Package ‘TreeTools’

March 19, 2022

Title Create, Modify and Analyse Phylogenetic Trees

Version 1.7.0

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;
plotting the position of rogue taxa (Klopfstein & Spasojevic 2019)
<doi:10.1371/journal.pone.0212942>;
calculation of ancestor-descendant relationships,
of 'stemwardness' (Asher & Smith, 2021) <doi:10.1093/sysbio/syab072>,
and of tree balance (Mir et al. 2013) <doi:10.1016/j.mbs.2012.10.005>;
artificial extinction (Asher & Smith, 2022) <doi:10.1093/sysbio/syab072>;
import and export of trees from Newick, Nexus (Maddison et al. 1997)
<doi:10.1093/sysbio/46.4.590>,
and TNT <http://www.lillo.org.ar/phylogeny/tnt/> formats;
and analysis of splits and cladistic information.

URL https://ms609.github.io/TreeTools/, https://github.com/ms609/TreeTools/

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

SystemRequirements C++14

Depends R (>= 3.4.0),
ape (>= 5.6),

Imports bit64,
lifecycle,
colorspace,
fastmatch (>= 1.1.3),
methods,
R.cache,
Rdpack (>= 2.2),

Suggests spelling,
knitr,
phangorn (>= 2.2.1),
Rcpp (>= 1.0.8),
rmarkdown,
shiny,
testthat (>= 3.0),
vdiffr (>= 1.0.0),

Config/Needs/check rcmdcheck

Config/Needs/coverage covr

Config/Needs/memcheck devtools, rcmdcheck

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

Roxygen list(markdown = TRUE)

R topics documented:

AddTip 4
ApeTime 6
ArtificialExtinction 6
as.multiPhylo 8
as.Newick 9
brewer 10
CharacterInformation 11
CladeSizes 12
CladisticInfo 12
CollapseNode 14
Consensus 15
ConsensusWithout 16
ConstrainedNJ 17
DescendantEdges 18
DoubleFactorial 19
doubleFactorials 20
DropTip 21
EdgeAncestry 22

EdgeDistances . . . 23
edge_to_splits . . . 24
EndSentence . . . 25
EnforceOutgroup . . . 26
GenerateTree . . . 27
Hamming . . . 28
ImposeConstraint . . . 29
KeptPaths . . . 30
KeptVerts . . . 31
LabelSplits . . . 32
LeafLabelInterchange . . . 33
ListAncestors . . . 34
Lobo.data . . . 35
logDoubleFactorials . . . 36
MakeTreeBinary . . . 37
match.Splits . . . 37
MatrixToPhyDat . . . 38
MorphoBankDecode . . . 40
MRCA . . . 40
MSTEdges . . . 41
N1Spr . . . 42
NDescendants . . . 43
NewickTree . . . 44
NJTree . . . 45
NodeDepth . . . 45
NodeOrder . . . 46
NPartitionPairs . . . 47
NRooted . . . 48
nRootedShapes . . . 50
NSplits . . . 51
NTip . . . 52
PairwiseDistances . . . 53
PathLengths . . . 54
PolarizeSplits . . . 55
print.TreeNumber . . . 55
ReadCharacters . . . 56
ReadTntTree . . . 58
Renumber . . . 60
RenumberTips . . . 61
RightmostCharacter . . . 62
RoguePlot . . . 63
RootNode . . . 65
RootTree . . . 66
sapply64 . . . 67
SingleTaxonTree . . . 68
sort.multiPhylo . . . 69
SortTree . . . 70
SpectrumLegend . . . 71
SplitFrequency . . . 72
SplitInformation . . . 73
SplitMatchProbability . . . 75
Splits . . . 76

SplitsInBinaryTree . . . 78
Stemwardness . . . 79
StringToPhyDat . . . 81
Subsplit . . . 83
Subtree . . . 84
SupportColour . . . 85
TipLabels . . . 86
TipsInSplits . . . 88
TotalCopheneticIndex . . . 89
TreeIsRooted . . . 91
TreeNumber . . . 91
TreesMatchingSplit . . . 95
TreesMatchingTree . . . 96
TrivialSplits . . . 97
Unquote . . . 98
UnrootedTreesMatchingSplit . . . 98
UnshiftTree . . . 99
WriteTntCharacters . . . 100
xor . . . 101

Index **103**

`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

`lengthBelow` 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.

`nTip, nNode, rootNode` 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.

`includeRoot` 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.

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()`, `KeptPaths()`, `KeptVerts()`, `LeafLabelInterchange()`, `MakeTreeBinary()`, `RenumberTips()`, `RenumberTree()`, `Renumber()`, `RootTree()`, `SingleTaxonTree()`, `SortTree()`, `Subtree()`

Examples

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

plot(AddTip(tree, 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 tokens. 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 (2022).

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

Asher R, Smith MR (2022). "Phylogenetic signal and bias in paleontology." *Systematic Biology*, **Online ahead of print**, syab072.

`\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1093/sysbio/syab072")}`

.

Examples

```
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)
```

---

as.Newick — *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 *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

```
brewer
```

Format

An object of class `list` of length 12.

Source

- ColourBrewer2.org
- Martin Krzywinski

Examples

```
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 *Character information content*

Description

`CharacterInformation()` calculates the cladistic information content (Steel and Penny 2006) of a given character, in bits. The total information in all characters gives a measure of the potential utility of a dataset (Cotton and Wilkinson 2008), which can be compared with a profile parsimony score (Faith and Trueman 2001) to evaluate the degree of homoplasy within a dataset.

Usage

```
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 ?.

Value

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

Author(s)

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

References

- Cotton JA, Wilkinson M (2008). "Quantifying the potential utility of phylogenetic characters." *Taxon*, **57**(1), 131–136.

  Faith DP, Trueman JWH (2001). "Towards an inclusive philosophy for phylogenetic inference." *Systematic Biology*, **50**(3), 331–350.

  `\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1080/10635150118627")}`

  .

  Steel MA, Penny D (2006). "Maximum parsimony and the phylogenetic information in multistate characters." In Albert VA (ed.), *Parsimony, Phylogeny, and Genomics*, 163–178. Oxford University Press, Oxford.

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()

Examples

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

CladisticInfo *Cladistic information content of a tree*

Description

CladisticInfo() calculates the cladistic (phylogenetic) information content of a phylogenetic object, *sensu* Thorley *et al.* (1998).

Usage

```
CladisticInfo(x)

PhylogeneticInfo(x)

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

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

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

## 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

Page RD (1992). "Comments on the information content of classifications." *Cladistics*, **8**(1), 87–95.

`\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1111/j.1096-0031.1992.tb00054.x")}`

.

Rohlf FJ (1982). "Consensus indices for comparing classifications." *Mathematical Biosciences*, **59**(1), 131–144.

\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1016/0025-5564(82)90112-2")}

, https://doi.org/10.1016/0025-5564(82)90112-2.

Steel MA, Penny D (2006). "Maximum parsimony and the phylogenetic information in multistate characters." In Albert VA (ed.), *Parsimony, Phylogeny, and Genomics*, 163–178. Oxford University Press, Oxford.

Thorley JL, Wilkinson M, Charleston M (1998). "The information content of consensus trees." In Rizzi A, Vichi M, Bock H (eds.), *Advances in Data Science and Classification*, 91–98. Springer, Berlin. ISBN 978-3-540-64641-9,

\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1007/978-3-642-72253-0")}

.

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

```
CollapseNode(tree, nodes)

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

CollapseEdge(tree, edges)
```

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(), KeptPaths(), KeptVerts(), LeafLabelInterchange(), MakeTreeBinary(), RenumberTips(), RenumberTree(), Renumber(), RootTree(), SingleTaxonTree(), SortTree(), Subtree()

Examples

```
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)

par(oldPar)
```

Consensus *Construct consensus trees*

Description

Consensus() calculates the consensus of a set of trees, using the algorithm of (Day 1985).

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)

References

Day WHE (1985). "Optimal algorithms for comparing trees with labeled leaves." *Journal of Classification*, **2**(1), 7–28.

`\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1007/BF01908061")}`

.

See Also

`TreeDist::ConsensusInfo()` calculates the information content of a consensus tree.

Other consensus tree functions: `ConsensusWithout()`, `RoguePlot()`

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), ...)

## 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(), KeptPaths(), KeptVerts(), LeafLabelInterchange(), MakeTreeBinary(), RenumberTips(), RenumberTree(), Renumber(), RootTree(), SingleTaxonTree(), SortTree(), Subtree()

Other tree properties: NSplits(), NTip(), PathLengths(), SplitsInBinaryTree(), TipLabels(), TreeIsRooted()

Other consensus tree functions: Consensus(), RoguePlot()

Examples

```
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, 't2'))
MarkMissing('t2')

par(oldPar)
```

ConstrainedNJ *Constrained neighbour-joining tree*

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.

Usage

```
ConstrainedNJ(dataset, constraint, weight = 1L, ratio = TRUE, ambig = "mean")
```

Arguments

dataset	A phylogenetic data matrix of class phyDat, whose names correspond to the labels of any accompanying tree.
constraint	An object of class phyDat; returned trees will be perfectly compatible with each character in constraint. See vignette for further examples.
weight	Numeric specifying degree to up-weight characters in constraint.
ratio	Settings of ambig and ratio to be used when computing Hamming() distances between sequences.
ambig	Settings of ambig and ratio to be used when computing Hamming() distances between sequences.

Value

ConstrainedNJ() returns a tree of class phylo.

Author(s)

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

See Also

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

Examples

```
dataset <- MatrixToPhyDat(matrix(
  c(0, 1, 1, 1, 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.

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 * (n - 2) * (n - 4) * (n - 6) * ...

Functions

- `DoubleFactorial64`: Returns the exact double factorial as a 64-bit `integer64`, 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

```
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 leaves from tree*

Description

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

Usage

```
DropTip(tree, tip, preorder = TRUE, check = TRUE)

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

KeepTipPreorder(tree, tip)

KeepTipPostorder(tree, tip)

## S3 method for class 'Splits'
DropTip(tree, tip, preorder, check = TRUE)

DropTipPhylo(tree, tip, preorder = TRUE, check = TRUE)

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

KeepTip(tree, tip, preorder = TRUE, check = 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 tree before dropping tips. Specifying FALSE saves a little time, but will result in undefined behaviour if tree is not in preorder.
check	Logical specifying whether to check validity of tip. If FALSE and tip contains entries that do not correspond to leaves of the tree, undefined behaviour may occur.

Details

This function differs from ape::drop.tip(), which roots unrooted trees, and which can crash when trees' internal numbering follows unexpected schema.

Value

DropTip() returns a tree of class phylo, with the requested leaves removed. The edges of the tree will be numbered in preorder, but their sequence may not conform to the conventions of Preorder().

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

Functions

- KeepTipPreorder: Faster version with no checks. Does not retain tip labels or edge weights. edges must be listed in preorder. May crash if improper input is specified.
- KeepTipPostorder: Faster version with no checks. Does not retain tip labels or edge weights. edges must be listed in postorder. May crash if improper input is specified.
- DropTipPhylo: Direct call to DropTip.phylo(), to avoid overhead of querying object's class.

Author(s)

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

See Also

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

Other split manipulation functions: Subsplit(), TrivialSplits()

Examples

```
tree <- BalancedTree(9)
plot(tree)
plot(DropTip(tree, c('t5', 't6')))

unrooted <- UnrootTree(tree)
plot(unrooted)
plot(DropTip(unrooted, 4:5))

summary(DropTip(as.Splits(tree), 4:5))
```

EdgeAncestry *Ancestors of an edge*

Description

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

Usage

```
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)

See Also

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

Examples

```
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))
```

EdgeDistances *Distance between edges*

Description

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

Usage

```
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)

See Also

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

Examples

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

EdgeDistances(tree)
```

---

edge_to_splits *Efficiently convert edge matrix to splits*

Description

Wrapper for internal C++ function for maximum efficiency. Improper input may crash R. Behaviour not guaranteed. It is advisable to contact the package maintainers before relying on this function.

Usage

```
edge_to_splits(
  edge,
  edgeOrder,
  tipLabels = NULL,
  asSplits = TRUE,
  nTip = NTip(edge),
  ...
)
```

Arguments

edge	A matrix with two columns, with each row listing the parent and child node of an edge in a phylogenetic tree. Property edge of objects of class phylo.
edgeOrder	Integer vector such that edge[edgeOrder,] returns a postorder ordering of edges.
tipLabels	Character vector specifying sequence in which to order tip labels. Label order must (currently) match to combine or compare separate Splits objects.

asSplits	Logical specifying whether to return a Splits object, or an unannotated two-dimensional array (useful where performance is paramount).
nTip	Integer specifying number of leaves in tree.
...	Presently unused.

Value

edge_to_splits() uses the same return format as as.Splits().

See Also

as.Splits() offers a safe access point to this function that should be suitable for most users.

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 specified 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)

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

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

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()`, `KeptPaths()`, `KeptVerts()`, `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, nodes)

PectinateTree(tips)

BalancedTree(tips)

StarTree(tips)
```

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.
nodes	Number of nodes to generate. The default and maximum, tips -1, generates a binary tree; setting a lower value will induce polytomies.

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 term 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, in preorder.

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]))
```

`Hamming` *Hamming distance between taxa in a phylogenetic dataset*

Description

The Hamming distance between a pair of taxa is the number of characters with a different coding, i.e. the smallest number of evolutionary steps that must have occurred since their common ancestor.

Usage

```
Hamming(
  dataset,
  ratio = TRUE,
  ambig = c("median", "mean", "zero", "one", "na", "nan")
)
```

Arguments

`dataset`	Object of class `phyDat`.
`ratio`	Logical specifying whether to weight distance against maximum possible, given that a token that is ambiguous in either of two taxa cannot contribute to the total distance between the pair.
`ambig`	Character specifying value to return when a pair of taxa have a zero maximum distance (perhaps due to a preponderance of ambiguous tokens). "median", the default, take the median of all other distance values; "mean", the mean; "zero" sets to zero; "one" to one; "NA" to `NA_integer_`; and "NaN" to `NaN`.

Details

Tokens that contain the inapplicable state are treated as requiring no steps to transform into any applicable token.

Value

`Hamming()` returns an object of class `dist` listing the Hamming distance between each pair of taxa.

Author(s)

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

See Also

Used to construct neighbour joining trees in NJTree().

dist.hamming() in the **phangorn** package provides an alternative implementation.

Examples

```
tokens <- matrix(c(0, 0, '0', 0, '?',
                   0, 0, '1', 0, 1,
                   0, 0, '1', 0, 1,
                   0, 0, '2', 0, 1,
                   1, 1, '-', '?', 0,
                   1, 1, '2', 1, '{01}'),
                   nrow = 6, ncol = 5, byrow = TRUE,
                   dimnames = list(
                     paste0("Taxon_", LETTERS[1:6]),
                     paste0("Char_", 1:5)))

dataset <- MatrixToPhyDat(tokens)
Hamming(dataset)
```

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(), KeptPaths(), KeptVerts(), LeafLabelInterchange(), MakeTreeBinary(), RenumberTips(), RenumberTree(), Renumber(), RootTree(), SingleTaxonTree(), SortTree(), Subtree()

Examples

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

constraint <- StringToPhyDat('0000?1111 000111111 0000??110', tips, FALSE)
plot(ImposeConstraint(tree, constraint))
```

KeptPaths *Paths present in reduced tree*

Description

Lists which paths present in a master tree are present when leaves are dropped.

Usage

```
KeptPaths(paths, keptVerts, all = TRUE)

## S3 method for class 'data.frame'
KeptPaths(paths, keptVerts, all = TRUE)

## S3 method for class 'matrix'
KeptPaths(paths, keptVerts, all = TRUE)
```

Arguments

paths	data.frame of paths in master tree, perhaps generated using PathLengths().
keptVerts	Logical specifying whether each entry is retained in the reduced tree, perhaps generated using KeptVerts().
all	Logical: if TRUE, return all paths that occur in the reduced tree; if FALSE, return only those paths that correspond to a single edge. that correspond to edges in the reduced tree. Ignored if paths is a matrix.

Value

KeptPaths() returns a logical vector specifying whether each path in paths occurs when keptVerts vertices are retained.

Author(s)

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

See Also

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

Examples

```
master <- BalancedTree(9)
paths <- PathLengths(master)
keptTips <- c(1, 5, 7, 9)
keptVerts <- KeptVerts(master, keptTips)
KeptPaths(paths, keptVerts)
paths[KeptPaths(paths, keptVerts, all = FALSE), ]
```

KeptVerts — *Identify vertices retained when leaves are dropped*

Description

Identify vertices retained when leaves are dropped

Usage

```
KeptVerts(tree, keptTips, tipLabels = TipLabels(tree))

## S3 method for class 'phylo'
KeptVerts(tree, keptTips, tipLabels = TipLabels(tree))

## S3 method for class 'numeric'
KeptVerts(tree, keptTips, tipLabels = TipLabels(tree))
```

Arguments

`tree`	Original tree of class `phylo`, in `Preorder`.
`keptTips`	Either: • a logical vector stating whether each leaf should be retained, in a sequence corresponding to `tree[["tip.label"]]`; or • a character vector listing the leaf labels to retain; or • a numeric vector listing the indices of leaves to retain.
`tipLabels`	Optional character vector naming the leaves of `tree`, if `keptTips` is not logical. Inferred from `tree` if unspecified.

Author(s)

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

See Also

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

Examples

```
master <- BalancedTree(12)
master <- Preorder(master) # Nodes must be listed in Preorder sequence
plot(master)
nodelabels()

allTips <- master[["tip.label"]]
keptTips <- sample(allTips, 8)
plot(KeepTip(master, keptTips))
kept <- KeptVerts(master, allTips %in% keptTips)

map <- which(kept)
# Node `i` in the reduced tree corresponds to node `map[i]` in the original.
```

`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.Splits`, `xor()`

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.

Author(s)

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

See Also

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

Examples

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

ListAncestors *List ancestors*

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.

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:
AllAncestors(edge[, 1], edge[, 2])
```

Lobo.data *Data from Zhang et al. 2016*

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

Zhang et al. (2016)

References

Zhang X, Smith MR, Yang J, Hou J (2016). "Onychophoran-like musculature in a phosphatized Cambrian lobopodian." *Biology Letters*, **12**(9), 20160492.

`\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1098/rsbl.2016.0492")}`

.

Examples

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

`logDoubleFactorials` *Natural logarithms of double factorials*

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.

See Also

Other double factorials: `DoubleFactorial()`, `doubleFactorials`

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()`, `KeptPaths()`, `KeptVerts()`, `LeafLabelInterchange()`, `RenumberTips()`, `RenumberTree()`, `Renumber()`, `RootTree()`, `SingleTaxonTree()`, `SortTree()`, `Subtree()`

Examples

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

match.Splits — *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

```
## S4 method for signature 'Splits,Splits'
match(x, table, nomatch = NA_integer_, incomparables = NULL)

in.Splits(x, table)

match(x, table, nomatch = NA_integer_, incomparables = NULL)

## S4 method for signature 'Splits,Splits'
x %in% table
```

Arguments

`x, table`	Object of class `Splits`.
`nomatch`	Integer value that will be used in place of `NA` in the case where no match is found.
`incomparables`	Ignored. (Included for consistency with generic.)

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.

See Also

Corresponding base functions are documented in `match()`.

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

Examples

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

match(splits1, splits2)
```

`MatrixToPhyDat` *Convert between matrices and* `phyDat` *objects*

Description

`MatrixToPhyDat()` converts a matrix of tokens to a `phyDat` object; `PhyDatToMatrix()` converts a `phyDat` object to a matrix of tokens.

Usage

```
MatrixToPhyDat(tokens)

PhyDatToMatrix(dataset, ambigNA = FALSE, inappNA = ambigNA)
```

Arguments

tokens	Matrix of tokens, possibly created with ReadCharacters() or ReadTntCharacters(). Row names should correspond to leaf labels; column names may optionally correspond to character labels.
dataset	A dataset of class phyDat.
ambigNA, inappNA	Logical specifying whether to denote ambiguous / inapplicable characters as NA values.

Value

MatrixToPhyDat() returns an object of class phyDat.

PhyDatToMatrix() returns a matrix corresponding to the uncompressed character states within a phyDat object.

Author(s)

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

See Also

Other phylogenetic matrix conversion functions: StringToPhyDat()

Examples

```
tokens <- matrix(c(0, 0, '0', 0, 0,
                   0, 0, '1', 0, 1,
                   0, 0, '1', 0, 1,
                   0, 0, '2', 0, 1,
                   1, 1, '-', 1, 0,
                   1, 1, '2', 1, '{01}'),
                   nrow = 6, ncol = 5, byrow = TRUE,
                   dimnames = list(
                     paste0("Taxon_", LETTERS[1:6]),
                     paste0("Char_", 1:5)))

MatrixToPhyDat(tokens)
```

MorphoBankDecode *Decode MorphoBank text*

Description

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

Usage

```
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

```
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

```
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 (Gower and Ross 1969) 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

Gower JC, Ross GJS (1969). "Minimum spanning trees and single linkage cluster analysis." *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, **18**(1), 54–64.

`\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.2307/2346439")}`

.

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 an numeric vector giving the phylogenetic information content of trees 0 or 1 SPR rearrangement from an *n*-leaf tree, in bits.

References

Allen BL, Steel MA (2001). "Subtree transfer operations and their induced metrics on evolutionary trees." *Annals of Combinatorics*, **5**(1), 1–15.

\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1007/s00026-001-8006-8")}

.

Examples

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

NDescendants *Count descendants for each node in a tree*

Description

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

Usage

```
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

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

NewickTree *Write Newick Tree*

Description

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

```
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

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

NJTree — *Generate a neighbour joining tree*

Description

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

Usage

```
NJTree(dataset, edgeLengths = FALSE, ratio = TRUE, ambig = "mean")
```

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.
ambig, ratio	Settings of ambig and ratio to be used when computing Hamming() distances between sequences.

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)
```

NodeDepth — *Distance of each node from tree exterior*

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.

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))
```

NodeOrder *Order of each node in a tree*

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)

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:

	A2	*B2*
A1	AB	C
B1	E	FGH

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 `sum(configuration)` taxa according to the specified pattern.

Author(s)

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

Examples

```
NPartitionPairs(c(2, 1, 1, 3))
```

`NRooted` *Number of trees*

Description

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

Usage

```
NRooted(tips)

NUnrooted(tips)

NRooted64(tips)

NUnrooted64(tips)

LnUnrooted(tips)

LnUnrooted.int(tips)

Log2Unrooted(tips)
```

```
Log2Unrooted.int(tips)

LnRooted(tips)

LnRooted.int(tips)

Log2Rooted(tips)

Log2Rooted.int(tips)

LnUnrootedSplits(...)

Log2UnrootedSplits(...)

NUnrootedSplits(...)

LnUnrootedMult(...)

Log2UnrootedMult(...)

NUnrootedMult(...)
```

Arguments

`tips`	Integer specifying the number of leaves.
`...`	Integer vector, or series of integers, listing the number of leaves in each split.

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.

Calculations follow Cavalli-Sforza and Edwards (1967) and Carter et al. (1990), Theorem 2.

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

Carter M, Hendy M, Penny D, Székely LA, Wormald NC (1990). "On the distribution of lengths of evolutionary trees." *SIAM Journal on Discrete Mathematics*, **3**(1), 38–47.

`\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1137/0403005")}`

.

Cavalli-Sforza LL, Edwards AWF (1967). "Phylogenetic analysis: models and estimation procedures." *Evolution*, **21**(3), 550–570. ISSN 00143820,

`\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1111/j.1558-5646.1967.tb03411.x")}`

.

See Also

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

Examples

```
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)
```

`nRootedShapes` *Number of rooted / unrooted tree shapes*

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

`nUnrootedShapes` is OEIS 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.

Author(s)

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

See Also

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

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

Examples

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

NTip — *Number of leaves in a phylogenetic tree*

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)
```

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(), PathLengths(), SplitsInBinaryTree(), TipLabels(), TreeIsRooted()

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

PairwiseDistances *Distances between each pair of trees*

Description

Distances between each pair of trees

Usage

```
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)
```

PathLengths *Calculate length of paths between each pair of vertices within tree*

Description

Given a weighted rooted tree `tree`, `PathLengths()` returns the distance from each vertex to each of its descendant vertices.

Usage

```
PathLengths(tree, fullMatrix = FALSE)
```

Arguments

`tree`	Original tree of class `phylo`, in `Preorder`.
`fullMatrix`	Logical specifying return format; see "value" section‘.

Value

If `fullMatrix = TRUE`, `PathLengths()` returns a square matrix in which entry [i, j] denotes the distance from internal node `i` to the descendant vertex `j`. Vertex pairs without a continuous directed path are denoted `NA`. If `fullMatrix = FALSE`, `PathLengths()` returns a `data.frame` with three columns: `start` lists the deepest node in each path (i.e. that closest to the root); `end` lists the shallowest node (i.e. that closest to a leaf); `length` lists the total length of that path.

Author(s)

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

See Also

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

Examples

```
tree <- rtree(6)
plot(tree)
add.scale.bar()
nodelabels()
tiplabels()
PathLengths(tree)
```

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.Splits, xor()

print.TreeNumber *Print* TreeNumber *object*

Description

S3 method for objects of class TreeNumber.

Usage

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

Arguments

x	Object of class TreeNumber.
...	Additional arguments for consistency with S3 method (unused).

ReadCharacters *Read phylogenetic characters from file*

Description

Parse a Nexus (Maddison et al. 1997) or TNT (Goloboff et al. 2008) file, reading character states and names.

Usage

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

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

ReadNotes(filepath, encoding = "UTF8")

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 (O'Leary and Kaufman 2011), 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

Goloboff PA, Farris JS, Nixon KC (2008). "TNT, a free program for phylogenetic analysis." *Cladistics*, **24**(5), 774–786.

Maddison DR, Swofford DL, Maddison WP (1997). "Nexus: an extensible file format for systematic information." *Systematic Biology*, **46**, 590–621.

`\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1093/sysbio/46.4.590")}`

.

O'Leary MA, Kaufman S (2011). "MorphoBank: phylophenomics in the "cloud"." *Cladistics*, **27**(5), 529–537.

See Also

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

Examples

```
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 (taxname-). 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 ))));")
```

Renumber *Renumber a tree's nodes and tips*

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(), KeptPaths(), KeptVerts(), LeafLabelInterchange(), MakeTreeBinary(), RenumberTips(), RenumberTree(), RootTree(), SingleTaxonTree(), SortTree(), Subtree()

Examples

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

RenumberTips *Renumber a tree's tips*

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(), KeptPaths(), KeptVerts(), LeafLabelInterchange(), MakeTreeBinary(), RenumberTree(), Renumber(), RootTree(), SingleTaxonTree(), SortTree(), Subtree()

Examples

```
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

```
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

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

RoguePlot *Visualize position of rogue taxa*

Description

Plots a consensus of trees with a rogue taxon omitted, with edges coloured according to the proportion of trees in which the taxon attaches to that edge, after Klopfstein and Spasojevic (2019).

Usage

```
RoguePlot(
  trees,
  tip,
  p = 1,
  plot = TRUE,
  Palette = colorRampPalette(c(par("fg"), "#009E73"), space = "Lab"),
  nullCol = rgb(colorRamp(unlist(par(c("fg", "bg"))), space = "Lab")(0.8)/255),
  edgeLength = NULL,
  thin = par("lwd"),
  fat = thin + 1L,
  outgroupTips,
  ...
)
```

Arguments

`trees`	List or `multiPhylo` object containing phylogenetic trees of class `phylo` to be summarized.
`tip`	Numeric or character identifying rogue leaf, in format accepted by `DropTip()`.
`p`	A numeric value between 0.5 and 1 giving the proportion for a clade to be represented in the consensus tree (see `Consensus()`).
`plot`	Logical specifying whether to plot the tree.
`Palette`	Function that takes a parameter `n` and generates a colour palette with `n` entries.
`nullCol`	Colour to paint regions of the tree on which the rogue is never found.
`edgeLength`	Numeric specifying edge lengths of consensus tree; `NULL` aligns tips by scaling edges proportional to clade size; `1` sets all edges to unit length.
`thin, fat`	Numeric specifying width to plot edges if the rogue tip never / sometimes does attach to them.
`outgroupTips`	Vector of type character, integer or logical, specifying the names or indices of the tips to include in the outgroup. If `outgroupTips` is a of type character, and a tree contains multiple tips with a matching label, the first will be used.
`...`	Additional parameters to `plot.phylo()`.

Details

Rogue taxa can be identified using the package **Rogue** (Smith 2022).

Value

`RoguePlot()` returns a list whose elements are:

- `cons`: The reduced consensus tree, in preorder;
- `onEdge`: a vector of integers specifying the number of trees in `trees` in which the rogue leaf is attached to each edge in turn of the consensus tree;
- `atNode`: a vector of integers specifying the number of trees in `trees` in which the rogue leaf is attached to an edge collapsed into each node of the consensus tree.

Author(s)

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

References

Klopfstein S, Spasojevic T (2019). "Illustrating phylogenetic placement of fossils using RoguePlots: An example from ichneumonid parasitoid wasps (Hymenoptera, Ichneumonidae) and an extensive morphological matrix." *PLOS ONE*, **14**(4), e0212942.

```
\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1371/journal.pone.0212942")}
```

.

Smith MR (2022). "Using information theory to detect rogue taxa and improve consensus trees." *Systematic Biology*, **Online ahead of print**, syab099.

```
\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1093/sysbio/syab099")}
```

.

See Also

Other consensus tree functions: `ConsensusWithout()`, `Consensus()`

Examples

```
trees <- list(read.tree(text = '(a, (b, (c, (rogue, (d, (e, f))))));'),
              read.tree(text = '(a, (b, (c, (rogue, (d, (e, f))))));'),
              read.tree(text = '(a, (b, (c, (rogue, (d, (e, f))))));'),
              read.tree(text = '(a, (b, (c, (rogue, (d, (e, f))))));'),
              read.tree(text = '(rogue, (a, (b, (c, (d, (e, f))))));'),
              read.tree(text = '((rogue, a), (b, (c, (d, (e, f)))));'),
              read.tree(text = '(a, (b, ((c, d), (rogue, (e, f)))));'),
              read.tree(text = '(a, (b, ((c, (rogue, d)), (e, f))));'),
              read.tree(text = '(a, (b, (c, (d, (rogue, (e, f))))));'))
RoguePlot(trees, 'rogue')
```

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 ape::unroot() on trees in preorder.

Usage

```
RootTree(tree, outgroupTips)

RootOnNode(tree, node, resolveRoot = FALSE)

UnrootTree(tree)
```

Arguments

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

Value

RootTree() returns a tree of class phylo, rooted on the smallest clade that contains the specified tips, with edges and nodes numbered in preorder.

RootOnNode() returns a tree of class phylo, rooted on the requested node and ordered in Preorder.

UnrootTree() returns tree, in preorder, having collapsed the first child of the root node in each tree.

Author(s)

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

See Also

- ape::root()
- EnforceOutgroup()

Other tree manipulation: AddTip(), CollapseNode(), ConsensusWithout(), DropTip(), EnforceOutgroup(), ImposeConstraint(), KeptPaths(), KeptVerts(), 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 `FUN` returns a vector of `integer64` objects. `vapply()`, `sapply()` or `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

```
sapply64(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)

vapply64(X, FUN, FUN.LEN = 1, ...)

replicate64(n, expr, simplify = "array")
```

Arguments

`X`	a vector (atomic or list) or an `expression` object. Other objects (including classed objects) will be coerced by `base::as.list`.
`FUN`	the function to be applied to each element of `X`: see 'Details'. In the case of functions like `+`, `%*%`, the function name must be backquoted or quoted.
`...`	optional arguments to `FUN`.
`simplify`	logical or character string; should the result be simplified to a vector, matrix or higher dimensional array if possible? For `sapply` it must be named and not abbreviated. The default value, `TRUE`, returns a vector or matrix if appropriate, whereas if `simplify = "array"` the result may be an `array` of "rank" (=`length(dim(.))`) one higher than the result of `FUN(X[[i]])`.
`USE.NAMES`	logical; if `TRUE` and if `X` is character, use `X` as `names` for the result unless it had names already. Since this argument follows `...` its name cannot be abbreviated.
`FUN.LEN`	Integer specifying the length of the output of `FUN`.
`n`	integer: the number of replications.
`expr`	the expression (a language object, usually a call) to evaluate repeatedly.

Details

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

Author(s)

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

See Also

bit64::integer64()

Examples

```
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

```
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(), KeptPaths(), KeptVerts(), LeafLabelInterchange(), MakeTreeBinary(), RenumberTips(), RenumberTree(), Renumber(), RootTree(), SortTree(), Subtree()

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

Examples

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

`sort.multiPhylo` *Sort a list of phylogenetic trees*

Description

Trees are sorted by their mixed base representation, treating their leaves in the order of their labels (i.e. alphabetically, if leaves are labelled with text).

Usage

```
## S3 method for class 'multiPhylo'
sort(x, decreasing = FALSE, na.last = NA, ...)

## S3 method for class 'phylo'
e1 == e2

## S3 method for class 'phylo'
e1 < e2

## S3 method for class 'phylo'
e1 > e2

## S3 method for class 'MixedBase'
e1 == e2

## S3 method for class 'MixedBase'
e1 < e2

## S3 method for class 'MixedBase'
e1 > e2
```

Arguments

`x, decreasing, na.last, ...`	As in `sort()`.
`e1, e2`	Objects to be compared.

Author(s)

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

Examples

```
sort(as.phylo(5:0, 7))
```

SortTree *Sort tree*

Description

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

Usage

```
SortTree(tree, how = "cladesize", order = TipLabels(tree))

## S3 method for class 'phylo'
SortTree(tree, how = "cladesize", order = TipLabels(tree))

## S3 method for class 'list'
SortTree(tree, how = "cladesize", order = TipLabels(tree[[1]]))

## S3 method for class 'multiPhylo'
SortTree(tree, how = "cladesize", order = TipLabels(tree[[1]]))
```

Arguments

tree	One or more trees of class phylo, optionally as a list or a multiPhylo object.
how	Character vector specifying sort method: "Cladesize" rotates each node such that the larger clade is first, thus appearing lower when plotted; "TipLabels" rotates nodes such that labels listed sooner in order are listed first, and thus plot lower.
order	Character vector listing tip labels in sequence they should appear on tree. Clades containing a taxon earlier in this list will be listed sooner and thus plot lower on a tree. Taxa not listed in order will be treated as if they were last in the list.

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.

Value

SortTree() returns tree in the format of tree, with each node in each tree sorted

Author(s)

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

See Also

Preorder() also rearranges trees into a consistent shape, based on the index of leaves.

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

Examples

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

sorted <- SortTree(messyTree)
plot(sorted)
ape::nodelabels()
ape::edgelabels()
ape::tiplabels(adj = c(2, 1/3))

plot(SortTree(messyTree, how = "tip"))
```

SpectrumLegend *Produce a legend for continuous gradient scales*

Description

Prints an annotated vertical bar coloured according to a continuous palette.

Usage

```
SpectrumLegend(
  x0 = 0.05,
  y0 = 0.05,
  x1 = x0,
  y1 = y0 + 0.2,
  absolute = FALSE,
  legend = character(0),
  palette,
  lwd = 4,
  lty = 1,
  lend = "square",
  pos = 4,
  ...
)
```

Arguments

x0, y0, x1, y1	Coordinates of the bottom-left and top-right end of the bar.
absolute	Logical specifying whether x and y values denote coordinates (TRUE) or relative position, where (0, 0) denotes the bottom-left of the plot area and (1, 1) the top right.
legend	Character vector with which to label points on palette.

palette	Colour palette to depict.
lwd, lty, lend	Additional parameters to segments(), controlling line style.
pos, ...	Additional parameters to text().

Details

This convenience function is not yet very customizable; do file a GitHub issue if you would value additional functionality.

Author(s)

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

Examples

```
plot(0:1, 0:1, type = "n", frame.plot = FALSE,
     xlab = "x", ylab = "y")
SpectrumLegend(legend = c("Dark", "Middle", "Bright"),
               palette = hcl.colors(32L), lwd = 5)
SpectrumLegend(0.4, 0.95, 0.9, 0.95, abs = TRUE,
               legend = seq(1, 9, by = 2), palette = 1:9, pos = 1)
```

SplitFrequency *Frequency of splits*

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

```
SplitFrequency(reference, forest)

SplitNumber(tips, tree, tipIndex, powersOf2)

ForestSplits(forest, powersOf2)

TreeSplits(tree)
```

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.Splits`, `xor()`

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 = '%',
            col = SupportColor(splitFreqs / 100),
            frame = 'none', pos = 3L)
```

SplitInformation — *Phylogenetic information content of splitting leaves into two partitions*

Description

Calculate the phylogenetic information content (*sensu* Steel and 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) ~ 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) ~ 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

Steel MA, Penny D (2006). "Maximum parsimony and the phylogenetic information in multistate characters." In Albert VA (ed.), *Parsimony, Phylogeny, and Genomics*, 163–178. Oxford University Press, Oxford.

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

```
# 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` *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 = attr(x, "tip.label"), ...)
```

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.Splits, xor()

Examples

```
splits <- as.Splits(BalancedTree(letters[1:6]))
summary(splits)
TipsInSplits(splits)
summary(!splits)
TipsInSplits(!splits)

length(splits + !splits)
length(unique(splits + !splits))

summary(c(splits[[2:3]], !splits[[1:2]]))

moreSplits <- as.Splits(PectinateTree(letters[6:1]), tipLabel = splits)
print(moreSplits, details = TRUE)
match(splits, moreSplits)
moreSplits %in% splits
```

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)

## 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(), PathLengths(), TipLabels(), TreeIsRooted()

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

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)
```

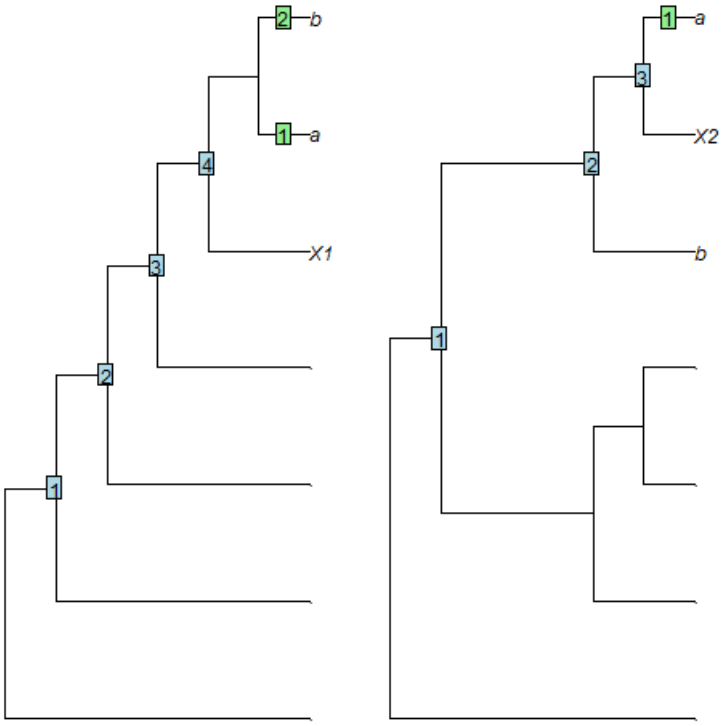
Arguments

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 (Asher and Smith 2022). 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, as proposed by Asher and Smith (2022). In the examples above, *X1* has a sister size of 2 leaves, whereas *X2*, 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

Asher R, Smith MR (2022). "Phylogenetic signal and bias in paleontology." *Systematic Biology*, **Online ahead of print**, syab072.

`\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1093/sysbio/syab072")}`

.

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(
  phy,
  parentheses = "{",
  collapse = "",
  ps = "",
```

```
  useIndex = TRUE,
  byTaxon = TRUE,
  concatenate = TRUE
)
```

Arguments

string	String of tokens, optionally containing whitespace, with no terminating semi-colon.
tips	(Optional) Character vector corresponding to the names (in order) of each taxon in the matrix, or an object 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()

Examples

```
StringToPhyDat("-?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

```
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

KeepTip() is a less flexible but faster equivalent.

Other split manipulation functions: DropTip(), TrivialSplits()

Examples

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

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

```
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(), KeptPaths(), KeptVerts(), LeafLabelInterchange(), MakeTreeBinary(), RenumberTips(), RenumberTree(), Renumber(), RootTree(), SingleTaxonTree(), SortTree()

Examples

```
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

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

SupportColor(
  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 outwith the valid range.

See Also

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

Examples

```
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

```
TipLabels(x, single = TRUE)

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

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

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

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

## S3 method for class 'MixedBase'
TipLabels(x, single = TRUE)

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

## S3 method for class 'Splits'
TipLabels(x, single = TRUE)

## S3 method for class 'list'
TipLabels(x, single = FALSE)

AllTipLabels(x)

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

## S3 method for class 'multiPhylo'
AllTipLabels(x)
```

```
## S3 method for class 'phylo'
AllTipLabels(x)

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

## S3 method for class 'TreeNumber'
AllTipLabels(x)

## S3 method for class 'matrix'
AllTipLabels(x)

## S3 method for class 'multiPhylo'
TipLabels(x, single = FALSE)

## S3 method for class 'character'
TipLabels(x, single = TRUE)

## S3 method for class 'numeric'
TipLabels(x, single = TRUE)

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

## Default S3 method:
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(), PathLengths(), SplitsInBinaryTree(), TreeIsRooted()

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

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, smallest = FALSE, ...)

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

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

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.
smallest	Logical; if TRUE, return the number of leaves in the smaller bipartition.
...	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.

See Also

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

Examples

```
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)
```

TotalCopheneticIndex *Total Cophenetic Index*

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 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

Mir A, Rosselló F, Rotger LA (2013). "A new balance index for phylogenetic trees." *Mathematical Biosciences*, **241**(1), 125–136.

```
\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1016/j.mbs.2012.10.005")}
```

.

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()`, `PathLengths()`, `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, based on a concept by John Tromp, employed in (Li et al. 1996).

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 'TreeNumber'
as.TreeNumber(x, ...)

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

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

## S3 method for class 'integer64'
as.MixedBase(x, tipLabels = NULL, ...)

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

## 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"), ...)

as.MixedBase(x, ...)

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

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

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

## S3 method for class 'MixedBase'
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 `NUnrooted(n)` unrooted trees with n leaves. As such, each n-leaf tree can be uniquely identified by a non-negative integer x < `NUnrooted(n)`.

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

We add leaf 2 by breaking an edge and inserting a node (numbered `2 + nTip -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.

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.label`.

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

Author(s)

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

References

Li M, Tromp J, Zhang L (1996). "Some notes on the nearest neighbour interchange distance." In Goos G, Hartmanis J, Leeuwen J, Cai J, Wong CK (eds.), *Computing and Combinatorics*, volume 1090, 343–351. Springer, Berlin, Heidelberg. ISBN 978-3-540-61332-9,

`\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1007/3-540-61332-3_168")}`

.

See Also

Describe the shape of a tree topology, independent of leaf labels: `TreeShape()`

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

Examples

```
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.
treeNumber <- as.TreeNumber("1234567890123", nTip = 14)
tree <- as.phylo(treeNumber)
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

```
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

```
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).

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: DropTip(), 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 theorem two 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

Carter M, Hendy M, Penny D, Székely LA, Wormald NC (1990). "On the distribution of lengths of evolutionary trees." *SIAM Journal on Discrete Mathematics*, **3**(1), 38–47.

`\Sexpr[results=rd,stage=build]{tools:::Rd_expr_doi("10.1137/0403005")}`

.

See Also

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

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

```
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.
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')
```

---

xor *Exclusive OR operation*

Description

Exclusive OR operation

Usage

```
xor(x, y)

## S4 method for signature 'Splits,Splits'
xor(x, y)
```

Arguments

x, y	Objects to be compared.

See Also

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

Index

∗ **C++ wrappers**
edge_to_splits, 24
∗ **Splits operations**
LabelSplits, 32
match.Splits, 37
NSplits, 51
NTip, 52
PolarizeSplits, 55
SplitFrequency, 72
Splits, 76
SplitsInBinaryTree, 78
TipLabels, 86
TipsInSplits, 88
xor, 101
∗ **consensus tree functions**
Consensus, 15
ConsensusWithout, 16
RoguePlot, 63
∗ **datasets**
brewer, 10
doubleFactorials, 20
Lobo.data, 35
logDoubleFactorials, 36
nRootedShapes, 50
∗ **double factorials**
DoubleFactorial, 19
doubleFactorials, 20
logDoubleFactorials, 36
∗ **methods**
match.Splits, 37
∗ **pairwise tree distances**
PairwiseDistances, 53
∗ **phylogenetic matrix conversion functions**
MatrixToPhyDat, 38
StringToPhyDat, 81
∗ **split information functions**
CharacterInformation, 11
SplitInformation, 73
SplitMatchProbability, 75
TreesMatchingSplit, 95
UnrootedTreesMatchingSplit, 98
∗ **split information function**
NRooted, 48
∗ **split manipulation functions**
DropTip, 21
Subsplit, 83
TrivialSplits, 97
∗ **string parsing functions**
EndSentence, 25
MorphoBankDecode, 40
RightmostCharacter, 62
Unquote, 98
∗ **tree characterization functions**
CladisticInfo, 12
Consensus, 15
Stemwardness, 79
TotalCopheneticIndex, 89
∗ **tree generation functions**
ConstrainedNJ, 17
GenerateTree, 27
NJTree, 45
SingleTaxonTree, 68
TreeNumber, 91
∗ **tree information functions**
CladisticInfo, 12
NRooted, 48
TreesMatchingTree, 96
∗ **tree manipulation**
AddTip, 4
CollapseNode, 14
ConsensusWithout, 16
DropTip, 21
EnforceOutgroup, 26
ImposeConstraint, 29
KeptPaths, 30
KeptVerts, 31
LeafLabelInterchange, 33
MakeTreeBinary, 37
Renumber, 60
RenumberTips, 61
RootTree, 66
SingleTaxonTree, 68
SortTree, 70
Subtree, 84
∗ **tree navigation**
CladeSizes, 12

DescendantEdges, 18
EdgeAncestry, 22
EdgeDistances, 23
ListAncestors, 34
MRCA, 40
NDescendants, 43
NodeDepth, 45
NodeOrder, 46
RootNode, 65
∗ **tree properties**
ConsensusWithout, 16
NSplits, 51
NTip, 52
PathLengths, 54
SplitsInBinaryTree, 78
TipLabels, 86
TreeIsRooted, 91
∗ **tree**
AddTip, 4
SingleTaxonTree, 68
<.MixedBase *(sort.multiPhylo)*, 69
<.phylo *(sort.multiPhylo)*, 69
==.MixedBase *(sort.multiPhylo)*, 69
==.phylo *(sort.multiPhylo)*, 69
>.MixedBase *(sort.multiPhylo)*, 69
>.phylo *(sort.multiPhylo)*, 69
%in%,Splits,Splits-method *(match.Splits)*, 37

AddTip, 4, *15*, *17*, *22*, *26*, *30*, *31*, *34*, *37*, *61*, *62*, *66*, *68*, *71*, *84*
AddTip(), *24*, *32*
AddTipEverywhere *(AddTip)*, 4
AddUnconstrained *(ImposeConstraint)*, 29
AllAncestors *(ListAncestors)*, 34
AllDescendantEdges *(DescendantEdges)*, 18
AllTipLabels *(TipLabels)*, 86
AncestorEdge, *12*, *19*, *23*, *24*, *35*, *41*, *44*, *46*, *47*, *65*
ape::consensus(), *17*
ape::drop.tip(), *21*
ape::edgelabels(), *32*
ape::mst(), *42*
ape::multi2di(), *37*
ape::node.depth, *46*
ape::Ntip(), *52*
ape::root(), *66*
ape::unroot, *66*
ape::write.tree(), *10*
ApeTime, 6
array, *67*
ArtEx *(ArtificialExtinction)*, 6
ArtificialExtinction, 6
as.list, *67*
as.logical.Splits *(Splits)*, 76
as.MixedBase *(TreeNumber)*, 91
as.multiPhylo, 8
as.Newick, 9, *44*
as.phylo.MixedBase *(TreeNumber)*, 91
as.phylo.numeric *(TreeNumber)*, 91
as.phylo.TreeNumber *(TreeNumber)*, 91
as.Splits *(Splits)*, 76
as.Splits(), *25*, *32*
as.TreeNumber *(TreeNumber)*, 91

BalancedTree *(GenerateTree)*, 27
base::lapply(), *67*
bind.tree, *5*
bit64::integer64(), *68*
brewer, 10

c(), *100*
CharacterInformation, 11, *75*, *76*, *95*, *99*
CladeSizes, 12, *19*, *23*, *24*, *35*, *41*, *44*, *46*, *47*, *65*
Cladewise, *23*, *60*, *84*
CladisticInfo, 12, *16*, *50*, *80*, *90*, *96*
CladisticInformation *(CladisticInfo)*, 12
CollapseEdge *(CollapseNode)*, 14
CollapseNode, *5*, 14, *17*, *22*, *26*, *30*, *31*, *34*, *37*, *61*, *62*, *66*, *68*, *71*, *84*
connection, *56*
Consensus, *14*, 15, *17*, *64*, *80*, *90*
Consensus(), *63*
ConsensusWithout, *5*, *15*, *16*, 16, *22*, *26*, *30*, *31*, *34*, *37*, *52–54*, *61*, *62*, *64*, *66*, *68*, *71*, *78*, *84*, *87*, *91*
ConstrainedNJ, 17, *27*, *45*, *68*, *94*

DescendantEdges, *12*, 18, *23*, *24*, *35*, *41*, *44*, *46*, *47*, *65*
DoubleFactorial, 19, *20*, *36*
DoubleFactorial64 *(DoubleFactorial)*, 19
doubleFactorials, *20*, 20, *36*
DropTip, *5*, *15*, *17*, 21, *26*, *30*, *31*, *34*, *37*, *61*, *62*, *66*, *68*, *71*, *83*, *84*, *97*
DropTip(), *63*
DropTipPhylo *(DropTip)*, 21

edge_to_splits, 24
EdgeAncestry, *12*, *19*, 22, *24*, *35*, *41*, *44*, *46*, *47*, *65*
EdgeDistances, *12*, *19*, *23*, 23, *35*, *41*, *44*, *46*, *47*, *65*
edgelabels, *14*, *19*
EndSentence, 25, *40*, *62*, *98*

EnforceOutgroup, *5*, *15*, *17*, *22*, 26, *30*, *31*, *34*, *37*, *61*, *62*, *66*, *68*, *71*, *84*
EnforceOutgroup(), *66*
expression, *67*
extract.clade, *84*

ForestSplits (SplitFrequency), 72

GenerateTree, *18*, 27, *45*, *68*, *94*

Hamming, 28
Hamming(), *18*, *45*

IC1Spr (N1Spr), 42
ImposeConstraint, *5*, *15*, *17*, *22*, *26*, 29, *31*, *34*, *37*, *61*, *62*, *66*, *68*, *71*, *84*
in.Splits (match.Splits), 37

KeepTip (DropTip), 21
KeepTip(), *83*
KeepTipPostorder (DropTip), 21
KeepTipPreorder (DropTip), 21
KeptPaths, *5*, *15*, *17*, *22*, *26*, *30*, 30, *31*, *34*, *37*, *61*, *62*, *66*, *68*, *71*, *84*
KeptVerts, *5*, *15*, *17*, *22*, *26*, *30*, *31*, 31, *34*, *37*, *61*, *62*, *66*, *68*, *71*, *84*
KeptVerts(), *30*

LabelSplits, 32, *38*, *52*, *53*, *55*, *73*, *77*, *79*, *87*, *89*, *102*
LabelSplits(), *85*
language object, *67*
lapply(), *67*
LeafLabelInterchange, *5*, *15*, *17*, *22*, *26*, *30*, *31*, 33, *37*, *61*, *62*, *66*, *68*, *71*, *84*
legend(), *17*
ListAncestors, *12*, *19*, *23*, *24*, 34, *41*, *44*, *46*, *47*, *65*
ListAncestors(), *40*
LnDoubleFactorial (DoubleFactorial), 19
LnRooted (NRooted), 48
LnSplitMatchProbability (SplitMatchProbability), 75
LnTreesMatchingSplit (TreesMatchingSplit), 95
LnTreesMatchingTree (TreesMatchingTree), 96
LnUnrooted (NRooted), 48
LnUnrootedMult (NRooted), 48
LnUnrootedSplits (NRooted), 48
LnUnrootedTreesMatchingSplit (UnrootedTreesMatchingSplit), 98
Lobo.data, 35
Lobo.phy (Lobo.data), 35
Log2DoubleFactorial (DoubleFactorial), 19
Log2Rooted (NRooted), 48
Log2TreesMatchingSplit (TreesMatchingSplit), 95
Log2TreesMatchingTree (TreesMatchingTree), 96
Log2Unrooted (NRooted), 48
Log2UnrootedMult (NRooted), 48
Log2UnrootedSplits (NRooted), 48
Log2UnrootedTreesMatchingSplit (UnrootedTreesMatchingSplit), 98
LogDoubleFactorial (DoubleFactorial), 19
logDoubleFactorials, *20*, 36

MakeTreeBinary, *5*, *15*, *17*, *22*, *26*, *30*, *31*, *34*, 37, *61*, *62*, *66*, *68*, *71*, *84*
MarkMissing (ConsensusWithout), 16
match (match.Splits), 37
match(), *38*
match,Splits,Splits-method (match.Splits), 37
match.Splits, *32*, 37, *52*, *53*, *55*, *73*, *77*, *79*, *87*, *89*, *102*
MatrixToPhyDat, 38, *82*
MatrixToPhyDat(), *57*, *58*
mixed base representation, *69*
MorphoBankDecode, *25*, 40, *62*, *98*
MRCA, *12*, *19*, *23*, *24*, *35*, 40, *44*, *46*, *47*, *65*
MSTEdges, 41
MSTLength (MSTEdges), 41
multiPhylo, *99*
MultiSplitInformation (SplitInformation), 73

N1Spr, 42
names, *67*
NDescendants, *12*, *19*, *23*, *24*, *35*, *41*, 43, *46*, *47*, *65*
NewickTree, 44
NewickTree(), *10*
NJTree, *18*, *27*, 45, *68*, *94*
NJTree(), *29*
NodeDepth, *12*, *19*, *23*, *24*, *35*, *41*, *44*, 45, *47*, *65*
nodelabels, *14*
NodeOrder, *12*, *19*, *23*, *24*, *35*, *41*, *44*, *46*, 46, *65*
NonDuplicateRoot, *12*, *19*, *23*, *24*, *35*, *41*, *44*, *46*, *47*, *65*
NPartitionPairs, 47

NPartitions (NSplits), 51
NRooted, *14*, 48, *96*
NRooted64 (NRooted), 48
nRootedShapes, 50
NSplits, *17*, *32*, *38*, 51, *53–55*, *73*, *77–79*, *87*, *89*, *91*, *102*
NTip, *17*, *32*, *38*, *52*, 52, *54*, *55*, *73*, *77–79*, *87*, *89*, *91*, *102*
NUnrooted (NRooted), 48
NUnrooted64 (NRooted), 48
NUnrootedMult (NRooted), 48
nUnrootedShapes (nRootedShapes), 50
NUnrootedSplits (NRooted), 48

PairwiseDistances, 53
PathLengths, *17*, *52*, *53*, 54, *78*, *87*, *91*
PathLengths(), *30*
PectinateTree (GenerateTree), 27
PhyDat (ReadCharacters), 56
PhyDatToMatrix (MatrixToPhyDat), 38
PhyDatToString (StringToPhyDat), 81
PhydatToString (StringToPhyDat), 81
phylo, *4*, *12*, *14*, *19*, *21*, *23*, *29*, *32–34*, *37*, *43*, *44*, *60*, *61*, *66*, *72*, *77*, *79*, *84*, *96*, *99*
PhylogeneticInfo (CladisticInfo), 12
PhylogeneticInformation (CladisticInfo), 12
PhyToString (StringToPhyDat), 81
PolarizeSplits, *32*, *38*, *52*, *53*, 55, *73*, *77*, *79*, *87*, *89*, *102*
Preorder, *21*, *23*, *31*, *34*, *54*, *66*, *84*
Preorder(), *21*
print.TreeNumber, 55

RandomTree (GenerateTree), 27
read.nexus.data, *56*
read.table(), *57*
ReadAsPhyDat (ReadCharacters), 56
ReadCharacters, 56
ReadCharacters(), *39*
ReadNotes (ReadCharacters), 56
ReadTntAsPhyDat (ReadCharacters), 56
ReadTntCharacters (ReadCharacters), 56
ReadTntCharacters(), *39*, *101*
ReadTntTree, 58
Renumber, *5*, *15*, *17*, *22*, *26*, *30*, *31*, *34*, *37*, 60, *62*, *66*, *68*, *71*, *84*
RenumberTips, *5*, *15*, *17*, *22*, *26*, *30*, *31*, *34*, *37*, *61*, 61, *66*, *68*, *71*, *84*
RenumberTips(), *9*, *10*
RenumberTree, *5*, *15*, *17*, *22*, *26*, *30*, *31*, *34*, *37*, *61*, *62*, *66*, *68*, *71*, *84*
replicate64 (sapply64), 67
RightmostCharacter, *25*, *40*, 62, *98*
RoguePlot, *16*, *17*, 63
RootNode, *12*, *19*, *23*, *24*, *35*, *41*, *44*, *46*, *47*, 65
RootNodeDist (Stemwardness), 79
RootNodeDistance (Stemwardness), 79
RootOnNode (RootTree), 66
RootTree, *5*, *15*, *17*, *22*, *26*, *30*, *31*, *34*, *37*, *61*, *62*, 66, *68*, *71*, *84*
RootTree(), *26*
rtree, *87*

sapply64, 67
segments(), *72*
SingleTaxonTree, *5*, *15*, *17*, *18*, *22*, *26*, *27*, *30*, *31*, *34*, *37*, *45*, *61*, *62*, *66*, 68, *71*, *84*, *94*
SisterSize (Stemwardness), 79
sort(), *69*
sort.multiPhylo, 69
SortTree, *5*, *15*, *17*, *22*, *26*, *30*, *31*, *34*, *37*, *61*, *62*, *66*, *68*, 70, *84*
SpectrumLegend, 71
SplitFrequency, *32*, *38*, *52*, *53*, *55*, 72, *77*, *79*, *87*, *89*, *102*
SplitFrequency(), *32*, *85*
SplitImbalance (TipsInSplits), 88
SplitInformation, *11*, 73, *76*, *95*, *99*
SplitMatchProbability, *11*, *75*, 75, *95*, *99*
SplitNumber (SplitFrequency), 72
Splits, *32*, *38*, *52*, *53*, *55*, *73*, 76, *79*, *83*, *87*, *89*, *97*, *102*
SplitsInBinaryTree, *17*, *32*, *38*, *52–55*, *73*, *77*, 78, *87*, *89*, *91*, *102*
StarTree (GenerateTree), 27
Stemwardness, *14*, *16*, 79, *90*
StringToPhyDat, *39*, 81
StringToPhydat (StringToPhyDat), 81
Subsplit, *22*, 83, *97*
Subtree, *5*, *15*, *17*, *22*, *26*, *30*, *31*, *34*, *37*, *61*, *62*, *66*, *68*, *71*, 84
SupportColor (SupportColour), 85
SupportColour, 85
SupportColour(), *32*

TCIContext (TotalCopheneticIndex), 89
text(), *72*
TipLabels, *17*, *32*, *38*, *52–55*, *73*, *77–79*, 86, *89*, *91*, *102*
TipLabels(), *27*, *93*
TipsInSplits, *32*, *38*, *52*, *53*, *55*, *73*, *77*, *79*, *87*, 88, *102*
TNTText2Tree (ReadTntTree), 58
TntText2Tree (ReadTntTree), 58

TotalCopheneticIndex, *14*, *16*, *80*, 89
TreeIsRooted, *17*, *52–54*, *78*, *87*, 91
TreeIsRooted(), *65*
TreeNumber, *18*, *27*, *45*, *68*, 91
TreeShape(), *94*
TreesMatchingSplit, *11*, *75, 76*, 95, *99*
TreesMatchingTree, *14*, *50*, 96
TreeSplits *(*SplitFrequency*)*, 72
TrivialSplits, *22*, *83*, 97

Unquote, *25*, *40*, *62*, 98
UnrootedTreesMatchingSplit, *11*, *75, 76*, *95*, 98
UnrootTree *(*RootTree*)*, 66
UnshiftTree, 99

vapply64 *(*sapply64*)*, 67

WithoutTrivialSplits *(*TrivialSplits*)*, 97
write.tree(), *44*
WriteTntCharacters, 100
WriteTntCharacters(), *58*

xor, *32*, *38*, *52, 53*, *55*, *73*, *77*, *79*, *87*, *89*, 101
xor,Splits,Splits-method *(*xor*)*, 101