

**Description**

Collapses specified nodes or edges on a phylogenetic tree, resulting in polytomies.

**Usage**

```r
CollapseNode(tree, nodes)
```

```r
## S3 method for class 'phylo'
CollapseNode(tree, nodes)
```

```r
CollapseEdge(tree, edges)
```
**CollapseNode**

**Arguments**

- **tree**: A tree of class `phylo`.
- **nodes, edges**: Integer vector specifying the nodes or edges in the tree to be dropped. (Use `nodelabels()` or `edgelabels()` to view numbers on a plotted tree.)

**Value**

`CollapseNode()` and `CollapseEdge()` return a tree of class `phylo`, corresponding to `tree` with the specified nodes or edges collapsed. The length of each dropped edge will (naively) be added to each descendant edge.

**Author(s)**

Martin R. Smith

**See Also**

Other tree manipulation: `AddTip()`, `ConsensusWithout()`, `DropTip()`, `EnforceOutgroup()`, `ImposeConstraint()`, `LeafLabelInterchange()`, `MakeTreeBinary()`, `RenumberTips()`, `RenumberTree()`, `Renumber()`, `RootTree()`, `SingleTaxonTree()`, `SortTree()`, `Subtree()`

**Examples**

```r
oldPar <- par(mfrow = c(3, 1), mar = rep(0.5, 4))

tree <- as.phylo(898, 7)
tree$edge.length <- 11:22
plot(tree)
nodelabels()
edgelabels()
edgelabels(round(tree$edge.length, 2),
          cex = 0.6, frame = 'n', adj = c(1, -1))

# Collapse by node number
newTree <- CollapseNode(tree, c(12, 13))
plot(newTree)
nodelabels()
edgelabels(round(newTree$edge.length, 2),
          cex = 0.6, frame = 'n', adj = c(1, -1))

# Collapse by edge number
newTree <- CollapseEdge(tree, c(2, 4))
plot(newTree)
p(oldPar)
```
**Consensus**

**Consensus trees**

**Description**

Calculates the consensus of a set of trees.

**Usage**

Consensus(trees, p = 1, check.labels = TRUE)

**Arguments**

trees List of trees, optionally of class multiPhylo.

p Proportion of trees that must contain a split for it to be reported in the consensus. p = 0.5 gives the majority-rule consensus; p = 1 (the default) gives the strict consensus.

check.labels Logical specifying whether to check that all trees have identical labels. Defaults to TRUE, which is slower.

**Value**

Consensus() returns an object of class phylo, rooted as in the first entry of trees.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Other consensus tree functions: ConsensusWithout()

Other tree characterization functions: CladisticInfo(), Stemwardness, TotalCopheneticIndex()

**Examples**

Consensus(as.phylo(0:2, 8))
ConsensusWithout

*Reduced consensus, omitting specified taxa*

**Description**

ConsensusWithout() displays a consensus plot with specified taxa excluded, which can be a useful way to increase the resolution of a consensus tree when a few wildcard taxa obscure a consistent set of relationships. MarkMissing() adds missing taxa as loose leaves on the plot.

**Usage**

ConsensusWithout(trees, tip = character(0), ...)

```r
## S3 method for class 'phylo'
ConsensusWithout(trees, tip = character(0), ...)

## S3 method for class 'multiPhylo'
ConsensusWithout(trees, tip = character(0), ...)

## S3 method for class 'list'
ConsensusWithout(trees, tip = character(0), ...)

MarkMissing(tip, position = "bottomleft", ...)
```

**Arguments**

- `trees` A list of phylogenetic trees, of class `multiPhylo` or `list`.
- `tip` A character vector specifying the names (or numbers) of tips to drop (using `ape::drop.tip()`).
- `...` Additional parameters to pass on to `ape::consensus()` or `legend()`.
- `position` Where to plot the missing taxa. See `legend()` for options.

**Value**

ConsensusWithout() returns a consensus tree (of class `phylo`) without the excluded taxa. MarkMissing() provides a null return, after plotting the specified tips as a legend.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Other tree manipulation: `AddTip()`, `CollapseNode()`, `DropTip()`, `EnforceOutgroup()`, `ImposeConstraint()`, `LeafLabelInterchange()`, `MakeTreeBinary()`, `RenumberTips()`, `RenumberTree()`, `Renumber()`, `RootTree()`, `SingleTaxonTree()`, `SortTree()`, `Subtree()"
Other tree properties: \texttt{NSplits()}, \texttt{NTip()}, \texttt{SplitsInBinaryTree()}, \texttt{TipLabels()}, \texttt{TreeIsRooted()}

Other consensus tree functions: \texttt{Consensus()}

\section*{Examples}

\begin{verbatim}
oldPar <- par(mfrow = c(1, 2), mar = rep(0.5, 4))

# Two trees differing only in placement of tip 2:
trees <- as.phylo(c(0, 53), 6)
plot(trees[[1]])
plot(trees[[2]])

# Strict consensus (left panel) lacks resolution:
plot(ape::consensus(trees))

# But omitting tip two (right panel) reveals shared structure in common:
plot(ConsensusWithout(trees, \texttt{'t2'})
MarkMissing('t2'))

par(oldPar)
\end{verbatim}

\section*{ConstrainedNJ \textit{Constrained neighbour-joining tree}}

\section*{Description}

Constructs an approximation to a neighbour-joining tree, modified in order to be consistent with a constraint. Zero-length branches are collapsed at random.

\section*{Usage}

\texttt{ConstrainedNJ(dataset, constraint, weight = 1L)}

\section*{Arguments}

\begin{itemize}
  \item \texttt{dataset} A phylogenetic data matrix of class \texttt{phyDat}, whose names correspond to the labels of any accompanying tree.
  \item \texttt{constraint} An object of class \texttt{phyDat}; returned trees will be perfectly compatible with each character in constraint. See \texttt{vignette} for further examples.
  \item \texttt{weight} Numeric specifying degree to up-weight characters in constraint.
\end{itemize}

\section*{Value}

\texttt{ConstrainedNJ()} returns a tree of class \texttt{phylo}.

\section*{Author(s)}

Martin R. Smith (martin.smith@durham.ac.uk)
DescendantEdges

See Also

Other tree generation functions: GenerateTree, NJTree(), SingleTaxonTree(), TreeNumber

Examples

dataset <- MatrixToPhyDat(matrix(
    c(0, 1, 1, 0, 0, 1,
     0, 1, 1, 0, 0, 1), ncol = 2,
     dimnames = list(letters[1:6], NULL)))
constraint <- MatrixToPhyDat(
    c(a = 0, b = 0, c = 0, d = 0, e = 1, f = 1))
plot(ConstrainedNJ(dataset, constraint))

DescendantEdges Identify descendant edges

Description

Quickly identify edges that are 'descended' from edges in a tree.

Usage

DescendantEdges(edge = NULL, parent, child, nEdge = length(parent))

AllDescendantEdges(parent, child, nEdge = length(parent))

Arguments

edge Integer specifying the number of the edge whose child edges are required (see edgelabels()).
parent Integer vector corresponding to the first column of the edge matrix of a tree of class phylo, i.e. tree$edge[,1]
child Integer vector corresponding to the second column of the edge matrix of a tree of class phylo, i.e. tree$edge[,2].
nEdge number of edges (calculated from length(parent) if not supplied).

Details

The order of parameters in DescendantEdges() will change in the future, to allow AllDescendantEdges() to be merged into this function (#31). Please explicitly name the edge parameter in DescendantEdges(), and replace AllDesdendantEdges() with DescendantEdges(edge = NULL), to future-proof your code.
DoubleFactorial

Value

DescendantEdges() returns a logical vector stating whether each edge in turn is a descendant of the specified edge (or the edge itself).

AllDescendantEdges() returns a matrix of class logical, with row $N$ specifying whether each edge is a descendant of edge $N$ (or the edge itself).

See Also

Other tree navigation: AncestorEdge(), CladeSizes(), EdgeAncestry(), EdgeDistances(), ListAncestors(), MRCA(), NDescendants(), NodeDepth(), NodeOrder(), NonDuplicateRoot(), RootNode()

---

**DoubleFactorial**

**Double factorial**

Description

Calculate the double factorial of a number, or its logarithm.

Usage

DoubleFactorial(n)

DoubleFactorial64(n)

LnDoubleFactorial(n)

Log2DoubleFactorial(n)

LogDoubleFactorial(n)

LnDoubleFactorial.int(n)

LogDoubleFactorial.int(n)

Arguments

$n$ Vector of integers.

Value

Returns the double factorial, $n \times (n - 2) \times (n - 4) \times (n - 6) \times ...$
doubleFactorials

Functions

- **DoubleFactorial64**: Returns the exact double factorial as a 64-bit integer, for \( n < 34 \).
- **LnDoubleFactorial**: Returns the logarithm of the double factorial.
- **Log2DoubleFactorial**: Returns the logarithm of the double factorial.
- **LnDoubleFactorial.int**: Slightly faster, when \( x \) is known to be length one and below 50001

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other double factorials: **doubleFactorials**, **logDoubleFactorials**

Examples

```r
DoubleFactorial (-4:0) # Return 1 if n < 2
DoubleFactorial (2) # 2
DoubleFactorial (5) # 1 * 3 * 5
exp(LnDoubleFactorial.int (8)) # log(2 * 4 * 6 * 8)
DoubleFactorial64(31)
```

---

**doubleFactorials**  
**Double factorials**

Description

A vector with pre-calculated values of double factorials up to 300!!, and the logarithms of double factorials up to 50 000!!.

Usage

doubleFactorials

Format

An object of class numeric of length 300.

Details

301!! is too large to store as an integer; use logDoubleFactorials instead.

See Also

Other double factorials: **DoubleFactorial()**, **logDoubleFactorials**
DropTip

*Drop tips from tree*

**Description**

DropTip() removes specified tips from a phylogenetic tree, collapsing incident branches.

**Usage**

DropTip(tree, tip, preorder = TRUE)

```r
## S3 method for class 'phylo'
DropTip(tree, tip, preorder = TRUE)
```

```r
## S3 method for class 'multiPhylo'
DropTip(tree, tip, preorder = TRUE)
```

KeepTip(tree, tip, preorder = TRUE)

**Arguments**

- `tree` A tree of class phylo.
- `tip` Character vector specifying labels of leaves in tree to be dropped, or integer vector specifying the indices of leaves to be dropped. Specifying the index of an internal node will drop all descendants of that node.
- `preorder` Logical specifying whether to Preorder the tree before dropping tips. Necessary if a tree’s edges may be unconventionally numbered.

**Details**

This function is more robust than ape::drop.tip() as it does not require any particular internal node numbering schema.

**Value**

DropTip() returns a tree of class phylo, with the requested leaves removed.

KeepTip() returns tree with all leaves not in `tip` removed, in preorder.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Other tree manipulation: AddTip(), CollapseNode(), ConsensusWithout(), EnforceOutgroup(), ImposeConstraint(), LeafLabelInterchange(), MakeTreeBinary(), RenumberTips(), RenumberTree(), Renumber(), RootTree(), SingleTaxonTree(), SortTree(), Subtree()
Examples

```r
tree <- BalancedTree(8)
plot(tree)
plot(DropTip(tree, c('t4', 't5')))
```

---

**EdgeAncestry**  
*Ancestors of an edge*

**Description**

Quickly identify edges that are 'ancestral' to a particular edge in a tree.

**Usage**

```r
EdgeAncestry(edge, parent, child, stopAt = (parent == min(parent)))
```

**Arguments**

- `edge`: Integer specifying the number of the edge whose child edges should be returned.
- `parent`: Integer vector corresponding to the first column of the edge matrix of a tree of class `phylo`, i.e. `tree$edge[,1]`
- `child`: Integer vector corresponding to the second column of the edge matrix of a tree of class `phylo`, i.e. `tree$edge[,2]`.
- `stopAt`: Integer or logical vector specifying the edge(s) at which to terminate the search; defaults to the edges with the smallest parent, which will be the root edges if nodes are numbered **Cladewise** or in **Preorder**.

**Value**

`EdgeAncestry()` returns a logical vector stating whether each edge in turn is a descendant of the specified edge.

**Author(s)**

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

**See Also**

Other tree navigation: `AncestorEdge()`, `CladeSizes()`, `DescendantEdges()`, `EdgeDistances()`, `ListAncestors()`, `MRCA()`, `NDescendants()`, `NodeDepth()`, `NodeOrder()`, `NonDuplicateRoot()`, `RootNode()`
### Examples

```r
tree <- PectinateTree(6)
plot(tree)
ape::edgelabels()
parent <- tree$edge[, 1]
child <- tree$edge[, 2]
EdgeAncestry(7, parent, child)
which(EdgeAncestry(7, parent, child, stopAt = 4))
```

---

### Description

Number of nodes that must be traversed to navigate from each edge to each other edge within a tree.

### Usage

```r
EdgeDistances(tree)
```

### Arguments

- **tree**: A tree of class `phylo`.

### Value

`EdgeDistances()` returns a symmetrical matrix listing the number of edges that must be traversed to travel from each numbered edge to each other. The two edges straddling the root of a rooted tree are treated as a single edge. Add a 'root' tip using `AddTip()` if the position of the root is significant.

### Author(s)

Martin R. Smith ([martin.smith@durham.ac.uk](mailto:martin.smith@durham.ac.uk))

### See Also

Other tree navigation: `AncestorEdge()`, `CladeSizes()`, `DescendantEdges()`, `EdgeAncestry()`, `ListAncestors()`, `MRCA()`, `NDescendants()`, `NodeDepth()`, `NodeOrder()`, `NonDuplicateRoot()`, `RootNode()`

### Examples

```r
tree <- BalancedTree(5)
plot(tree)
ape::edgelabels()

EdgeDistances(tree)
```
**EndSentence**

*Add full stop to end of a sentence*

**Description**

Add full stop to end of a sentence

**Usage**

EndSentence(string)

**Arguments**

- string: Input string

**Value**

EndSentence() returns string, punctuated with a final full stop (period).

**Author(s)**

Martin R. Smith

**See Also**

Other string parsing functions: MorphoBankDecode(), RightmostCharacter(), Unquote()

**Examples**

EndSentence("Hello World") # "Hello World."

---

**EnforceOutgroup**

*Generate a tree with a specific outgroup*

**Description**

Given a tree or a list of taxa, EnforceOutgroup() rearranges the ingroup and outgroup taxa such that the two are sister taxa across the root, without changing the relationships within the ingroup or within the outgroup.

**Usage**

EnforceOutgroup(tree, outgroup)

```r
## S3 method for class 'phylo'
EnforceOutgroup(tree, outgroup)

## S3 method for class 'character'
EnforceOutgroup(tree, outgroup)
```
GenerateTree

Arguments

tree     Either a tree of class phylo; or (for EnforceOutgroup()) a character vector listing the names of all the taxa in the tree, from which a random tree will be generated.

outgroup Character vector containing the names of taxa to include in the outgroup.

Value

EnforceOutgroup() returns a tree of class phylo where all outgroup taxa are sister to all remaining taxa, without modifying the ingroup topology.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

For a more robust implementation, see RootTree(), which will eventually replace this function (#30).

Other tree manipulation: AddTip(), CollapseNode(), ConsensusWithout(), DropTip(), ImposeConstraint(), LeafLabelInterchange(), MakeTreeBinary(), RenumberTips(), RenumberTree(), Renumber(), RootTree(), SingleTaxonTree(), SortTree(), Subtree()

Examples

tree <- EnforceOutgroup(letters[1:9], letters[1:3])
plot(tree)

GenerateTree

Generate pectinate, balanced or random trees

Description

RandomTree(), PectinateTree(), BalancedTree() and StarTree() generate trees with the specified shapes and leaf labels.

Usage

RandomTree(tips, root = FALSE)

PectinateTree(tips)

BalancedTree(tips)

StarTree(tips)
GenerateTree

Arguments

tips  An integer specifying the number of tips, or a character vector naming the tips, or any other object from which TipLabels() can extract leaf labels.
root  Character or integer specifying tip to use as root, if desired; or FALSE for an unrooted tree.

Value

Each function returns an unweighted binary tree of class phylo with the specified leaf labels. Trees are rooted unless root = FALSE.

RandomTree() returns a topology drawn at random from the uniform distribution (i.e. each binary tree is drawn with equal probability). Trees are generated by inserting each tip in turn at a randomly selected edge in the tree. Random numbers are generated using a Mersenne Twister. If root = FALSE, the tree will be unrooted, with the first tip in a basal position. Otherwise, the tree will be rooted on root.

PectinateTree() returns a pectinate (caterpillar) tree.

BalancedTree() returns a balanced (symmetrical) tree.

StarTree() returns a completely unresolved (star) tree.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other tree generation functions: ConstrainedNJ(), NJTree(), SingleTaxonTree(), TreeNumber

Examples

RandomTree(LETTERS[1:10])

data('Lobo')
RandomTree(Lobo.phy)

plot(PectinateTree(LETTERS[1:10]))

plot(BalancedTree(LETTERS[1:10]))
plot(StarTree(LETTERS[1:10]))
ImposeConstraint

**Force a tree to match a constraint**

**Description**

Modify a tree such that it matches a specified constraint. This is at present a somewhat crude implementation that attempts to retain much of the structure of tree whilst guaranteeing compatibility with each entry in constraint.

**Usage**

```
ImposeConstraint(tree, constraint)
```

```
AddUnconstrained(constraint, toAdd, asPhyDat = TRUE)
```

**Arguments**

- `tree`: A tree of class `phylo`.
- `constraint`: An object of class `phyDat`; returned trees will be perfectly compatible with each character in `constraint`. See vignette for further examples.
- `toAdd`: Character vector specifying taxa to add to constraint.
- `asPhyDat`: Logical: if TRUE, return a phyDat object; if FALSE, return a matrix.

**Value**

`ImposeConstraint()` returns a tree of class phylo, consistent with `constraint`.

**Functions**

- `AddUnconstrained`: Expand a constraint to include unconstrained taxa.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Other tree manipulation: `AddTip()`, `CollapseNode()`, `ConsensusWithout()`, `DropTip()`, `EnforceOutgroup()`, `LeafLabelInterchange()`, `MakeTreeBinary()`, `RenumberTips()`, `RenumberTree()`, `Renumber()`, `RootTree()`, `SingleTaxonTree()`, `SortTree()`, `Subtree()`
**Examples**

```r
tips <- letters[1:9]
tree <- as.phylo(1, 9, tips)
plot(tree)

constraint <- StringToPhyDat("0000?1111 00011111 0000??110", tips, FALSE)
plot(ImposeConstraint(tree, constraint))
```

**LabelSplits**

 LABEL splits

**Description**

Labels the edges associated with each split on a plotted tree.

**Usage**

`LabelSplits(tree, labels = NULL, unit = "", ...)`

**Arguments**

- `tree`: A tree of class `phylo`.
- `labels`: Named vector listing annotations for each split. Names should correspond to the node associated with each split; see `as.Splits()` for details. If `NULL`, each splits will be labelled with its associated node.
- `unit`: Character specifying units of `labels`, if desired. Include a leading space if necessary.
- `...`: Additional parameters to `ape::edgelabels()`.

**Details**

As the two root edges of a rooted tree denote the same split, only the rightmost (plotted at the bottom, by default) edge will be labelled. If the position of the root is significant, add a tip at the root using `AddTip()`.

**Value**

`LabelSplits()` returns `invisible()`, after plotting labels on each relevant edge of a plot (which should already have been produced using `plot(tree)`).

**See Also**

- Calculate split support: `SplitFrequency()`
- Colour labels according to value: `SupportColour()`
- Other Splits operations: `NSplits()`, `NTip()`, `PolarizeSplits()`, `SplitFrequency()`, `SplitsInBinaryTree()`, `Splits`, `TipLabels()`, `TipsInSplits()`, `match()`
Examples

tree <- BalancedTree(LETTERS[1:5])
splits <- as.Splits(tree)
plot(tree)
LabelSplits(tree, as.character(splits), frame = 'none', pos = 3L)
LabelSplits(tree, TipsInSplits(splits), unit = 'tips', frame = 'none',
            pos = 1L)

# An example forest of 100 trees, some identical
forest <- as.phylo(c(1, rep(10, 79), rep(100, 15), rep(1000, 5)), nTip = 9)

# Generate an 80% consensus tree
cons <- ape::consensus(forest, p = 0.8)
plot(cons)

splitFreqs <- SplitFrequency(cons, forest)
LabelSplits(cons, splitFreqs, unit = '%',
            col = SupportColor(splitFreqs / 100),
            frame = 'none', pos = 3L)

LeafLabelInterchange  Leaf label interchange

Description

LeafLabelInterchange() exchanges the position of leaves within a tree.

Usage

LeafLabelInterchange(tree, n = 2L)

Arguments

tree A tree of class phylo.
n Integer specifying number of leaves whose positions should be exchanged.

Details

Modifies a tree by switching the positions of $n$ leaves. To avoid later swaps undoing earlier exchanges, all $n$ leaves are guaranteed to change position. Note, however, that no attempt is made to avoid swapping equivalent leaves, for example, a pair that are each others' closest relatives. As such, the relationships within a tree are not guaranteed to be changed.

Value

LeafLabelInterchange() returns a tree of class phylo on which the position of $n$ leaves have been exchanged. The tree’s internal topology will not change.
ListAncestors

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other tree manipulation: AddTip(), CollapseNode(), ConsensusWithout(), DropTip(), EnforceOutgroup(), ImposeConstraint(), MakeTreeBinary(), RenumberTips(), RenumberTree(), Renumber(), RootTree(), SingleTaxonTree(), SortTree(), Subtree()

Examples

tree <- PectinateTree(8)
plot(LeafLabelInterchange(tree, 3L))

<table>
<thead>
<tr>
<th>ListAncestors</th>
<th>List ancestors</th>
</tr>
</thead>
</table>

Description

ListAncestors() reports all ancestors of a given node.

Usage

ListAncestors(parent, child, node = NULL)
AllAncestors(parent, child)

Arguments

parent  Integer vector corresponding to the first column of the edge matrix of a tree of class phylo, i.e. tree$edge[,1]
child   Integer vector corresponding to the second column of the edge matrix of a tree of class phylo, i.e. tree$edge[,2].
node    Integer giving the index of the node or tip whose ancestors are required, or NULL to return ancestors of all nodes.

Details

Note that if node = NULL, the tree’s edges must be listed such that each internal node (except the root) is listed as a child before it is listed as a parent, i.e. its index in child is less than its index in parent. This will be true of trees listed in Preorder.
ListAncestors

Value

If node = NULL, ListAncestors() returns a list. Each entry i contains a vector containing, in order, the nodes encountered when traversing the tree from node i to the root node. The last entry of each member of the list is therefore the root node, with the exception of the entry for the root node itself, which is a zero-length integer.

If node is an integer, ListAncestors() returns a vector of the numbers of the nodes ancestral to the given node, including the root node.

Functions

- **AllAncestors**: Alias for ListAncestors(node = NULL).

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Implemented less efficiently in phangorn:::Ancestors, on which this code is based.

Other tree navigation: AncestorEdge(), CladeSizes(), DescendantEdges(), EdgeAncestry(), EdgeDistances(), MRCA(), NDescendants(), NodeDepth(), NodeOrder(), NonDuplicateRoot(), RootNode()

Other tree navigation: AncestorEdge(), CladeSizes(), DescendantEdges(), EdgeAncestry(), EdgeDistances(), MRCA(), NDescendants(), NodeDepth(), NodeOrder(), NonDuplicateRoot(), RootNode()

Examples

tree <- PectinateTree(5)
edge <- tree$edge

# Identify desired node with:
plot(tree)
nodelabels()
tiplabels()

# Ancestors of specific nodes:
ListAncestors(edge[, 1], edge[, 2], 4L)
ListAncestors(edge[, 1], edge[, 2], 8L)

# Ancestors of each node, if tree numbering system is uncertain:
lapply(seq_len(max(edge)), ListAncestors,
parent = edge[, 1], child = edge[, 2])

# Ancestors of each node, if tree is in preorder:
ListAncestors(edge[, 1], edge[, 2])

# Alias:
Lobo.data

Description

Phylogenetic data from Zhang et al. (2016) in raw (Lobo.data) and phyDat (Lobo.phy) formats.

Usage

Lobo.data

Lobo.phy

Format

An object of class list of length 48.
An object of class phyDat of length 48.

Source


Examples

data("Lobo", package = "TreeTools")
Lobo.data
Lobo.phy

logDoubleFactorials

Description

logDoubleFactorials is a numeric vector with pre-calculated values of double factorials up to 50 000!!.

Usage

logDoubleFactorials

Format

An object of class numeric of length 50000.
MakeTreeBinary

Generate binary tree by collapsing polytomies

Description

MakeTreeBinary() resolves, at random, all polytomies in a tree or set of trees, such that all trees compatible with the input topology are drawn with equal probability.

Usage

MakeTreeBinary(tree)

Arguments

tree A tree of class phylo.

Value

MakeTreeBinary() returns a rooted binary tree of class phylo, corresponding to tree uniformly selected from all those compatible with the input tree topologies.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Since ape v5.5, this functionality is available through ape::multi2di(); previous versions of 'ape' did not return topologies in equal frequencies.

Other tree manipulation: AddTip(), CollapseNode(), ConsensusWithout(), DropTip(), EnforceOutgroup(), ImposeConstraint(), LeafLabelInterchange(), RenumberTips(), RenumberTree(), Renumber(), RootTree(), SingleTaxonTree(), SortTree(), Subtree()

Examples

MakeTreeBinary(CollapseNode(PectinateTree(7), c(9, 11, 13)))
UnrootTree(MakeTreeBinary(StarTree(5)))
**match**  

### Split matching

**Description**

`match()` returns a vector of the positions of (first) matches of splits in its first argument in its second. `%in%` is a more intuitive interface as a binary operator, which returns a logical vector indicating whether there is a match or not for each split in its left operand.

**Usage**

```r
match(x, table, ...)  
## S3 method for class 'Splits'
match(x, table, ...)  
## S3 method for class 'list'
match(x, table, ...)
```

```r
x %in% table  
## S3 method for class 'Splits'
x %in% table
```

```r
in.Splits(x, table)
```

**Arguments**

- `x`, `table` Object of class `Splits`.
- `...` Specify `nomatch =` to provide an integer value that will be used in place of `NA` in the case where no match is found.

**Details**

`in.Splits()` is an alias for `%in%`, included for backwards compatibility. It will be deprecated in a future release.

**Value**

`match()` returns an integer vector specifying the position in `table` that matches each element in `x`, or `nomatch` if no match is found.  

`%in%` returns a logical vector specifying which of the splits in `x` are present in `table`.

**See Also**

Corresponding base functions are documented in `match()`.  
Other Splits operations: `LabelSplits()`, `NSplits()`, `NTip()`, `PolarizeSplits()`, `SplitFrequency()`, `SplitsInBinaryTree()`, `Splits`, `TipLabels()`, `TipsInSplits()`
Examples

```r
splits1 <- as.Splits(BalancedTree(7))
splits2 <- as.Splits(PectinateTree(7))

match(splits1, splits2)
splits1 %in% splits2
```

---

MorphoBankDecode

**Decode MorphoBank text**

**Description**

Converts strings from MorphoBank notes into a Latex-compatible format.

**Usage**

```r
MorphoBankDecode(string)
```

**Arguments**

- `string` String to process

**Value**

`MorphoBankDecode()` returns a string with new lines and punctuation reformatted.

**Author(s)**

Martin R. Smith

**See Also**

Other string parsing functions: `EndSentence()`, `RightmostCharacter()`, `Unquote()`
**MRCA**

*Most recent common ancestor*

**Description**

MRCA() calculates the last common ancestor of specified nodes.

**Usage**

```r
MRCA(x1, x2, ancestors)
```

**Arguments**

- `x1, x2`: Integer specifying index of leaves or nodes whose most recent common ancestor should be found.
- `ancestors`: List of ancestors for each node in a tree. Perhaps produced by `ListAncestors()`.

**Details**

MRCA() requires that node values within a tree increase away from the root, which will be true of trees listed in `Preorder`. No warnings will be given if trees do not fulfil this requirement.

**Value**

MRCA() returns an integer specifying the node number of the last common ancestor of `x1` and `x2`.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Other tree navigation: `AncestorEdge()`, `CladeSizes()`, `DescendantEdges()`, `EdgeAncestry()`, `EdgeDistances()`, `ListAncestors()`, `NDescendants()`, `NodeDepth()`, `NodeOrder()`, `NonDuplicateRoot()`, `RootNode()`

**Examples**

```r
tree <- BalancedTree(7)

# Verify that node numbering increases away from root
plot(tree)
nodelabels()

# ListAncestors expects a tree in Preorder
tree <- Preorder(tree)
edge <- tree$edge
ancestors <- ListAncestors(edge[, 1], edge[, 2])
MRCA(1, 4, ancestors)
```
# If a tree must be in postorder, use:
```
tree <- Postorder(tree)
edge <- tree$edge
ancestors <- lapply(seq_len(max(edge)), ListAncestors,
    parent = edge[, 1], child = edge[, 2])
```

---

**MSTEdges**  
*Minimum spanning tree*

**Description**

Calculate or plot the minimum spanning tree of a distance matrix.

**Usage**

```
MSTEdges(distances, plot = FALSE, x = NULL, y = NULL, ...)

MSTLength(distances, mst = NULL)
```

**Arguments**

- `distances` Either a matrix that can be interpreted as a distance matrix, or an object of class `dist`.
- `plot` Logical specifying whether to add the minimum spanning tree to an existing plot.
- `x, y` Numeric vectors specifying the X and Y coordinates of each element in `distances`. Necessary only if `plot = TRUE`.
- `...` Additional parameters to send to `lines()`.
- `mst` Optional parameter specifying the minimum spanning tree in the format returned by `MSTEdges()`; if `NULL`, calculated from `distances`.

**Value**

`MSTEdges()` returns a matrix in which each row corresponds to an edge of the minimum spanning tree, listed in non-decreasing order of length. The two columns contain the indices of the entries in `distances` that each edge connects, with the lower value listed first.

`MSTLength()` returns the length of the minimum spanning tree.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**References**

See Also

Slow implementation returning the association matrix of the minimum spanning tree: `ape::mst()`.

Examples

# Corners of an almost-regular octahedron
points <- matrix(c(0, 0, 2, 2, 1.1, 1,
                   0, 2, 0, 2, 1, 1.1,
                   0, 0, 0, 0, 1, -1), 6)
distances <- dist(points)
mst <- MSTEdges(distances)
MSTLength(distances, mst)
plot(points[, 1:2], ann = FALSE, asp = 1)
MSTEdges(distances, TRUE, x = points[, 1], y = points[, 2], lwd = 2)

N1Spr

Number of trees one SPR step away

Description

`N1Spr()` calculates the number of trees one subtree prune-and-regraft operation away from a binary input tree using the formula given by Allen and Steel (2001); `IC1Spr()` calculates the information content of trees at this distance: i.e. the entropy corresponding to the proportion of all possible n-tip trees whose SPR distance is at most one from a specified tree.

Usage

N1Spr(n)

IC1Spr(n)

Arguments

n

Integer vector specifying the number of tips in a tree.

Value

N1Spr() returns an integer vector denoting the number of trees one SPR rearrangement away from the input tree.  
IC1Spr() returns a numeric vector giving the phylogenetic information content of trees 0 or 1 SPR rearrangement from an n-leaf tree, in bits.

References

NDescendants

**Examples**

```r
N1Spr(4:6)
IC1Spr(5)
```

**Description**

NDescendants() counts the number of nodes (including leaves) directly descended from each node in a tree.

**Usage**

```r
NDescendants(tree)
```

**Arguments**

- `tree` A tree of class `phylo`.

**Value**

NDescendants() returns an integer listing the number of direct descendants (leaves or internal nodes) for each node in a tree.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Other tree navigation: `AncestorEdge()`, `CladeSizes()`, `DescendantEdges()`, `EdgeAncestry()`, `EdgeDistances()`, `ListAncestors()`, `MRCA()`, `NodeDepth()`, `NodeOrder()`, `NonDuplicateRoot()`, `RootNode()`

**Examples**

```r
tree <- CollapseNode(BalancedTree(8), 12:15)
NDescendants(tree)
plot(tree)
nodelabels(NDescendants(tree))
```
**NewickTree**

```r
NewickTree() encodes a tree as a Newick-format string. This differs from `write.tree()` in the encoding of spaces as spaces, rather than underscores.
```

**Usage**

```r
NewickTree(tree)
```

**Arguments**

- `tree`: A tree of class `phylo`.

**Value**

`NewickTree()` returns a character string denoting `tree` in Newick format.

**See Also**

Use tip numbers, rather than leaf labels: `as.Newick`

**Examples**

```r
NewickTree(BalancedTree(LETTERS[4:9]))
```

---

**NJTree**

```r
Generate a neighbour joining tree
```

**Description**

`NJTree()` generates a rooted neighbour joining tree from a phylogenetic dataset.

**Usage**

```r
NJTree(dataset, edgeLengths = FALSE)
```

**Arguments**

- `dataset`: A phylogenetic data matrix of class `phyDat`, whose names correspond to the labels of any accompanying tree.
- `edgeLengths`: Logical specifying whether to include edge lengths.
**NodeDepth**

**Value**

NJTree returns an object of class `phylo`.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Other tree generation functions: `ConstrainedNJ()`, `GenerateTree`, `SingleTaxonTree()`, `TreeNumber`

**Examples**

data('Lobo')
NJTree(Lobo.phy)

<table>
<thead>
<tr>
<th>NodeDepth</th>
<th>Distance of each node from tree exterior</th>
</tr>
</thead>
</table>

**Description**

NodeDepth() evaluates how 'deep' each node is within a tree.

**Usage**

NodeDepth(x, shortest = FALSE, includeTips = TRUE)

**Arguments**

- **x**: A tree of class `phylo`, its `edge` property, or a list thereof.
- **shortest**: Logical specifying whether to calculate the length of the shortest away-from-root path to a leaf. If FALSE, the length of the longest such route will be returned.
- **includeTips**: Logical specifying whether to include leaves (each of depth zero) in return value.

**Details**

For a rooted tree, the depth of a node is the minimum (if `shortest` = TRUE) or maximum (shortest = FALSE) number of edges that must be traversed, moving away from the root, to reach a leaf.

Unrooted trees are treated as if a root node occurs in the 'middle' of the tree, meaning the position that will minimise the maximum node depth.

**Value**

NodeDepth() returns an integer vector specifying the depth of each external and internal node in x.
NodeOrder

Author(s)
Martin R. Smith (martin.smith@durham.ac.uk)

See Also
ape::node.depth returns the number of tips descended from a node.

Other tree navigation: AncestorEdge(), CladeSizes(), DescendantEdges(), EdgeAncestry(), EdgeDistances(), ListAncestors(), MRCA(), NDescendants(), NodeOrder(), NonDuplicateRoot(), RootNode()

Examples

tree <- CollapseNode(BalancedTree(10), c(12:13, 19))
plot(tree)
nodelabels(NodeDepth(tree, includeTips = FALSE))

Description
NodeOrder() calculates the number of edges incident to each node in a tree. Includes the root edge in rooted trees.

Usage
NodeOrder(x, includeAncestor = TRUE, internalOnly = FALSE)

Arguments
x A tree of class phylo, its $edge property, or a list thereof.
includeAncestor Logical specifying whether to count edge leading to ancestral node in calculation of order.
internalOnly Logical specifying whether to restrict to results to internal nodes, i.e. to omit leaves. Irrelevant if includeAncestor = FALSE.

Value
NodeOrder() returns an integer listing the order of each node; entries are named with the number of each node.

Author(s)
Martin R. Smith (martin.smith@durham.ac.uk)
NPartitionPairs

See Also

Other tree navigation: AncestorEdge(), CladeSizes(), DescendantEdges(), EdgeAncestry(), EdgeDistances(), ListAncestors(), MRCA(), NDescendants(), NodeDepth(), NonDuplicateRoot(), RootNode()

Examples

tree <- CollapseNode(BalancedTree(8), 12:15)
NodeOrder(tree)
plot(tree)
nodelabels(NodeOrder(tree, internalOnly = TRUE))

NPartitionPairs

Distributions of tips consistent with a partition pair

Description

NPartitionPairs() calculates the number of terminal arrangements matching a specified configuration of two splits.

Usage

NPartitionPairs(configuration)

Arguments

configuration Integer vector of length four specifying the number of terminals that occur in both (1) splits A1 and A2; (2) splits A1 and B2; (3) splits B1 and A2; (4) splits B1 and B2.

Details

Consider splits that divide eight terminals, labelled A to H.

Bipartition 1: ABCD:EFGH A1 = ABCD B1 = EFGH
Bipartition 2: ABE:CDFGH A2 = ABE B2 = CDFGH

This can be represented by an association matrix:

\[
\begin{array}{ccc}
A2 & B2 \\
A1 & AB & C \\
B1 & E & FGH \\
\end{array}
\]

The cells in this matrix contain 2, 1, 1 and 3 terminals respectively; this four-element vector (c(2,1,1,3)) is the configuration implied by this pair of bipartition splits.
Value
The number of ways to distribute \texttt{sum(configuration)} taxa according to the specified pattern.

Author(s)
Martin R. Smith (martin.smith@durham.ac.uk)

Examples
\begin{verbatim}
NPartitionPairs(c(2, 1, 1, 3))
\end{verbatim}

\begin{tabular}{lc}
\hline
\texttt{NRoloted} & \textit{Number of trees} \\
\hline
\end{tabular}

Description
These functions return the number of rooted or unrooted binary trees consistent with a given pattern of splits.

Usage
\begin{verbatim}
NRoloted(tips) \\
NUrrooted(tips) \\
NRoloted64(tips) \\
NUrrooted64(tips) \\
LnUrrooted(tips) \\
LnUrrooted.int(tips) \\
Log2Urrooted(tips) \\
Log2Urrooted.int(tips) \\
LnRrooted(tips) \\
LnRrooted.int(tips) \\
Log2Rrooted(tips) \\
Log2Rrooted.int(tips) \\
LnUrrootedSplits(...) \\
\end{verbatim}
\[
\text{Log2UnrootedSplits}(\ldots)
\]
\[
\text{NUnrootedSplits}(\ldots)
\]
\[
\text{LnUnrootedMult}(\ldots)
\]
\[
\text{Log2UnrootedMult}(\ldots)
\]
\[
\text{NUnrootedMult}(\ldots)
\]

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tips</td>
<td>Integer specifying the number of leaves.</td>
</tr>
<tr>
<td>...</td>
<td>Integer vector, or series of integers, listing the number of leaves in each split.</td>
</tr>
</tbody>
</table>

**Details**

Functions starting `N` return the number of rooted or unrooted trees. Replace this initial `N` with `Ln` for the natural logarithm of this number; or `Log2` for its base 2 logarithm.


**Functions**

- `NUnrooted`: Number of unrooted trees
- `NRooted64`: Exact number of rooted trees as 64-bit integer (`13 < nTip < 19`)
- `NUnrooted64`: Exact number of unrooted trees as 64-bit integer (`14 < nTip < 20`)
- `LnUnrooted`: Log Number of unrooted trees
- `LnUnrooted.int`: Log Number of unrooted trees (as integer)
- `LnRooted`: Log Number of rooted trees
- `LnRooted.int`: Log Number of rooted trees (as integer)
- `NUnrootedSplits`: Number of unrooted trees consistent with a bipartition split.
- `NUnrootedMult`: Number of unrooted trees consistent with a multi-partition split.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**References**


**See Also**

Other tree information functions: `CladisticInfo()`, `TreesMatchingTree()`
**nRootedShapes**

**Examples**

```r
NRooted(10)
NUnrooted(10)
LnRooted(10)
LnUnrooted(10)
Log2Unrooted(10)
# Number of trees consistent with a character whose states are
# 00000 11111 222
NUnrootedMult(c(5,5,3))

NUnrooted64(18)
LnUnrootedSplits(c(2,4))
LnUnrootedSplits(3, 3)
Log2UnrootedSplits(c(2,4))
Log2UnrootedSplits(3, 3)
NUnrootedSplits(c(2,4))
NUnrootedSplits(3, 3)
```

---

<table>
<thead>
<tr>
<th>nRootedShapes</th>
<th>Number of rooted / unrooted tree shapes</th>
</tr>
</thead>
<tbody>
<tr>
<td>nRootedShapes</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

nRootedShapes and nUnrootedShapes give the number of (un)rooted binary trees on \( n \) unlabelled leaves.

**Usage**

nRootedShapes

nUnrootedShapes

**Format**

An object of class integer64 of length 55.

An object of class integer64 of length 60.

**Source**

nRootedShapes corresponds to the Wedderburn-Etherington numbers, [OEIS A001190](https://oeis.org/A001190)
nUnrootedShapes is [OEIS A000672](https://oeis.org/A000672)
NSplits

Number of distinct splits

Description

NSplits() counts the unique bipartition splits in a tree or object.

Usage

NSplits(x)

NPartitions(x)

## S3 method for class 'phylo'
NSplits(x)

## S3 method for class 'list'
NSplits(x)

## S3 method for class 'multiPhylo'
NSplits(x)

## S3 method for class 'Splits'
NSplits(x)

## S3 method for class 'numeric'
NSplits(x)

## S3 method for class '\NULL\'
NSplits(x)

## S3 method for class 'ClusterTable'
NSplits(x)

## S3 method for class 'character'
NSplits(x)

Arguments

x  A phylogenetic tree of class phylo; a list of such trees (of class list or multiPhylo); a Splits object; a vector of integers; or a character vector listing tips of a tree, or a character of length one specifying a tree in Newick format.

Value

NSplits() returns an integer specifying the number of bipartitions in the specified objects, or in a binary tree with x tips.
**NTip**

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Other tree properties: ConsensusWithout(), NTip(), SplitsInBinaryTree(), TipLabels(), TreeIsRooted()

Other Splits operations: LabelSplits(), NTip(), PolarizeSplits(), SplitFrequency(), SplitsInBinaryTree(), Splits.TipLabels(), TipsInSplits(), match()

**Examples**

NSplits(8L)
NSplits(PectinateTree(8))
NSplits(as.Splits(BalancedTree(8)))

---

**Description**

NTip() extends ape::Ntip() to handle objects of class Splits and list, and edge matrices (equivalent to tree$edge).

**Usage**

NTip(phy)

## Default S3 method:
NTip(phy)

## S3 method for class 'Splits'
NTip(phy)

## S3 method for class 'list'
NTip(phy)

## S3 method for class 'phylo'
NTip(phy)

## S3 method for class 'multiPhylo'
NTip(phy)

## S3 method for class 'phyDat'
NTip(phy)

## S3 method for class 'matrix'
NTip(phy)
**PairwiseDistances**

**Arguments**

- **phy**
  
  Object representing one or more phylogenetic trees.

**Value**

- **NTip()** returns an integer specifying the number of tips in each object in `phy`.

**See Also**

- Other tree properties: `ConsensusWithout()`, `NSplits()`, `SplitsInBinaryTree()`, `TipLabels()`, `TreeIsRooted()`.
- Other Splits operations: `LabelSplits()`, `NSplits()`, `PolarizeSplits()`, `SplitFrequency()`, `SplitsInBinaryTree()`, `Splits`, `TipLabels()`, `TipsInSplits()`, `match()`.

---

**PairwiseDistances**  
*Distances between each pair of trees*

**Description**

Distances between each pair of trees

**Usage**

```r
PairwiseDistances(trees, Func, valueLength = 1L, ...)
```

**Arguments**

- **trees**
  
  List of trees of class phylo.

- **Func**
  
  Function returning a distance between two trees.

- **valueLength**
  
  Integer specifying expected length of the value returned by `Func`.

- **...**
  
  Additional arguments to `Func`.

**Value**

Matrix detailing distance between each pair of trees. Identical trees are assumed to have zero distance.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)
Examples

    trees <- list(BalancedTree(8), PectinateTree(8), StarTree(8))
    TCIDiff <- function (tree1, tree2) {
        TotalCopheneticIndex(tree1) - TotalCopheneticIndex(tree2)
    }
    PairwiseDistances(trees, TCIDiff, 1)
    TCIRange <- function (tree1, tree2) {
        range(TotalCopheneticIndex(tree1), TotalCopheneticIndex(tree2))
    }
    PairwiseDistances(trees, TCIRange, 2)

PolarizeSplits

Polarize splits on a single taxon

Description

Polarize splits on a single taxon

Usage

    PolarizeSplits(x, pole = 1L)

Arguments

    x Object of class Splits.
    pole Numeric or character identifying tip that should polarize each split.

Value

    PolarizeSplits() returns a Splits object in which pole is represented by a zero bit

See Also

    Other Splits operations: LabelSplits(), NSplits(), NTip(), SplitFrequency(), SplitsInBinaryTree(), Splits, TipLabels(), TipsInSplits(), match()
print.TreeNumber  \hspace{1em} \textit{Print TreeNumber object}

\textbf{Description}

S3 method for objects of class \texttt{TreeNumber}.

\textbf{Usage}

```r
## S3 method for class 'TreeNumber'
print(x, ...)
```

\textbf{Arguments}

\begin{itemize}
  \item \texttt{x} \hspace{1em} Object of class \texttt{TreeNumber}.
  \item \texttt{...} \hspace{1em} Additional arguments for consistency with S3 method (unused).
\end{itemize}

\textbf{ReadCharacters}  \hspace{1em} \textit{Read phylogenetic characters from file}

\textbf{Description}

Parse a Nexus or TNT file, reading character states and names.

\textbf{Usage}

```r
ReadCharacters(
  filepath,
  character_num = NULL,
  encoding = "UTF8",
  session = NULL
)
```

```r
ReadTntCharacters(
  filepath,
  character_num = NULL,
  type = NULL,
  session = NULL,
  encoding = "UTF8"
)
```

```r
ReadNotes(filepath, encoding = "UTF8")
```

```r
ReadAsPhyDat(...)```


ReadTntAsPhyDat(...)

PhyDat(dataset)

Arguments

filepath character string specifying location of file, or a connection to the file.
character_num Index of character(s) to return. NULL, the default, returns all characters.
encoding Character encoding of input file.
session (Optional) A Shiny session with a numericInput named character_num whose maximum should be updated.
type Character vector specifying categories of data to extract from file. Setting type = c('num', 'dna') will return only characters following a &[num] or &[dna] tag in a TNT input file, listing num character blocks before dna characters. Leave as NULL (the default) to return all characters in their original sequence.
... Parameters to pass to Read[Tnt]Characters().
dataset list of taxa and characters, in the format produced by read.nexus.data: a list of sequences each made of a single character vector, and named with the taxon name.

Details

Tested with matrices downloaded from MorphoBank, but should also work more widely; please report incompletely or incorrectly parsed files.

Matrices must contain only continuous or only discrete characters; maximum one matrix per file. Continuous characters will be read as strings (i.e. base type 'character').

The encoding of an input file will be automatically determined by R. Errors pertaining to an invalid multibyte string or string invalid at that locale indicate that R has failed to detect the appropriate encoding. Either re-save the file in a supported encoding (UTF-8 is a good choice) or specify the file encoding (which you can find by, for example, opening in Notepad++ and identifying the highlighted option in the "Encoding" menu) following the example below.

Value

ReadCharacters() and ReadTNTCharacters() return a matrix whose row names correspond to tip labels, and column names correspond to character labels, with the attribute state.labels listing the state labels for each character; or a list of length one containing a character string explaining why the function call was unsuccessful.

ReadAsPhyDat() and ReadTntAsPhyDat() return a phyDat object.

ReadNotes() returns a list in which each entry corresponds to a single character, and itself contains a list of with two elements:

1. A single character object listing any notes associated with the character
2. A named character vector listing the notes associated with each taxon for that character, named with the names of each note-bearing taxon.
Functions

- **PhyDat**: A convenient wrapper for *phangorn*’s `phyDat()`, which converts a list of morphological characters into a `phyDat` object. If your morphological characters are in the form of a matrix, perhaps because they have been read using `read.table()`, try `MatrixToPhyDat()` instead.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References


See Also

- Convert between matrices and `phyDat` objects: `MatrixToPhyDat()`
- Write characters to TNT-format file: `WriteTntCharacters()`

Examples

```r
fileName <- paste0(system.file(package = 'TreeTools'),
                    '/extdata/input/dataset.nex')
ReadCharacters(fileName)

fileName <- paste0(system.file(package = 'TreeTools'),
                    '/extdata/tests/continuous.nex')

continuous <- ReadCharacters(fileName, encoding = 'UTF8')

# To convert from strings to numbers:
at <- attributes(continuous)
continuous <- suppressWarnings(as.numeric(continuous))
attributes(continuous) <- at
continuous
```

---

**ReadTntTree**

*Parse TNT Tree*

Description

Read a tree from TNT’s parenthetical output.
Usage

ReadTntTree(filepath, relativePath = NULL, keepEnd = 1L, tipLabels = NULL)

TntText2Tree(treeText)

TNTText2Tree(treeText)

Arguments

filepath character string specifying path to TNT .tre file, relative to the R working directory (visible with getwd()).

relativePath (discouraged) character string specifying location of the matrix file used to generate the TNT results, relative to the current working directory. Taxon names will be read from this file if they are not specified by tipLabels.

keepEnd (optional, default 1) integer specifying how many elements of the file path to conserve when creating relative path (see examples).

tipLabels (optional) character vector specifying the names of the taxa, in the sequence that they appear in the TNT file. If not specified, taxon names will be loaded from the data file linked in the first line of the .tre file specified in filepath.

treeText Character string describing a tree, in the parenthetical format output by TNT.

Details

TNT is software for parsimony analysis. Whilst its implementation of tree search is extremely rapid, analysis of results in TNT is made difficult by its esoteric and scantly documented scripting language.

ReadTntTree() aims to aid the user by facilitating the import of trees generated in TNT into R for further analysis.

The function depends on tree files being saved by TNT in parenthetical notation, using the TNT command tsav*. Trees are easiest to load into R if taxa have been saved using their names (TNT command taxname=). In this case, the TNT .tre file contains tip labels and can be parsed directly. The downside is that the uncompressed .tre files will have a larger file size.

ReadTntTree() can also read .tre files in which taxa have been saved using their numbers (tax-name-). Such files contain a hard-coded link to the matrix file that was used to generate the trees, in the first line of the .tre file. This poses problems for portability: if the matrix file is moved, or the .tre file is accessed on another computer, the taxon names may be lost. As such, it is important to check that the matrix file exists in the expected location – if it does not, either use the relativePath argument to point to its new location, or specify tipLabels to manually specify the tip labels.

TntText2Tree() converts text representation of a tree in TNT to an object of class phylo.

Value

ReadTntTree() returns a tree of class phylo, corresponding to the tree in filepath, or NULL if no trees are found.
Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

Examples

# In the examples below, TNT has read a matrix from
# "c:/TreeTools/input/dataset.nex"
# The results of an analysis were written to
# "c:/TreeTools/output/results1.tnt"
# results1.tnt will contain a hard-coded reference to
# "c:/TreeTools/input/dataset.nex".
# On the original machine (but not elsewhere), it would be possible to read
# this hard-coded reference from results.tnt:
# ReadTntTree('output/results1.tnt')

# These datasets are provided with the 'TreeTools' package, which will
# probably not be located at c:/TreeTools on your machine:

oldWD <- getwd() # Remember the current working directory
setwd(system.file(package = 'TreeTools'))

# If taxon names were saved within the file (using 'taxname=' in TNT),
# then our job is easy:
ReadTntTree('extdata/output/named.tre')

# But if taxa were compressed to numbers (using 'taxname-'), we need to
# look up the original matrix in order to dereference the tip names.
# We need to extract the relevant file path from the end of the
# hard-coded path in the original file.
# We are interested in the last two elements of
# c:/TreeTools/input/dataset.nex
# 2 1
# '.' means "relative to the current directory"
ReadTntTree('extdata/output/numbered.tre', '.extdata', 2)

# If working in a lower subdirectory
setwd('./extdata/otherfolder')

# then it will be necessary to navigate up the directory path with '..':
ReadTntTree('../output/numbered.tre', '..', 2)

setwd(oldWD) # Restore original working directory

TNTText2Tree("(A (B (C (D E ))));")
Description

Renumber() numbers the nodes and tips in a tree to conform with the phylo standards.

Usage

Renumber(tree)

Arguments

tree A tree of class phylo.

Details

The 'ape' class phylo is not formally defined, but expects trees' internal representation to conform to certain principles: for example, nodes should be numbered sequentially, with values increasing away from the root.

Renumber() attempts to reformat any tree into a representation that will not cause 'ape' functions to produce unwanted results or to crash R.

Value

Renumber() returns a tree of class phylo, numbered in a Cladewise fashion consistent with the expectations of 'ape' functions.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Preorder() provides a faster and simpler alternative, but also rotates nodes.

Other tree manipulation: AddTip(), CollapseNode(), ConsensusWithout(), DropTip(), EnforceOutgroup(), ImposeConstraint(), LeafLabelInterchange(), MakeTreeBinary(), RenumberTips(), RenumberTree(), RootTree(), SingleTaxonTree(), SortTree(), Subtree()

Examples

tree <- RandomTree(letters[1:10])
Renumber(tree)
RenumberTips

Description

RenumberTips(tree, tipOrder) sorts the tips of a phylogenetic tree tree such that the indices in tree$edge[,2] correspond to the order of leaves given in tipOrder.

Usage

RenumberTips(tree, tipOrder)

## S3 method for class 'phylo'
RenumberTips(tree, tipOrder)

## S3 method for class 'multiPhylo'
RenumberTips(tree, tipOrder)

## S3 method for class 'list'
RenumberTips(tree, tipOrder)

## S3 method for class ```NULL```
RenumberTips(tree, tipOrder)

Arguments

- **tree**: A tree of class phylo.
- **tipOrder**: A character vector containing the values of tree$tip.label in the desired sort order, or an object (perhaps of class phylo or Splits) with tip labels.

Value

RenumberTips() returns tree, with the tips’ internal representation numbered to match tipOrder.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other tree manipulation: `AddTip()`, `CollapseNode()`, `ConsensusWithout()`, `DropTip()`, `EnforceOutgroup()`, `ImposeConstraint()`, `LeafLabelInterchange()`, `MakeTreeBinary()`, `RenumberTree()`, `Renumber()`, `RootTree()`, `SingleTaxonTree()`, `SortTree()`, `Subtree()`
RightmostCharacter

Examples

```r
data('Lobo') # Loads the phyDat object Lobo.phy
tree <- RandomTree(Lobo.phy)
tree <- RenumberTips(tree, names(Lobo.phy))
```

RightmostCharacter Rightmost character of string

Description

RightmostCharacter() is a convenience function that returns the final character of a string.

Usage

```r
RightmostCharacter(string, len = nchar(string))
```

Arguments

- `string`: Character string.
- `len`: (Optional) Integer specifying number of characters in string.

Value

RightmostCharacter() returns the rightmost character of a string.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other string parsing functions: `EndSentence()`, `MorphoBankDecode()`, `Unquote()`

Examples

```r
RightmostCharacter("Hello, World!")
```
RootNode

Which node is a tree’s root?

Description

RootNode() identifies the root node of a (rooted or unrooted) phylogenetic tree. Unrooted trees are represented internally by a rooted tree with a polytomy at the root.

Usage

RootNode(x)

Arguments

x A tree of class phylo, or its edge matrix; or a list or multiPhylo object containing multiple trees.

Value

RootNode() returns an integer denoting the root node for each tree. Badly conformed trees trigger an error.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Test whether a tree is rooted: TreeIsRooted()
phangorn::getRoot()

Other tree navigation: AncestorEdge(), CladeSizes(), DescendantEdges(), EdgeAncestry(), EdgeDistances(), ListAncestors(), MRCA(), NDescendants(), NodeDepth(), NodeOrder(), NonDuplicateRoot()

Examples

RootNode(BalancedTree(8))
RootNode(UnrootTree(BalancedTree(8)))
RootTree

Root or unroot a phylogenetic tree

Description

RootTree() roots a tree on the smallest clade containing the specified tips; RootOnNode() roots a tree on a specified internal node; UnrootTree() collapses a root node, without the undefined behaviour encountered when using \texttt{ape::unroot()} on trees in preorder.

Usage

\begin{verbatim}
RootTree(tree, outgroupTips)
\end{verbatim}

\begin{verbatim}
RootOnNode(tree, node, resolveRoot = FALSE)
\end{verbatim}

\begin{verbatim}
UnrootTree(tree)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{tree} \hspace{1cm} A tree of class \texttt{phylo}, or a list of trees of class \texttt{list} or \texttt{multiPhylo}.
  \item \texttt{outgroupTips} \hspace{1cm} Vector of type character, integer or logical, specifying the names or indices of the tips to include in the outgroup. If \texttt{outgroupTips} is of type character, and a tree contains multiple tips with a matching label, the first will be used.
  \item \texttt{node} \hspace{1cm} integer specifying node (internal or tip) to set as the root.
  \item \texttt{resolveRoot} \hspace{1cm} logical specifying whether to resolve the root node.
\end{itemize}

Details

Note: Edge lengths are not (yet) supported. Contact the maintainer or file a GitHub issue if you would find this useful.

Value

\begin{itemize}
  \item \texttt{RootTree()} returns a tree of class \texttt{phylo}, rooted on the smallest clade that contains the specified tips, with edges and nodes numbered in preorder.
  \item \texttt{RootOnNode()} returns a tree of class \texttt{phylo}, rooted on the requested node and ordered in Preorder.
  \item \texttt{UnrootTree()} returns \texttt{tree}, in preorder, having collapsed the first child of the root node in each tree.
\end{itemize}

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)
See Also

- ape::root()
- EnforceOutgroup()

Other tree manipulation: AddTip(), CollapseNode(), ConsensusWithout(), DropTip(), EnforceOutgroup(), ImposeConstraint(), LeafLabelInterchange(), MakeTreeBinary(), RenumberTips(), RenumberTree(), Renumber(), SingleTaxonTree(), SortTree(), Subtree()

Examples

tree <- PectinateTree(8)
plot(tree)
ape::nodelabels()

plot(RootTree(tree, c('t6', 't7')))

plot(RootOnNode(tree, 12))
plot(RootOnNode(tree, 2))

---

sapply64

Apply a function that returns 64-bit integers over a list or vector

Description

Wrappers for members of the \lapply()\ family intended for use when a function \texttt{FUN}\ returns a vector of integer64 objects. \texttt{vapply()}, \texttt{sapply()} or \texttt{replicate()} drop the integer64 class, resulting in a vector of numerics that require conversion back to 64-bit integers. These functions restore the missing class attribute.

Usage

\begin{verbatim}
sapply64(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
\end{verbatim}

\begin{verbatim}
vapply64(X, FUN, FUN.LEN = 1, ...)
\end{verbatim}

\begin{verbatim}
replicate64(n, expr, simplify = "array")
\end{verbatim}

Arguments

\textbf{X} a vector (atomic or list) or an \texttt{expression} object. Other objects (including classed objects) will be coerced by \texttt{base::as.list}.

\textbf{FUN} the function to be applied to each element of \texttt{X;} see ‘Details’. In the case of functions like +, \%,\%, the function name must be backquoted or quoted.

\textbf{...} optional arguments to \texttt{FUN}. 
**Details**

For details of the underlying functions, see `base::sapply()`.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

`bit64::integer64()`

**Examples**

```r
sapply64(as.phylo(1:6, 6), as.TreeNumber)
vapply64(as.phylo(1:6, 6), as.TreeNumber, 1)
set.seed(0)
replicate64(6, as.TreeNumber(RandomTree(6)))
```

---

**SingleTaxonTree**

Generate a single taxon tree

**Description**

`SingleTaxonTree()` creates a phylogenetic 'tree' that contains a single taxon.

**Usage**

```r
SingleTaxonTree(label)
```

**Arguments**

- `label` a character vector specifying the label of the tip.

**Value**

`SingleTaxonTree()` returns a phylo object containing a single tip with the specified label.
See Also

Other tree manipulation: `AddTip()`, `CollapseNode()`, `ConsensusWithout()`, `DropTip()`, `EnforceOutgroup()`, `ImposeConstraint()`, `LeafLabelInterchange()`, `MakeTreeBinary()`, `RenumberTips()`, `RenumberTree()`, `Renumber()`, `RootTree()`, `SortTree()`, `Subtree`

Other tree generation functions: `ConstrainedNJ()`, `GenerateTree`, `NJTree()`, `TreeNumber`

Examples

```r
SingleTaxonTree('Homo_sapiens')
plot(SingleTaxonTree('root') + BalancedTree(4))
```

---

SortTree  Sort tree

Description

`SortTree()` sorts each node into a consistent order, so that node rotation does not obscure similarities between similar trees.

Usage

```r
SortTree(tree)
```

## S3 method for class `phylo`
```r
SortTree(tree)
```

## S3 method for class `list`
```r
SortTree(tree)
```

## S3 method for class `multiPhylo`
```r
SortTree(tree)
```

Arguments

- `tree` One or more trees of class `phylo`, optionally as a list or a `multiPhylo` object.

Details

At each node, clades will be listed in `tree$edge` in decreasing size order.

Clades that contain the same number of leaves are sorted in decreasing order of minimum leaf number, so (2, 3) will occur before (1, 4).

As trees are plotted from ‘bottom up’, the largest clades will ‘sink’ to the bottom of a plotted tree.

`tree` must (presently) be binary (#25).
**Value**

`SortTree()` returns tree in the format of tree, with each node in each tree sorted such that the larger clade is first.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

`Preorder()` also rearranges trees into a consistent shape, but based on the index of leaves rather than the size of subtrees.

Other tree manipulation: `AddTip()`, `CollapseNode()`, `ConsensusWithout()`, `DropTip()`, `EnforceOutgroup()`, `ImposeConstraint()`, `LeafLabelInterchange()`, `MakeTreeBinary()`, `RenumberTips()`, `RenumberTree()`, `Renumber()`, `RootTree()`, `SingleTaxonTree()`, `Subtree()`

**Examples**

```r
messyTree <- as.phylo(10, 6)
plot(messyTree)

sorted <- SortTree(messyTree)
plot(sorted)
ape::nodelabels()
ape::edgelabels()
```

**Description**

`SplitFrequency()` provides a simple way to count the number of times that bipartition splits, as defined by a reference tree, occur in a forest of trees. May be used to calculate edge ("node") support for majority consensus or bootstrap trees.

**Usage**

```r
SplitFrequency(reference, forest)
SplitNumber(tips, tree, tipIndex, powersOf2)
ForestSplits(forest, powersOf2)
TreeSplits(tree)
```
SplitFrequency

Arguments

- **reference**: A tree of class phylo, a Splits object.
- **forest**: A list of trees of class phylo, or a multiPhylo object; or a Splits object. See `vignette` for possible methods of loading trees into R.
- **tips**: Integer vector specifying the tips of the tree within the chosen split.
- **tree**: A tree of class phylo.
- **tipIndex**: Character vector of tip names, in a fixed order.
- **powersOf2**: Integer vector of same length as tipIndex, specifying a power of 2 to be associated with each tip in turn.

Details

If multiple calculations are required, some time can be saved by using the constituent functions (see examples)

Value

SplitFrequency() returns the number of trees in forest that contain each split in reference. If reference is a tree of class phylo, then the sequence will correspond to the order of nodes (use `ape::nodelabels()` to view). Note that the three nodes at the root of the tree correspond to a single split; see the example for how these might be plotted on a tree.

Functions

- SplitNumber: Assign a unique integer to each split
- ForestSplits: Frequency of splits in a given forest of trees
- TreeSplits: Deprecated. Listed the splits in a given tree. Use as.Splits instead.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other Splits operations: LabelSplits(), NSplits(), NTip(), PolarizeSplits(), SplitsInBinaryTree(), Splits, TipLabels(), TipsInSplits(), match()

Examples

# An example forest of 100 trees, some identical
forest <- as.phylo(c(1, rep(10, 79), rep(100, 15), rep(1000, 5)), nTip = 9)

# Generate an 80% consensus tree
cons <- ape::consensus(forest, p = 0.8)
plot(cons)

splitFreqs <- SplitFrequency(cons, forest)
LabelSplits(cons, splitFreqs, unit = '%'
Description

Calculate the phylogenetic information content (*sensu* Steel & Penny, 2006) of a split, which reflects the probability that a uniformly selected random tree will contain the split: a split that is consistent with a smaller number of trees will have a higher information content.

Usage

SplitInformation(A, B = A[1])

MultiSplitInformation(partitionSizes)

Arguments

A  Integer specifying the number of taxa in each partition.
B  Integer specifying the number of taxa in each partition.
partitionSizes  Integer vector specifying the number of taxa in each partition of a multi-partition split.

Details

SplitInformation() addresses bipartition splits, which correspond to edges in an unrooted phylogeny; MultiSplitInformation() supports splits that subdivide taxa into multiple partitions, which may correspond to multi-state characters in a phylogenetic matrix.

A simple way to characterise trees is to count the number of edges. (Edges are almost, but not quite, equivalent to nodes.) Counting edges (or nodes) provides a quick measure of a tree’s resolution, and underpins the Robinson-Foulds tree distance measure. Not all edges, however, are created equal.

An edge splits the leaves of a tree into two subdivisions. The more equal these subdivisions are in size, the more instructive this edge is. Intuitively, the division of mammals from reptiles is a profound revelation that underpins much of zoology; recognizing that two species of bat are more closely related to each other than to any other mammal or reptile is still instructive, but somewhat less fundamental.

Formally, the phylogenetic (Shannon) information content of a split $S$, $h(S)$, corresponds to the probability that a uniformly selected random tree will contain the split, $P(S)$: $h(S) = -\log P(S)$. Base 2 logarithms are typically employed to yield an information content in bits.

As an example, the split $AB|CDEF$ occurs in 15 of the 105 six-leaf trees; $h(AB|CDEF) = -\log P(AB|CDEF) = -\log(15/105) \approx 2.81$ bits. The split $ABC|DEF$ subdivides the leaves more evenly, and is thus more instructive: it occurs in just nine of the 105 six-leaf trees, and $h(ABC|DEF) = -\log(9/105) \approx 3.54$ bits.
As the number of leaves increases, a single even split may contain more information than multiple uneven splits – see the examples section below.

Summing the information content of all splits within a tree, perhaps using the 'TreeDist' function `SplitwiseInfo()`, arguably gives a more instructive picture of its resolution than simply counting the number of splits that are present – though with the caveat that splits within a tree are not independent of one another, so some information may be double counted. (This same charge applies to simply counting nodes, too.)

Alternatives would be to count the number of quartets that are resolved, perhaps using the 'Quartet' function `QuartetStates()`, or to use a different take on the information contained within a split, the clustering information: see the 'TreeDist' function `ClusteringInfo()` for details.

Value

`SplitInformation()` and `MultiSplitInformation()` return the phylogenetic information content, in bits, of a split that subdivides leaves into partitions of the specified sizes.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References


See Also

- Sum the phylogenetic information content of splits within a tree: `TreeDist::SplitwiseInfo()`
- Sum the clustering information content of splits within a tree: `TreeDist::ClusteringInfo()`
- Other split information functions: `CharacterInformation()`, `SplitMatchProbability()`, `TreesMatchingSplit()`, `UnrootedTreesMatchingSplit()`

Examples

```r
# Eight leaves can be split evenly:
SplitInformation(4, 4)

# or unevenly, which is less informative:
SplitInformation(2, 6)

# A single split that evenly subdivides 50 leaves contains more information
# that seven maximally uneven splits on the same leaves:
SplitInformation(25, 25)
7 * SplitInformation(2, 48)

# Three ways to split eight leaves into multiple partitions:
MultiSplitInformation(c(2, 2, 4))
MultiSplitInformation(c(2, 3, 3))
MultiSplitInformation(rep(2, 4))
```
SplitMatchProbability  \hspace{1em} Probability of matching this well

**Description**

(Ln)SplitMatchProbability() calculates the probability that two random splits of the sizes provided will be at least as similar as the two specified.

**Usage**

SplitMatchProbability(split1, split2)

LnSplitMatchProbability(split1, split2)

**Arguments**

split1, split2  Logical vectors listing terminals in same order, such that each terminal is identified as a member of the ingroup (TRUE) or outgroup (FALSE) of the respective bipartition split.

**Value**

SplitMatchProbability() returns a numeric giving the proportion of permissible non-trivial splits that divide the terminals into bipartitions of the sizes given, that match as well as split1 and split2 do.

LnSplitMatchProbability() returns the natural logarithm of the probability.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Other split information functions: CharacterInformation(), SplitInformation(), TreesMatchingSplit(), UnrootedTreesMatchingSplit()

**Examples**

split1 <- as.Splits(c(rep(TRUE, 4), rep(FALSE, 4)))
split2 <- as.Splits(c(rep(TRUE, 3), rep(FALSE, 5)))
SplitMatchProbability(split1, split2)
LnSplitMatchProbability(split1, split2)
Splits

Convert object to Splits

Description

as.Splits() converts a phylogenetic tree to a Splits object representing its constituent bipartition splits.

Usage

as.Splits(x, tipLabels = NULL, ...)

## S3 method for class 'phylo'
as.Splits(x, tipLabels = NULL, asSplits = TRUE, ...)

## S3 method for class 'multiPhylo'
as.Splits(x, tipLabels = x[[1]]$tip.label, asSplits = TRUE, ...)

## S3 method for class 'Splits'
as.Splits(x, tipLabels = NULL, ...)

## S3 method for class 'list'
as.Splits(x, tipLabels = NULL, asSplits = TRUE, ...)

## S3 method for class 'matrix'
as.Splits(x, tipLabels = NULL, ...)

## S3 method for class 'logical'
as.Splits(x, tipLabels = NULL, ...)

## S3 method for class 'Splits'
as.logical(x, tipLabels = NULL, ...)

Arguments

x Object to convert into splits: perhaps a tree of class phylo. If a logical matrix is provided, each row will be considered as a separate split.

tipLabels Character vector specifying sequence in which to order tip labels. Label order must (currently) match to combine or compare separate Splits objects.

... Presently unused.

asSplits Logical specifying whether to return a Splits object, or an unannotated two-dimensional array (useful where performance is paramount).
Value

as.Splits() returns an object of class Splits, or (if asSplits = FALSE) a two-dimensional array of raw objects, with each bit specifying whether or not the leaf corresponding to the respective bit position is a member of the split. Splits are named according to the node at the non-root end of the edge that defines them. In rooted trees, the child of the rightmost root edge names the split.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other Splits operations: LabelSplits(), NSplits(), NTip(), PolarizeSplits(), SplitFrequency(), SplitsInBinaryTree(), TipLabels(), TipsInSplits(), match()

Examples

splits <- as.Splits(BalancedTree(letters[1:6]))
split <- SplitsInBinaryTree(tree)

SplitsInBinaryTree

Maximum splits in an n-leaf tree

Description

SplitsInBinaryTree() is a convenience function to calculate the number of splits in a fully-resolved (binary) tree with n leaves.

Usage

SplitsInBinaryTree(tree)

# S3 method for class 'list'
SplitsInBinaryTree(tree)
SplitsInBinaryTree

## S3 method for class 'multiPhylo'
SplitsInBinaryTree(tree)

## S3 method for class 'numeric'
SplitsInBinaryTree(tree)

## S3 method for class 'NULL'
SplitsInBinaryTree(tree)

## Default S3 method:
SplitsInBinaryTree(tree)

## S3 method for class 'Splits'
SplitsInBinaryTree(tree)

## S3 method for class 'phylo'
SplitsInBinaryTree(tree)

### Arguments

tree
An object of a supported format that represents a tree or set of trees, from which
the number of leaves will be calculated.

### Value

SplitsInBinaryTree() returns an integer vector detailing the number of unique non-trivial splits
in a binary tree with \( n \) leaves.

### Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

### See Also

Other tree properties: `ConsensusWithout()`, `NSplits()`, `NTip()`, `TipLabels()`, `TreeIsRooted()`
Other Splits operations: `LabelSplits()`, `NSplits()`, `NTip()`, `PolarizeSplits()`, `SplitFrequency()`,
`Splits`, `TipLabels()`, `TipsInSplits()`, `match()`

### Examples

tree <- BalancedTree(8)
SplitsInBinaryTree(tree)
SplitsInBinaryTree(as.Splits(tree))
SplitsInBinaryTree(8)
SplitsInBinaryTree(list(tree, tree))
Stemwardness

'Stemwardness' of a leaf

Description

Functions to describe the position of a leaf relative to the root. 'Stemmier' leaves ought to exhibit a smaller root-node distance and a larger sister size.

Usage

SisterSize(tree, tip)

## S3 method for class 'numeric'
SisterSize(tree, tip)

## S3 method for class 'character'
SisterSize(tree, tip)

RootNodeDistance(tree, tip)

## S3 method for class 'numeric'
RootNodeDistance(tree, tip)

## S3 method for class 'character'
RootNodeDistance(tree, tip)

RootNodeDist(tree, tip)

Arguments

tree A tree of class phylo.
tip Either a numeric specifying the index of a single tip, or a character specifying its label.

Details

RootNodeDistance() calculates the number of nodes between the chosen leaf and the root of tree. This is an unsatisfactory measure, as the range of possible distances is a function of the shape of the tree. As an example, leaf X1 in the tree (.,(.,(.,(.,(X1,(a,b)))))) falls outside the clade (a, b) and has a root-node distance of 4, whereas leaf X2 in the tree (.,(.,(.,(.,(b,(X2,a)))))) falls within the clade (a, b), so should be considered more 'crownwards', yet has a smaller root-node distance (3).
SisterSize() measures the number of leaves in the clade that is sister to the chosen leaf. In the examples above, $X_1$ has a sister size of 2 leaves, whereas $X_2$, which is 'more crownwards', has a smaller sister size (1 leaf), as desired.

**Value**

SisterSize() returns an integer specifying the number of leaves in the clade that is sister to tip. RootNodeDist() returns an integer specifying the number of nodes between tip and the root node of tree.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**References**


**See Also**

Other tree characterization functions: CladisticInfo(), Consensus(), TotalCopheneticIndex()
Examples

bal8 <- BalancedTree(8)
pec8 <- PectinateTree(8)

SisterSize(bal8, 3)
SisterSize(pec8, 't3')
SisterSize(RootTree(pec8, 't3'), 't3')

RootNodeDist(bal8, 3)
RootNodeDist(pec8, 't3')
RootNodeDist(RootTree(pec8, 't3'), 't3')

StringToPhyDat

Convert between strings and phyDat objects

Description

PhyDatToString() converts a phyDat object as a string; StringToPhyDat() converts a string of character data to a phyDat object.

Usage

StringToPhyDat(string, tips, byTaxon = TRUE)

StringToPhydat(string, tips, byTaxon = TRUE)

PhyToString(
    phy,
    parentheses = "{",
    collapse = "",
    ps = "",
    useIndex = TRUE,
    byTaxon = TRUE,
    concatenate = TRUE
)

PhyDatToString(
    phy,
    parentheses = "{",
    collapse = "",
    ps = "",
    useIndex = TRUE,
    byTaxon = TRUE,
    concatenate = TRUE
)

PhydatToString()
StringToPhyDat

phy,
parentheses = "{",
collapse = "",
ps = "",
useIndex = TRUE,
byTaxon = TRUE,
concatenate = TRUE
)

Arguments

string   String of tokens, optionally containing whitespace, with no terminating semicolon.
tips     Character vector corresponding to the names (in order) of each taxon in the matrix, or an objects such as a tree from which tip labels can be extracted.
byTaxon  Logical. If TRUE, write one taxon followed by the next. If FALSE, write one character followed by the next.
phy      An object of class phyDat.
parentheses Character specifying format of parentheses with which to surround ambiguous tokens. Choose from: { (default), [.(<.
collapse  Character specifying text, perhaps ,, with which to separate multiple tokens within parentheses.
ps       Character specifying text, perhaps ;, to append to the end of the string.
useIndex Logical (default: TRUE) specifying whether to print duplicate characters multiple times, as they appeared in the original matrix.
concatenate Logical specifying whether to concatenate all characters/taxa into a single string, or to return a separate string for each entry.

Value

StringToPhyDat() returns an object of class phyDat.

PhyToString() returns a character vector listing a text representation of the phylogenetic character state for each taxon in turn.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other phylogenetic matrix conversion functions: MatrixToPhyDat()
Subsplit

**Examples**

```r
StringLength("-?01231230?-", c('Lion', 'Gazelle'), byTaxon = TRUE)
# encodes the following matrix:
# Lion -?0123
# Gazelle 1230?-

fileName <- paste0(system.file(package='TreeTools'),
                     '/extdata/input/dataset.nex')
phyDat <- ReadAsPhyDat(fileName)
PhyToString(phyDat, concatenate = FALSE)
```

---

Subsplit

*Subset of a split on fewer leaves*

**Description**

Subsplit() removes leaves from a Splits object.

**Usage**

```r
Subsplit(splits, tips, keepAll = FALSE, unique = TRUE)
```

**Arguments**

- `splits` An object of class *Splits*.
- `tips` A vector specifying a subset of the leaf labels applied to split.
- `keepAll` logical specifying whether to keep entries that define trivial splits (i.e. splits of zero or one leaf) on the subset of leaves.
- `unique` logical specifying whether to remove duplicate splits.

**Value**

Subsplit() returns an object of class Splits, defined on the leaves tips.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Other split manipulation functions: TrivialSplits()
Examples

```r
splits <- as.Splits(PectinateTree(letters[1:9]))
splits
efgh <- Subsplit(splits, tips = letters[5:8], keepAll = TRUE)
split
TrivialSplits(efgh)

summary(Subsplit(splits, tips = letters[5:8], keepAll = FALSE))
```

---

**Subtree**

*Extract a subtree*

**Description**

Subtree() safely extracts a clade from a phylogenetic tree.

**Usage**

```r
Subtree(tree, node)
```

**Arguments**

- `tree` A tree of class `phylo`, with internal numbering in cladewise order (use `Preorder(tree)` or (slower) `Cladewise(tree)`).
- `node` The number of the node at the base of the clade to be extracted.

**Details**

Modified from the `ape` function `extract.clade`, which sometimes behaves erratically. Unlike extract.clade, this function supports the extraction of 'clades' that constitute a single tip.

**Value**

Subtree() returns a tree of class `phylo` that represents a clade extracted from the original tree.

**Author(s)**

Martin R. Smith (martin.smith@durham.ac.uk)

**See Also**

Other tree manipulation: `AddTip()`, `CollapseNode()`, `ConsensusWithout()`, `DropTip()`, `EnforceOutgroup()`, `ImposeConstraint()`, `LeafLabelInterchange()`, `MakeTreeBinary()`, `RenumberTips()`, `RenumberTree()`, `Renumber()`, `RootTree()`, `SingleTaxonTree()`, `SortTree()`
Examples

```r
tree <- Preorder(BalancedTree(8))
plot(tree)
ape::nodelabels()
ape::nodelabels(13, 13, bg='yellow')
plot(Subtree(tree, 13))
```

---

### SupportColour

**Colour for node support value**

**Description**

Colour value with which to display node support.

**Usage**

```r
SupportColour(
  support,
  show1 = TRUE,
  scale = rev(diverge_hcl(101, h = c(260, 0), c = 100, l = c(50, 90), power = 1)),
  outOfRange = "red"
)
```

**Arguments**

- `support` A numeric vector of values in the range 0–1.
- `show1` Logical specifying whether to display values of 1. A transparent white will be returned if `FALSE`.
- `scale` 101-element vector listing colours in sequence. Defaults to a diverging HCL scale.
- `outOfRange` Colour to use if results are outside the range 0–1.

**Value**

`SupportColour()` returns the appropriate value from `scale`, or `outOfRange` if a value is outside the valid range.
See Also

Use in conjunction with `LabelSplits()` to colour split labels, possibly calculated using `SplitFrequency()`.

Examples

```r
SupportColour((-1):4 / 4, show1 = FALSE)

# An example forest of 100 trees, some identical
forest <- as.phylo(c(1, rep(10, 79), rep(100, 15), rep(1000, 5)), nTip = 9)

# Generate an 80% consensus tree
cons <- ape::consensus(forest, p = 0.8)
plot(cons)

splitFreqs <- SplitFrequency(cons, forest)
LabelSplits(cons, splitFreqs, unit = '%',
    col = SupportColor(splitFreqs / 100),
    frame = 'none', pos = 3L)
```

---

**TipLabels**

*Extract tip labels*

---

**Description**

`TipLabels()` extracts labels from an object: for example, names of taxa in a phylogenetic tree or data matrix. `AllTipLabels()` extracts all labels, where entries of a list of trees may pertain to different taxa.

**Usage**

```r
TipLabels(x, single = TRUE)
```

```r
## S3 method for class 'matrix'
TipLabels(x, single = TRUE)
```

```r
## S3 method for class 'phylo'
TipLabels(x, single = TRUE)
```

```r
## Default S3 method:
TipLabels(x, single = TRUE)
```

```r
## S3 method for class 'phyDat'
TipLabels(x, single = TRUE)
```

```r
## S3 method for class 'TreeNumber'
TipLabels(x, single = TRUE)
```
## Arguments

- **x**: An object of a supported class (see Usage section above).
- **single**: Logical specifying whether to report the labels for the first object only (TRUE), or for each object in a list (FALSE).
Value

TipLabels() returns a character vector listing the tip labels appropriate to x. If x is a single integer, this will be a vector t1, t2 ... tx, to match the default of ape::rtree().

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other tree properties: ConsensusWithout(), NSplits(), NTip(), SplitsInBinaryTree(), TreeIsRooted()
Other Splits operations: LabelSplits(), NSplits(), NTip(), PolarizeSplits(), SplitFrequency(), SplitsInBinaryTree(), Splits, TipsInSplits(), match()

Examples

TipLabels(BalancedTree(letters[5:1]))
TipLabels(5)

data('Lobo')
head(TipLabels(Lobo.phy))

AllTipLabels(c(BalancedTree(4), PectinateTree(8)))

TipsInSplits

Tips contained within splits

Description

TipsInSplits() specifies the number of tips that occur within each bipartition split in a Splits object.

Usage

TipsInSplits(splits, keep.names = TRUE, ...)

## S3 method for class 'Splits'
TipsInSplits(splits, keep.names = TRUE, ...)

## S3 method for class 'phylo'
TipsInSplits(splits, keep.names = TRUE, ...)

SplitImbalance(splits, keep.names = TRUE, ...)

## S3 method for class 'Splits'
SplitImbalance(splits, keep.names = TRUE, ...)
## S3 method for class 'phylo'
SplitImbalance(splits, keep.names = TRUE, ...)

### Arguments

- **splits**: Object of class `Splits` or `phylo`.
- **keep.names**: Logical specifying whether to include the names of `splits` in the output.
- **...**: Additional parameters to pass to `as.Splits()`.

### Value

- `TipsInSplits()` returns a named vector of integers, specifying the number of tips contained within each split in `splits`.
- `SplitImbalance()` returns a named vector of integers, specifying the number of leaves within a split that are not 'balanced' by a leaf outside it; i.e. a split that divides leaves evenly has an imbalance of zero; one that splits two tips from ten has an imbalance of 10 - 2 = 8.

### Examples

```r
tree <- PectinateTree(8)
splits <- as.Splits(tree)
TipsInSplits(splits)
plot(tree)
LabelSplits(tree, as.character(splits), frame = 'none', pos = 3L, cex = 0.7)
LabelSplits(tree, TipsInSplits(splits), unit = 'tips', frame = 'none',
            pos = 1L)
```

### Description

`TotalCopheneticIndex()` calculates the total cophenetic index (Mir *et al.* 2013) for any tree, a measure of its balance; `TCIContext()` lists its possible values.
Usage

TotalCopheneticIndex(x)

TCIContext(x)

## S3 method for class 'numeric'
TCIContext(x)

Arguments

x
A tree of class phylo, its $edge property, or a list thereof.

Details

The Total Cophenetic Index is a measure of tree balance – i.e. whether a (phylogenetic) tree comprises symmetric pairs of nodes, or has a pectinate 'caterpillar' shape. The index has a greater resolution power than Sackin’s and Colless’ indices, and can be applied to trees that are not perfectly resolved.

For a tree with $n$ leaves, the Total Cophenetic Index can take values of 0 to $\text{choose}(n,3)$. The minimum value is higher for a perfectly resolved (i.e. dichotomous) tree (see Lemma 14 of Mir et al. 2013). Formulae to calculate the expected values under the Yule and Uniform models of evolution are given in Theorems 17 and 23.

Full details are provided by Mir et al. (2013).

Value

TotalCopheneticIndex() returns an integer denoting the total cophenetic index.

TCIContext() returns a data frame detailing the maximum and minimum value obtainable for the Total Cophenetic Index for rooted binary trees with the number of leaves of the given tree, and the expected value under the Yule and Uniform models. The variance of the expected value is given under the Yule model, but cannot be obtained by calculation for the Uniform model.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References


See Also

cophen.index() in the package 'CollessLike' provides an alternative implementation of this index and its predecessors.

Other tree characterization functions: CladisticInfo(), Consensus(), Stemwardness
Examples

# Balanced trees have the minimum index for a binary tree;
# Pectinate trees the maximum:
TCIContext(8)
TotalCopheneticIndex(PectinateTree(8))
TotalCopheneticIndex(BalancedTree(8))
TotalCopheneticIndex(StarTree(8))

# Examples from Mir et al. (2013):
tree12 <- ape::read.tree(text="'(1, 2, (3, (4, 5))));'"; #Fig. 4, tree 12
TotalCopheneticIndex(tree12) # 10

tree8 <- ape::read.tree(text="'((1, 2, 3, 4), 5);'"; #Fig. 4, tree 8
TotalCopheneticIndex(tree8) # 6
TCIContext(tree8)
TCIContext(5L) # Context for a tree with 5 leaves.

TreeIsRooted

Is tree rooted?

Description

TreeIsRooted() is a fast alternative to ape::is.rooted().

Usage

TreeIsRooted(tree)

Arguments

tree A phylogenetic tree of class phylo.

Value

TreeIsRooted() returns a logical specifying whether a root node is resolved.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other tree properties: ConsensusWithout(), NSplits(), NTip(), SplitsInBinaryTree(), TipLabels()

Examples

TreeIsRooted(BalancedTree(6))
TreeIsRooted(UnrootTree(BalancedTree(6)))
TreeNumber

Unique integer indices for bifurcating tree topologies

Description

Functions converting between phylogenetic trees and their unique decimal representation.

Usage

as.TreeNumber(x, ...)

## S3 method for class 'phylo'
as.TreeNumber(x, ...)

## S3 method for class 'multiPhylo'
as.TreeNumber(x, ...)

## S3 method for class 'character'
as.TreeNumber(x, nTip, tipLabels = TipLabels(nTip), ...)

## S3 method for class 'numeric'
as.phylo(x, nTip = attr(x, "nTip"), tipLabels = attr(x, "tip.label"), ...)

## S3 method for class 'TreeNumber'
as.phylo(x, nTip = attr(x, "nTip"), tipLabels = attr(x, "tip.label"), ...)

Arguments

x Integer identifying the tree (see details).
...
Additional parameters for consistency with S3 methods (unused).
nTip Integer specifying number of leaves in the tree.
tipLabels Character vector listing the labels assigned to each tip in a tree, perhaps obtained using TipLabels().

Details

There are \( \text{NUnrooted}(n) \) unrooted trees with \( n \) leaves. As such, each \( n \)-leaf tree can be uniquely identified by a non-negative integer \( x < \text{NUnrooted}(n) \).

This integer can be converted by a tree by treating it as a mixed-base number, with bases 1, 3, 5, 7, ...
\( (2n - 5) \).

Each digit of this mixed base number corresponds to a leaf, and determines the location on a growing tree to which that leaf should be added.

We start with a two-leaf tree, and treat 0 as the origin of the tree.

\[ 0 \quad \text{----} \quad 1 \]
We add leaf 2 by breaking an edge and inserting a node (numbered $2 + n_{\text{Tip}} - 1$). In this example, we’ll work up to a six-leaf tree; this node will be numbered $2 + 6 - 1 = 7$. There is only one edge on which leaf 2 can be added. Let’s add node 7 and leaf 2:

$$0 ---- 7 ---- 1$$
$$|$$
$$|$$
$$2$$

There are now three edges on which leaf 3 can be added. Our options are:
Option 0: the edge leading to 1;
Option 1: the edge leading to 2;
Option 2: the edge leading to 7.
If we select option 1, we produce:

$$0 ---- 7 ---- 1$$
$$|$$
$$|$$
$$8 ---- 2$$
$$|$$
$$|$$
$$3$$

1 is now the final digit of our mixed-base number.
There are five places to add leaf 4:
Option 0: the edge leading to 1;
Option 1: the edge leading to 2;
Option 2: the edge leading to 3;
Option 3: the edge leading to 7;
Option 4: the edge leading to 8.
If we chose option 3, then 3 would be the penultimate digit of our mixed-base number.
If we chose option 0 for the next two additions, we could specify this tree with the mixed-base number 0021. We can convert this into decimal:

$$0 \times (1 \times 3 \times 5 \times 9) +$$
$$0 \times (1 \times 3 \times 5) +$$
$$3 \times (1 \times 3) +$$
$$1 \times (1)$$

= 10

Note that the hyperexponential nature of tree space means that there are $> 2^{64}$ unique 20-leaf trees. As a TreeNumber is a 64-bit integer, only trees with at most 19 leaves can be accommodated.
TreesMatchingSplit

Value

as.TreeNumber() returns an object of class TreeNumber, which comprises a numeric vector, whose elements represent successive nine-digit chunks of the decimal integer corresponding to the tree topology (in big endian order). The TreeNumber object has attributes nTip and tip.labels.

as.phylo.numeric() returns a tree of class phylo.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References

Based on a concept by John Tromp, employed in Li et al. 1996.

See Also

Describe the shape of a tree topology, independent of leaf labels: TreeShape()
Other tree generation functions: ConstrainedNJ(), GenerateTree, NJTree(), SingleTaxonTree()

Examples

```r
tree <- as.phylo(10, nTip = 6)
plot(tree)
as.TreeNumber(tree)

# Larger trees:
as.TreeNumber(BalancedTree(19))

# If > 9 digits, represent the tree number as a string.
treeNode <- as.TreeNumber("1234567890123", nTip = 14)
tree <- as.phylo(treeNode)
as.phylo(0:2, nTip = 6, tipLabels = letters[1:6])
```

---

TreesMatchingSplit

Number of trees matching a bipartition split

Description

Calculates the number of unrooted bifurcated trees that are consistent with a bipartition split that divides taxa into groups of size A and B.
Usage

TreesMatchingSplit(A, B = A[2])
LnTreesMatchingSplit(A, B = A[2])
Log2TreesMatchingSplit(A, B = A[2])

Arguments

A, B  Integer specifying the number of taxa in each partition.

Value

TreesMatchingSplit() returns a numeric specifying the number of trees that are compatible with the given split.
LnTreesMatchingSplit() and Log2TreesMatchingSplit() give the natural and base-2 logarithms of this number.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

Other split information functions: CharacterInformation(), SplitInformation(), SplitMatchProbability(), UnrootedTreesMatchingSplit()

Examples

TreesMatchingSplit(5, 6)
LnTreesMatchingSplit(5, 6)
Log2TreesMatchingSplit(5, 6)

| TreesMatchingTree | Number of trees containing a tree |

Description

TreesMatchingTree() calculates the number of unrooted binary trees that are consistent with a tree topology on the same leaves.

Usage

TreesMatchingTree(tree)
LnTreesMatchingTree(tree)
Log2TreesMatchingTree(tree)
Arguments

tree A tree of class `phylo`.

Details

Remember to unroot a tree first if the position of its root is arbitrary.

Value

TreesMatchingTree() returns a numeric specifying the number of unrooted binary trees that contain all the edges present in the input tree.

LnTreesMatchingTree() gives the natural logarithm of this number.

Author(s)

Martin R. Smith (`martin.smith@durham.ac.uk`)

See Also

Other tree information functions: `CladisticInfo()`, `NRooted()`

Examples

```r
partiallyResolvedTree <- CollapseNode(BalancedTree(8), 12:15)
TreesMatchingTree(partiallyResolvedTree)
LnTreesMatchingTree(partiallyResolvedTree)

# Number of rooted trees:
rootedTree <- AddTip(partiallyResolvedTree, where = 0)
TreesMatchingTree(partiallyResolvedTree)
```

TrivialSplits

Identify and remove trivial splits

Description

TrivialSplits() identifies trivial splits (which separate one or zero leaves from all others); WithoutTrivialSplits() removes them from a Splits object.

Usage

```r
TrivialSplits(splits, nTip = attr(splits, "nTip"))
WithoutTrivialSplits(splits, nTip = attr(splits, "nTip"))
```

Arguments

- `splits` An object of class `Splits`.
- `nTip` Integer specifying number of tips (leaves).
Unquote

Value
TrivialSplits() returns a logical vector specifying whether each split in splits is trivial, i.e. includes or excludes only a single tip or no tips at all.
WithoutTrivialSplits() returns a Splits object with trivial splits removed.

Author(s)
Martin R. Smith (martin.smith@durham.ac.uk)

See Also
Other split manipulation functions: Subsplit()

Examples
splits <- as.Splits(PectinateTree(letters[1:9]))
efgh <- Subsplit(splits, tips = letters[5:8], keepAll = TRUE)
summary(efgh)

TrivialSplits(efgh)
summary(WithoutTrivialSplits(efgh))

Unquote
Remove quotation marks from a string

Description
Remove quotation marks from a string

Usage
Unquote(string)

Arguments
string Input string

Value
Unquote() returns string, with any matched punctuation marks and trailing whitespace removed.

Author(s)
Martin R. Smith

See Also
Other string parsing functions: EndSentence(), MorphoBankDecode(), RightmostCharacter()
Examples

Unquote("'Hello World'")

UnrootedTreesMatchingSplit

*Number of trees consistent with split*

Description

Calculates the number of unrooted bifurcating trees consistent with the specified multi-partition split, using the formula of Carter *et al.* (1990).

Usage

UnrootedTreesMatchingSplit(...)

LnUnrootedTreesMatchingSplit(...)

Log2UnrootedTreesMatchingSplit(...)

Arguments

... A series or vector of integers listing the number of tips in each of a number of tree splits (e.g. bipartitions). For example, 3, 5 states that a character divides a set of eight tips into a group of three and a group of five.

Value

UnrootedTreesMatchingSplit() returns an integer specifying the number of unrooted bifurcating trees consistent with the specified split.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

References


See Also

Other split information functions: CharacterInformation(), SplitInformation(), SplitMatchProbability(), TreesMatchingSplit()
UnshiftTree

Examples

Examples

UnrootedTreesMatchingSplit(c(3, 5))
UnrootedTreesMatchingSplit(3, 2, 1, 2)

UnshiftTree  Add tree to start of list

Description

UnshiftTree() adds a phylogenetic tree to the start of a list of trees. This is useful where the class of a list of trees is unknown, or where names of trees should be retained.

Usage

UnshiftTree(add, treeList)

Arguments

add Tree to add to the list, of class phylo.
treeList A list of trees, of class list, multiPhylo, or, if a single tree, phylo.

Details

Caution: adding a tree to a multiPhylo object whose own attributes apply to all trees, for example trees read from a Nexus file, causes data to be lost.

Value

UnshiftTree() returns a list of class list or multiPhylo (following the original class of treeList), whose first element is the tree specified as ‘add.

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

c() joins a tree or series of trees to a multiPhylo object, but loses names and does not handle lists of trees.

Examples

Examples

forest <- as.phylo(0:5, 6)
tree <- BalancedTree(6)

UnshiftTree(tree, forest)
UnshiftTree(tree, tree)
WriteTntCharacters

Write morphological character matrix to TNT file

Description

Write morphological character matrix to TNT file

Usage

WriteTntCharacters(
  dataset,
  filepath = NULL,
  comment = "Dataset written by 'TreeTools::WriteTntCharacters()'",
  types = NULL,
  pre = "",
  post = ""
)

## S3 method for class 'phyDat'
WriteTntCharacters(
  dataset,
  filepath = NULL,
  comment = "Dataset written by 'TreeTools::WriteTntCharacters()'",
  types = NULL,
  pre = "",
  post = ""
)

## S3 method for class 'matrix'
WriteTntCharacters(
  dataset,
  filepath = NULL,
  comment = "Dataset written by 'TreeTools::WriteTntCharacters()'",
  types = NULL,
  pre = "",
  post = ""
)

Arguments

dataset Morphological dataset of class phyDat or matrix.
filepath Path to file; if NULL, returns a character vector.
comment Optional comment with which to entitle matrix.
types Optional list specifying where different data types begin. c(num = 1, dna = 10) sets characters 1..9 as numeric, 10..end as DNA.
WriteTntCharacters

pre, post

Character vector listing text to print before and after the character matrix. Specify pre = 'piwe='; if the matrix is to be analysed using extended implied weighting (xpiwe=).

Author(s)

Martin R. Smith (martin.smith@durham.ac.uk)

See Also

ReadTntCharacters()

Examples

data('Lobo', package = 'TreeTools')

WriteTntCharacters(Lobo.phy)

# Read with extended implied weighting
WriteTntCharacters(Lobo.phy, pre = 'piwe=10;', post = 'xpiwe=')

# Write to a file with:
# WriteTntCharacters(Lobo.phy, 'example_file.tnt')
Index

* Splits operations
  * LabelSplits, 29
  * match, 35
  * NSplits, 48
  * NTip, 49
  * PolarizeSplits, 51
  * SplitFrequency, 65
  * Splits, 70
  * SplitsInBinaryTree, 71
  * TipLabels, 80
  * TipsInSplits, 82

* consensus tree functions
  * Consensus, 16
  * ConsensusWithout, 17

* datasets
  * brewer, 11
  * doubleFactorials, 21
  * Lobo.data, 33
  * logDoubleFactorials, 33
  * nRootedShapes, 47

* double factorials
  * DoubleFactorial, 20
  * doubleFactorials, 21
  * logDoubleFactorials, 33

* methods
  * match, 35

* pairwise tree distances
  * PairwiseDistances, 50

* phylogenetic matrix conversion functions
  * StringToPhyDat, 75

* split information functions
  * CharacterInformation, 11
  * SplitInformation, 67
  * SplitMatchProbability, 69
  * TreesMatchingSplit, 88
  * UnrootedTreesMatchingSplit, 92

* split information function
  * NRouted, 45

* split manipulation functions
  * Subsplit, 77
  * TrivialSplits, 90

* string parsing functions
  * EndSentence, 25
  * MorphoBankDecode, 36
  * RightmostCharacter, 59
  * Unquote, 91

* tree characterization functions
  * CladisticInfo, 13
  * Consensus, 16
  * Stemwardness, 73
  * TotalCopheneticIndex, 83

* tree generation functions
  * ConstrainedNJ, 18
  * GenerateTree, 26
  * NJTree, 41
  * SingleTaxonTree, 63
  * TreeNumber, 86

* tree information functions
  * CladisticInfo, 13
  * NRouted, 45
  * TreesMatchingTree, 89

* tree manipulation
  * AddTip, 4
  * CollapseNode, 14
  * ConsensusWithout, 17
  * DropTip, 22
  * EnforceOutgroup, 25
  * ImposeConstraint, 28
  * LeafLabelInterchange, 30
  * MakeTreeBinary, 34
  * Renumber, 57
  * RenumberTips, 58
  * RootTree, 61
  * SingleTaxonTree, 63
  * SortTree, 64
  * Subtree, 78

* tree navigation
  * CladeSizes, 12
INDEX

DescendantEdges, 19
EdgeAncestry, 23
EdgeDistances, 24
ListAncestors, 31
MRCA, 37
NDescendants, 40
NodeDepth, 42
NodeOrder, 43
RootNode, 60
* tree properties
  ConsensusWithout, 17
  NSplits, 48
  NTip, 49
  SplitsInBinaryTree, 71
  TipLabels, 80
  TreesIsRooted, 85
* tree
  AddTip, 4
  SingleTaxonTree, 63
  %in% (match), 35
  AddTip, 4, 15, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 65, 78
  AddTip(), 24, 29
  AddTipEverywhere (AddTip), 4
  AddUnconstrained (ImposeConstraint), 28
  AllAncestors (ListAncestors), 31
  AllDescendantEdges (DescendantEdges), 19
  AllTipLabels (TipLabels), 80
  AncestorEdge, 12, 20, 23, 24, 32, 37, 40, 43, 44, 60
  ape::consensus(), 17
  ape::drop.tip(), 22
  ape::edgelabels(), 29
  ape::mst(), 39
  ape::multi2di(), 34
  ape::node.depth, 43
  ape::ntip(), 49
  ape::root(), 62
  ape::unroot, 61
  ape::write.tree(), 10
  ApeTime, 6
array, 63
ArtEx (ArtificialExtinction), 6
ArtificialExtinction, 6
as.list, 62
as.logical.Splits (Splits), 70
as.multiPhylo, 9
as.Newick, 10, 41
as.phylo.numeric (TreeNumber), 86
as.phylo.TreeNumber (TreeNumber), 86
as.Splits (Splits), 70
as.Splits(), 29
as.TreeNumber (TreeNumber), 86
BalancedTree (GenerateTree), 26
base::lapply(), 63
bind.tree, 5
bit64::integer64(), 63
brewer, 11
c(), 93
CharacterInformation, 11, 68, 69, 89, 92
CladeSizes, 12, 20, 23, 24, 32, 37, 40, 43, 44, 60
Cladewise, 23, 57, 78
CladisticInfo, 13, 16, 46, 74, 84, 90
CladisticInformation (CladisticInfo), 13
CollapseEdge (CollapseNode), 14
CollapseNode, 5, 14, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 65, 78
connection, 53
Consensus, 14, 16, 18, 74, 84
ConsensusWithout, 5, 15, 16, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 65, 72, 78, 82, 85
ConstrainedNJ, 18, 27, 42, 64, 88
DescendantEdges, 12, 19, 23, 24, 32, 37, 40, 43, 44, 60
DoubleFactorial, 20, 21, 34
DoubleFactorial64 (DoubleFactorial), 20
doubleFactorials, 21, 21, 34
DropTip, 5, 15, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 65, 78
EdgeAncestry, 12, 20, 23, 24, 32, 37, 40, 43, 44, 60
EdgeDistances, 12, 20, 23, 24, 32, 37, 40, 43, 44, 60
edgelabels, 15, 19
EndSentence, 25, 36, 59, 91
EnforceOutgroup, 5, 15, 17, 22, 25, 28, 31, 34, 57, 58, 62, 64, 65, 78
EnforceOutgroup(), 62
expression, 62
extract.clade, 78
ForestSplits (SplitFrequency), 65
GenerateTree, 19, 26, 42, 64, 88
getRoot(), 60
IC1Spr (N1Spr), 39
ImposeConstraint, 5, 15, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 65, 78
in.Splits (match), 35
KeepTip (DropTip), 22
LabelSplits, 29, 35, 49–51, 66, 71, 72, 82, 83
LabelSplits(), 80
language object, 63
lapply(), 62
LeafLabelInterchange, 5, 15, 17, 22, 26, 28, 30, 34, 57, 58, 62, 64, 65, 78
legend(), 17
ListAncestors, 12, 20, 23, 24, 31, 37, 40, 43, 44, 60
ListAncestors(), 37
LnDoubleFactorial (DoubleFactorial), 20
LnRooted (NRooted), 45
LnSplitMatchProbability (SplitMatchProbability), 69
LnTreesMatchingSplit (TreesMatchingSplit), 88
LnTreesMatchingTree (TreesMatchingTree), 89
LnUnrooted (NRooted), 45
LnUnrootedMult (NRooted), 45
LnUnrootedSplits (NRooted), 45
LnUnrootedTreesMatchingSplit (UnrootedTreesMatchingSplit), 92
Lobo.data, 33
Lobo.phy (Lobo.data), 33
Log2DoubleFactorial (DoubleFactorial), 20
Log2DoubleFactorials, 21, 33
MakeTreeBinary, 5, 15, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 65, 78
MarkMissing (ConsensusWithout), 17
match, 29, 35, 49–51, 66, 71, 72, 82, 83
match(), 35
MatrixToPhyDat, 76
MatrixToPhyDat(), 54
MorphoBankDecode, 25, 36, 59, 91
MRCA, 12, 20, 23, 24, 32, 37, 40, 43, 44, 60
MSTEdges, 38
MSTLength (MSTEdges), 38
multiPhylo, 93
MultiSplitInformation (SplitInformation), 67
N1Spr, 39
names, 63
NDescendants, 12, 20, 23, 24, 32, 37, 40, 43, 44, 60
NewickTree, 41
NewickTree(), 10
NJTree, 19, 27, 41, 64, 88
NodeDepth, 12, 20, 23, 24, 32, 37, 40, 42, 44, 46, 60
nodelabels, 15
NodeOrder, 12, 20, 23, 24, 32, 37, 40, 43, 43, 60
NonDuplicateRoot, 12, 20, 23, 24, 32, 37, 40, 43, 44, 46, 60
NPartitionPairs, 44
NPartitions (NSplits), 48
NRooted, 14, 45, 90
NRooted64 (NRooted), 45
nRootedShapes, 47
NSplits, 18, 29, 35, 48, 50, 51, 66, 71, 72, 82, 83, 85
NTip, 18, 29, 35, 49, 49, 51, 66, 71, 72, 82, 83, 85
nUnrooted (NRooted), 45
nUnrooted64 (NRooted), 45
nRootedShapes (nRootedShapes), 47
nUnrootedSplits (NRooted), 45
PairwiseDistances, 50
PectinateTree (GenerateTree), 26
PhyDat (ReadCharacters), 52
phyDat, 18, 41, 53, 75, 76
PhyDatToString (StringToPhyDat), 75
PhyDatToString (StringToPhyDat), 75
phylo, 4, 12, 13, 19, 22–24, 28–31, 34, 40, 41, 57, 58, 61, 66, 70, 73, 78, 90, 93
PhylogeneticInfo (CladisticInfo), 13
PhylogeneticInformation (CladisticInfo), 13
PhyToString (StringToPhyDat), 75
PolarizeSplits, 29, 35, 49, 50, 51, 66, 71, 72, 82, 83
Preorder, 22, 23, 31, 61, 78
print.TreeNumber, 52
RandomTree (GenerateTree), 26
read.nexus.data, 53
read.table(), 54
ReadAsPhyDat (ReadCharacters), 52
ReadCharacters, 52
ReadNotes (ReadCharacters), 52
ReadTntAsPhyDat (ReadCharacters), 52
ReadTntCharacters (ReadCharacters), 52
ReadTntCharacters(), 95
ReadTntTree, 54
Renumber, 5, 15, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 65, 78
RenumberTips, 5, 15, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 65, 78
RenumberTips(), 10
RenumberTree, 5, 15, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 65, 78
replicate64 (sapply64), 62
RightmostCharacter, 25, 36, 59, 91
RootNode, 12, 20, 23, 24, 32, 37, 40, 43, 44, 60
RootNodeDist (Stemwardness), 73
RootNodeDistance (Stemwardness), 73
RootOnNode (RootTree), 61
RootTree, 5, 15, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 65, 78
RootTree(), 26
rtree, 82
sapply64, 62
SingleTaxonTree, 5, 15, 17, 19, 22, 26–28, 31, 34, 42, 57, 58, 62, 63, 65, 78, 88
SisterSize (Stemwardness), 73
SortTree, 5, 15, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 64, 78
SplitFrequency, 29, 35, 49–51, 65, 71, 72, 82, 83
SplitFrequency(), 29, 80
SplitImbalance (TipsInSplits), 82
SplitInformation, 12, 67, 69, 89, 92
SplitMatchProbability, 12, 68, 69, 89, 92
SplitNumber (SplitFrequency), 65
Splits, 29, 35, 49–51, 66, 70, 72, 77, 82, 83, 90
SplitsInBinaryTree, 18, 29, 35, 49–51, 66, 71, 72, 83, 85
StarTree (GenerateTree), 26
Stemwardness, 14, 16, 73, 84
StringToPhyDat, 75
StringToPhydat (StringToPhyDat), 75
Subsplit, 77, 91
Subtree, 5, 15, 17, 22, 26, 28, 31, 34, 57, 58, 62, 64, 65, 78
SupportColor (SupportColour), 79
SupportColour, 79
SupportColour(), 29
TCIContext (TotalCopheneticIndex), 83
TipLabels, 18, 29, 35, 49–51, 66, 71, 72, 80, 83, 85
TipLabels(), 27, 86
TipsInSplits, 29, 35, 49–51, 66, 71, 72, 82, 83
TNTText2Tree (ReadTntTree), 54
TntText2Tree (ReadTntTree), 54
TotalCopheneticIndex, 14, 16, 74, 83
TreeIsRooted, 18, 49, 50, 72, 82, 85
TreeIsRooted(), 60
TreeNode, 19, 27, 42, 64, 86
TreeShape(), 88
TreesMatchingSplit, 12, 68, 69, 88, 92
TreesMatchingTree, 14, 46, 89
TreeSplits (SplitFrequency), 65
TrivialSplits, 77, 90
Unquote, 25, 36, 59, 91
UnrootedTreesMatchingSplit, 12, 68, 69, 89, 92
UnrootTree (RootTree), 61
UnshiftTree, 93
vapply64 (sapply64), 62
WithoutTrivialSplits (TrivialSplits), 90
write.tree().41
WriteTntCharacters.94
WriteTntCharacters().54