Package 'TreeNode'

May 26, 2020

Type Package

Title Read and Write Trees in Binary and NWKA Formats

Version 1.0.0

Author Giorgio Bianchini

Maintainer Giorgio Bianchini <giorgio.bianchini@bristol.ac.uk>

Description The TreeNode package provides functions to read and write files containing phylogenetic trees in Binary Tree format and in Newick-with-Attributes (NWKA) format.
TreeNode produces and consumes trees stored in the same phylo or multiPhylo objects used by the package ape.
More information on TreeNode can be found at https://github.com/arklumpus/TreeNode .

License GPL-3

Encoding UTF-8

LazyData true

Imports Rcpp (>= 1.0.4.6)

LinkingTo Rcpp

SystemRequirements C++17

RoxygenNote 7.1.0

Suggests ape

R topics documented:

`begin_writing_binary_trees` *Write Tree File Header in Binary Format*

Description

This function initializes a file that will be used to store trees in binary format.

Usage

```
begin_writing_binary_trees(file)
```

Arguments

`file`	A file name.

Details

This function will create an empty header for the binary format file (without writing any trees). Trees should written after the header using the `keep_writing_binary_trees` function. The file should be finalised using the `finish_writing_binary_trees` function.

Note that, since node names are not stored in the header, files produced with this workflow may be much larger than files produced using the `write_binary_trees` function (e.g. if the file contains many trees which all have the same tip labels). The advantage of this approach is that the trees do not need to be all available/stored in memory at the same time.

The vector that is returned by this function contains the position at which the first tree will be appended in the file. This vector should be provided to subsequent calls to `keep_writing_binary_trees`.

Value

A vector of mode integer, which should be used to keep track of the addresses of trees that will be added to the file.

Author(s)

Giorgio Bianchini

References

https://github.com/arklumpus/TreeNode/blob/master/BinaryTree.md

See Also

`keep_writing_binary_trees`, `finish_writing_binary_trees`, `ape`, `write.tree`

Examples

```
#A simple tree
tree1 <- ape::read.tree(text = "((A,B),(C,D));")

# Initialise the output file
addresses <- begin_writing_binary_trees("outputFile.tbi")

# Append a tree to the output file
addresses <- keep_writing_binary_trees(tree1, "outputFile.tbi", addresses)

# Some more trees (note that we are overwriting tree1)
tree1 <- ape::read.tree(text = "(((A,B),C),D);")
tree2 <- ape::read.tree(text = "((D,(A,B)),C);")

# Append them to the file
addresses <- keep_writing_binary_trees(tree1, "outputFile.tbi", addresses)
addresses <- keep_writing_binary_trees(tree2, "outputFile.tbi", addresses)

#Some raw data
raw_data <- as.raw(seq(1, 5))

# Finalise the output file
finish_writing_binary_trees("outputFile.tbi", addresses, raw_data)
```

`finish_writing_binary_trees` *Finalise Tree File in Binary Format*

Description

This function finalises a tree file in binary format.

Usage

```
finish_writing_binary_trees(
  file,
  addresses,
  additional_data = vector("raw", 0)
)
```

Arguments

`file`	A file name.
`addresses`	A vector of mode integer containing the addresses of previous trees that have been added to the file.
`additional_data`	A vector of mode raw containg additional binary data that will be included within the tree file.

Details

This function will finalise a tree file in binary format, by writing the file trailer containing the addresses of the trees stored in the file.

Note that, since node names are not stored in the header, files produced with this workflow may be much larger than files produced using the `write_binary_trees` function (e.g. if the file contains many trees which all have the same tip labels). The advantage of this approach is that the trees do not need to be all available/stored in memory at the same time.

Finalising the file is not *strictly* necessary, in the sense that a file with a missing or incomplete trailer can still be parsed. However, parsing such a file requires scanning through the whole file to determine tree addresses (which is not necessary if they are stored in a proper trailer).

The additional binary data (if any) will be written in the file after the trees and before the trailer.

Author(s)

Giorgio Bianchini

References

`https://github.com/arklumpus/TreeNode/blob/master/BinaryTree.md`

See Also

`begin_writing_binary_trees`, `keep_writing_binary_trees`, `ape`, `write.tree`

Examples

```
#A simple tree
tree1 <- ape::read.tree(text = "((A,B),(C,D));")

# Initialise the output file
addresses <- begin_writing_binary_trees("outputFile.tbi")

# Append a tree to the output file
addresses <- keep_writing_binary_trees(tree1, "outputFile.tbi", addresses)

# Some more trees (note that we are overwriting tree1)
tree1 <- ape::read.tree(text = "(((A,B),C),D);")
tree2 <- ape::read.tree(text = "((D,(A,B)),C);")

# Append them to the file
addresses <- keep_writing_binary_trees(tree1, "outputFile.tbi", addresses)
addresses <- keep_writing_binary_trees(tree2, "outputFile.tbi", addresses)

#Some raw data
raw_data <- as.raw(seq(1, 5))

# Finalise the output file
finish_writing_binary_trees("outputFile.tbi", addresses, raw_data)
```

keep_writing_binary_trees

Add Tree to File in Binary Format

Description

This function adds trees to a file in binary format.

Usage

```
keep_writing_binary_trees(trees, file, addresses)
```

Arguments

`trees`	An object of class `"phylo"` or `"multiPhylo"`.
`file`	A file name.
`addresses`	A vector of mode integer containing the addresses of previous trees that have been added to the file.

Details

This function will append trees in binary format to a file. The file should have been already initialised by the `begin_writing_binary_trees` function and may already contain some trees. It should be finalised using the `finish_writing_binary_trees` function.

Note that, since node names are not stored in the header, files produced with this workflow may be much larger than files produced using the `write_binary_trees` function (e.g. if the file contains many trees which all have the same tip labels). The advantage of this approach is that the trees do not need to be all available/stored in memory at the same time.

The vector that is returned by this function contains the position at which the trees have been appendend to the file, as well as the position at which the next tree will be appended. This vector should be provided to subsequent calls to `keep_writing_binary_trees` and to `finish_writing_binary_trees`.

Value

A vector of mode integer containing the addresses of the trees that have been added to the file, which should be used to keep track of the addresses of subsequent trees.

Author(s)

Giorgio Bianchini

References

`https://github.com/arklumpus/TreeNode/blob/master/BinaryTree.md`

See Also

`write_binary_trees`, `begin_writing_binary_trees`, `finish_writing_binary_trees`, `ape`, `write.tree`

Other functions to write trees: `write_binary_trees()`, `write_nwka_nexus()`, `write_nwka_tree()`

Examples

```
#A simple tree
tree1 <- ape::read.tree(text = "((A,B),(C,D));")

# Initialise the output file
addresses <- begin_writing_binary_trees("outputFile.tbi")

# Append a tree to the output file
addresses <- keep_writing_binary_trees(tree1, "outputFile.tbi", addresses)

# Some more trees (note that we are overwriting tree1)
tree1 <- ape::read.tree(text = "(((A,B),C),D);")
tree2 <- ape::read.tree(text = "((D,(A,B)),C);")

# Append them to the file
addresses <- keep_writing_binary_trees(tree1, "outputFile.tbi", addresses)
addresses <- keep_writing_binary_trees(tree2, "outputFile.tbi", addresses)

#Some raw data
raw_data <- as.raw(seq(1, 5))

# Finalise the output file
finish_writing_binary_trees("outputFile.tbi", addresses, raw_data)
```

`read_binary_trees` *Read Tree File in Binary Format*

Description

This function reads a file containing one or more trees in binary format.

Usage

```
read_binary_trees(file, tree.names = NULL, keep.multi = FALSE)
```

Arguments

`file`	A file name.
`tree.names`	A vector of mode character containing names for the trees that are read from the file; if `NULL` (the default), the trees will be named according to the names in the tree file or, if these are missing, as `"tree1"`, `"tree2"`, ...
`keep.multi`	If `TRUE`, this function will return an object of class `"multiPhylo"` even when the tree file contains only a single tree. Defaults to `FALSE`, which means that if the file contains a single tree, an object of class `"phylo"` is returned.

Details

This function reads the whole file in memory at once. If you wish to process the file tree-by-tree, you should use the `read_one_binary_tree` function.

Node attributes (e.g. support values, rates, ages...) are parsed by this function and returned in the `tip.attributes` and `node.attributes` elements of the returned `"phylo"` objects.

Attribute names may appear in any kind of casing (e.g. `Name`, `name` or `NAME`), but they should be treated using case-insensitive comparisons.

If the file has an invalid trailer (e.g. because it is incomplete), the function will print a warning and attempt anyways to extract as many trees as possible.

Value

An object of class `"phylo"` or `"multiPhylo"`, compatible with the `ape` package.

In addition to the elements described in the documentation for the `read.tree` function of the `ape` package, a `"phylo"` object produced by this function will also have the following components:

`tip.attributes`	A named list of attributes for the tips of the tree. Each element of this list is a vector of mode character or numeric (depending on the attribute).
`node.attributes`	A named list of attributes for the internal nodes of the tree. Each element of this list is a vector of mode character or numeric (depending on the attribute).

Author(s)

Giorgio Bianchini

References

`https://github.com/arklumpus/TreeNode/blob/master/BinaryTree.md`

See Also

`read_one_binary_tree`, `ape`, `read.tree`

Other functions to read trees: `read_nwka_nexus()`, `read_nwka_tree()`, `read_one_binary_tree()`

Examples

```
# Tree file (replace with your own)
treeFile <- system.file("extdata", "oneTree.tbi", package="TreeNode")

# Read the tree file
tree <- read_binary_trees(treeFile)

# Use support values as node labels
tree$node.label = tree$node.attributes$Support

# Plot the tree with support values at the nodes
ape::plot.phylo(tree, show.node.label = TRUE)
```

`read_binary_tree_metadata` *Read Tree Metadata in Binary Format*

Description

This function reads the metadata from a file containing trees in binary format.

Usage

```
read_binary_tree_metadata(file, invalid_trailer = c("scan", "fail", "ignore"))
```

Arguments

`file`	A file name.
`invalid_trailer`	If this is set to `"scan"` (the default), if the tree file has an invalid trailer, the function will print a warning and then read the whole file, attempting to parse as many trees as possible and storing the addresses of those trees. If this is set to `"fail"`, an error will be generated if the tree file has an invalid trailer. If this is set to `"ignore"` and the tree file has an invalid trailer, a warning will be printed and the returned object will be missing the `TreeAddresses` element.

Details

This function reads the metadata information from the header and trailer of a file containing trees in binary format. This information consists in the addresses of the trees (i.e. byte offsets at which the data stream describing each tree starts) and in any names or attributes that are stored in the tree header.

If there are such names or attributes in the header, it *usually* means that every tree in the file should have the same names and attributes. However, this is not required by the file format; some (or all) of the trees in the file may have additional/missing taxa, or additional/missing attributes.

If the file's trailer is invalid (e.g. because the file is incomplete), the default behaviour is to read the whole file, attempting to parse as many trees as possible. The trees themselves are discarded, while their addresses are stored. This is desirable when the concern preventing all the trees in the file from being read at once (i.e., the use of `read_binary_trees`) is memory. If this is not the case, changing the value of `invalid_trailer` provides alternative ways to deal with this situation, either by generating an error, or by returning a valid object which is however missing the `TreeAddresses` attribute.

Due to limitations with R's integral types, this function may have issues with files larger than 2GB.

Value

An object of class `"BinaryTreeMetadata"` with the following components:

`GlobalNames`	A logical value indicating whether the tree file contains a list of names in the header.
`Names`	Only present if `GlobalNames` is `TRUE`. A vector of mode character containing the names specified in the file header.

`GlobalAttributes`	A logical value indicating whether the tree file contains a list of attributes in the header.
`Attributes`	Only present if `GlobalAttributes` is `TRUE`. A list of attributes. Each attribute is itself a list of two elements: `AttributeName` is a character object describing the attribute's name (e.g. "Length"), and `IsNumeric` describes whether the attribute represents a numeric value (e.g. a branch's length) or not.
`TreeAddresses`	A vector of mode integer containing the addresses (i.e. byte offsets from the start of the file) of the trees. If `invalid_trailer` is `"ignore"` and the file has an invalid trailer, this element will be missing.

Author(s)

Giorgio Bianchini

References

https://github.com/arklumpus/TreeNode/blob/master/BinaryTree.md

See Also

`read_binary_trees`, `read_one_binary_tree`, `ape`, `read.tree`

Examples

```
# Tree file (replace with your own)
treeFile <- system.file("extdata", "manyTrees.tbi", package="TreeNode")

# Read the binary tree metadata
meta <- read_binary_tree_metadata(treeFile)

#Print a list of the names defined in the file's header
meta$Names

#Print a list of the attributes defined in the file's header
meta$Attributes
```

`read_nwka_nexus` *Read Tree File in NEXUS Format with NWKA Trees*

Description

This function reads a file containing one or more trees in NEXUS format. Each tree is parsed according to the Newick-with-Attributes (NWKA) format.

Usage

```
read_nwka_nexus(file, tree.names = NULL, force.multi = FALSE, debug = FALSE)
```

Arguments

`file`	A file name.
`tree.names`	A vector of mode character containing names for the trees that are read from the file; if `NULL` (the default), the trees will be named according to the names in the tree file or, if these are missing, as `"tree1"`, `"tree2"`, ...
`force.multi`	If `TRUE`, this function will return an object of class `"multiPhylo"` even when the tree file contains only a single tree. Defaults to `FALSE`, which means that if the file contains a single tree, an object of class `"phylo"` is returned.
`debug`	A logical value indicating whether to enable verbose debug output while parsing the tree. If this is `TRUE`, the function will print information about each node in the each tree as it parses it.

Details

Only the `Trees` block of the NEXUS file is parsed.

Node attributes (e.g. support values, rates, ages...) are parsed by this function and returned in the `tip.attributes` and `node.attributes` elements of the returned `"phylo"` objects. If the nodes contain a `prob` attribute, its value will also be copied to the `Support` attribute.

The translation table (if any) of the `Trees` block is used to translate the names of both tips and internal nodes. However, if the untranslated names of internal nodes are numbers, these may be interpreted as support values (and thus, not translated).

Attribute names may appear in any kind of casing (e.g. `Name`, `name` or `NAME`), but they should be treated using case-insensitive comparisons.

Setting the debug argument to `TRUE` can be useful when analysing malformed trees (to understand at which point in the tree the problem lies).

Value

An object of class `"phylo"` or `"multiPhylo"`, compatible with the `ape` package.

In addition to the elements described in the documentation for the `read.tree` function of the `ape` package, a `"phylo"` object produced by this function will also have the following components:

`tip.attributes`	A named list of attributes for the tips of the tree. Each element of this list is a vector of mode character or numeric (depending on the attribute).
`node.attributes`	A named list of attributes for the internal nodes of the tree. Each element of this list is a vector of mode character or numeric (depending on the attribute).

Author(s)

Giorgio Bianchini

References

`https://github.com/arklumpus/TreeNode/blob/master/NWKA.md`

See Also

`ape`, `read.tree`, `read.nexus`

Other functions to read trees: `read_binary_trees()`, `read_nwka_tree()`, `read_one_binary_tree()`

read_nwka_tree *Read Tree File in NWKA Format*

Description

This function reads a file containing one or more trees in Newick-with-Attributes (NWKA) format.

Usage

```
read_nwka_tree(
  file = "",
  text = NULL,
  tree.names = NULL,
  keep.multi = FALSE,
  debug = FALSE
)
```

Arguments

`file`	A file name.
`text`	A variable of mode character containing the tree(s) to parse. By default, this is set to `NULL` and ignored (i.e. the tree is read from the file specified by the `file` argument); otherwise, the file argument is ignored and the trees are read from the `text` argument.
`tree.names`	A vector of mode character containing names for the trees that are read from the file; if `NULL` (the default), the trees will be named as `"tree1"`, `"tree2"`, ...
`keep.multi`	If `TRUE`, this function will return an object of class `"multiPhylo"` even when the tree file contains only a single tree. Defaults to `FALSE`, which means that if the file contains a single tree, an object of class `"phylo"` is returned.
`debug`	A logical value indicating whether to enable verbose debug output while parsing the tree. If this is `TRUE`, the function will print information about each node in the tree as it parses it.

Details

The Newick-with-Attributes format parsed by this function is backwards compatible with the Newick/New Hampshire format and some of its extensions (e.g. Extended Newick, New Hampshire X).

Node attributes (e.g. support values, rates, ages...) are parsed by this function and returned in the `tip.attributes` and `node.attributes` elements of the returned `"phylo"` objects. If the nodes contain a `prob` attribute, its value will also be copied to the `Support` attribute.

Attribute names may appear in any kind of casing (e.g. `Name`, `name` or `NAME`), but they should be treated using case-insensitive comparisons.

Setting the `debug` argument to `TRUE` can be useful when analysing malformed trees (to understand at which point in the tree the problem lies).

Value

An object of class "phylo" or "multiPhylo", compatible with the ape package.

In addition to the elements described in the documentation for the read.tree function of the ape package, a "phylo" object produced by this function will also have the following components:

tip.attributes	A named list of attributes for the tips of the tree. Each element of this list is a vector of mode character or numeric (depending on the attribute).
node.attributes	A named list of attributes for the internal nodes of the tree. Each element of this list is a vector of mode character or numeric (depending on the attribute).

Author(s)

Giorgio Bianchini

References

https://github.com/arklumpus/TreeNode/blob/master/NWKA.md

See Also

ape, read.tree

Other functions to read trees: read_binary_trees(), read_nwka_nexus(), read_one_binary_tree()

Examples

```
# Parse a tree string
# Topology from https://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi?id=207598
tree <- read_nwka_tree(text="(((('Homo sapiens'[rank=species])'Homo'[rank=genus])
'Hominina'[rank=subtribe],(('Pan paniscus'[rank=species],'Pan troglodytes'
[rank=species])'Pan'[rank=genus])'Panina'[rank=subtribe])'Hominini'[rank=tribe],
(('Gorilla gorilla'[rank=species],'Gorilla beringei'[rank=species])'Gorilla'
[rank=genus])'Gorillini'[rank=tribe])'Homininae'[rank=subfamily];")

# Show the tree's structure
str(tree)

# Plot the tree with node labels
ape::plot.phylo(tree, show.node.label = TRUE, node.depth = 2)

# Add taxonomic rank (stored in the "rank" attribute of the tree)
tree$tip.label = paste(tree$tip.attributes$rank, tree$tip.attributes$Name, sep="\n")
tree$node.label = paste(tree$node.attributes$rank, tree$node.attributes$Name, sep="\n")

# Plot again
ape::plot.phylo(tree, show.node.label = TRUE, node.depth = 2, y.lim=c(0.5, 5.5))
```

`read_one_binary_tree` *Read Tree in Binary Format*

Description

This function reads one tree from a file in binary format.

Usage

```
read_one_binary_tree(file, index = 1, address = NA, metadata = NA)
```

Arguments

`file`	A file name.
`index`	The index of the tree that should be read (starting from 1).
`address`	The address (i.e. byte offset from the start of the file) of the tree that should be read.
`metadata`	An object of class `"BinaryTreeMetadata"` containing the metadata extracted from the tree file. If this is not provided, it will be read from the file (see details).

Details

This function extracts only one tree from the file. The information provided by `metadata` is used to determine where in the file the requested tree starts. If this is not provided, this function will read the metadata from the file (using the `read_binary_tree_metadata` function).

Reusing the metadata is efficient when multiple trees need to be read from the same file (so that the metadata only needs to be read once).

Node attributes (e.g. support values, rates, ages...) are parsed by this function and returned in the `tip.attributes` and `node.attributes` elements of the returned `"phylo"` objects.

Attribute names may appear in any kind of casing (e.g. `Name`, `name` or `NAME`), but they should be treated using case-insensitive comparisons.

Due to limitations with R's integral types, this function may have issues with files larger than 2GB.

Value

An object of class `"phylo"`, compatible with the `ape` package.

In addition to the elements described in the documentation for the `read.tree` function of the `ape` package, a `"phylo"` object produced by this function will also have the following components:

`tip.attributes`	A named list of attributes for the tips of the tree. Each element of this list is a vector of mode character or numeric (depending on the attribute).
`node.attributes`	A named list of attributes for the internal nodes of the tree. Each element of this list is a vector of mode character or numeric (depending on the attribute).

Author(s)

Giorgio Bianchini

References

https://github.com/arklumpus/TreeNode/blob/master/BinaryTree.md

See Also

read_binary_trees, read_binary_tree_metadata, ape, read.tree

Other functions to read trees: read_binary_trees(), read_nwka_nexus(), read_nwka_tree()

Examples

```
# Tree file (replace with your own)
treeFile <- system.file("extdata", "manyTrees.tbi", package="TreeNode")

# Read the 5th tree in the file
tree <- read_one_binary_tree(treeFile, 5)
#Do something with the tree


# Read the binary tree metadata
meta <- read_binary_tree_metadata(treeFile)

# Process every tree in the file
for (add in meta$TreeAddresses)
{
    tree <- read_one_binary_tree(treeFile, address = add, metadata = meta)
    #Do something with the tree
}
```

TreeNode *TreeNode: Read and Write Trees in Binary and NWKA Formats*

Description

The **TreeNode** package provides functions to read and write files containing phylogenetic trees in Binary Tree format and in Newick-with-Attributes (NWKA) format.

TreeNode produces and consumes trees stored in the same "phylo" or "multiPhylo" objects used by the package ape.

More information on **TreeNode** can be found at https://github.com/arklumpus/TreeNode.

Author(s)

Giorgio Bianchini

Maintainer: Giorgio Bianchini <giorgio.bianchini@bristol.ac.uk>

References

Paradis, E., Claude, J. and Strimmer, K. (2004) *APE: analyses of phylogenetics and evolution in R language.* Bioinformatics, **20**, 289-290

See Also

https://github.com/arklumpus/TreeNode, ape

`write_binary_trees` *Write Tree File in Binary Format*

Description

This function writes one or more trees to a file in binary format.

Usage

```
write_binary_trees(trees, file, additional_data = vector("raw", 0))
```

Arguments

`trees`	An object of class `"phylo"` or `"multiPhylo"`.
`file`	A file name.
`additional_data`	A vector of mode raw containg additional binary data that will be included within the tree file.

Details

This function writes all the trees at once. If you wish to write the trees one at a time, you should use the `keep_writing_binary_trees` function.

This function will analyse all the trees to determine whether it is appropriate to include any names or attributes in the file header. It will then write the header, the trees and conclude the file with an appropriate trailer.

The tip names can be specified either as a `Name` element in the tree's `tip.attributes` element, or as the `tip.label` element of the tree. If both are specified, the values stored in the `Name` attribute take precedence (this allows backward compatibility for trees created using `ape`).

The node names and support values can similarly be specified either with a `Name` or `Support` element in the tree's `node.attributes`, or as the tree's `node.label`. If all the node labels can be parsed as numbers, they will be assumed to contain support values; otherwise, they will be assumed to contain node names. If the `node.attributes` already contain a `Name` or `Support` element, the node labels will be ignored.

The additional binary data (if any) will be written in the file after the trees and before the trailer.

Author(s)

Giorgio Bianchini

References

https://github.com/arklumpus/TreeNode/blob/master/BinaryTree.md

See Also

`keep_writing_binary_trees`, `ape`, `write.tree`

Other functions to write trees: `keep_writing_binary_trees()`, `write_nwka_nexus()`, `write_nwka_tree()`

`write_nwka_nexus` *Write Tree File in NEXUS format*

Description

This function writes one or more trees to a NEXUS format file. Within the NEXUS file, the trees are stored in the Newick-with-Attributes (NWKA) format.

Usage

```
write_nwka_nexus(trees, file, translate = TRUE, translate_quotes = TRUE)
```

Arguments

`trees`	An object of class `"phylo"` or `"multiPhylo"`.
`file`	A file name.
`translate`	If this is `TRUE` (the default), the produced nexus tree will contain, in addition to the `Trees` block, a `Taxa` block containing the taxon labels, as well as a `Translate` instruction in the `Trees` block. Otherwise, it will only contain a `Trees` block without a `Translate` instruction.
`translate_quotes`	If this is `TRUE` (the default), the entries in the `Taxa` block and in the `Translate` instruction will be placed within single quotes. Otherwise, they will be written without single quotes.

Details

Only the tip labels are included in the `Taxa` block and the `Translate` instruction (if applicable).

The trees inside the NEXUS file will be stored in NWKA format, including all of the available attributes. This is compatible with the NEXUS specification, because attributes that cannot be represented in standard Newick format are enclosed within square brackets (`[]`);

The tip names can be specified either as a `Name` element in the tree's `tip.attributes` element, or as the `tip.label` element of the tree. If both are specified, the values stored in the `Name` attribute take precedence (this allows backward compatibility for trees created using `ape`).

The node names and support values can similarly be specified either with a `Name` or `Support` element in the tree's `node.attributes`, or as the tree's `node.label`. If all the node labels can be parsed as numbers, they will be assumed to contain support values; otherwise, they will be assumed to contain node names. If the `node.attributes` already contain a `Name` or `Support` element, the node labels will be ignored.

Author(s)

Giorgio Bianchini

References

`https://github.com/arklumpus/TreeNode/blob/master/NWKA.md`

See Also

`ape`, `write.nexus`

Other functions to write trees: `keep_writing_binary_trees()`, `write_binary_trees()`, `write_nwka_tree()`

`write_nwka_tree` *Write Tree File in NWKA format*

Description

This function writes one or more trees in Newick-with-Attributes (NWKA) format to a file or to the standard output.

Usage

```
write_nwka_tree(trees, file = "", append = FALSE, nwka = TRUE, quotes = FALSE)
```

Arguments

`trees`	An object of class `"phylo"` or `"multiPhylo"`.
`file`	A file name. If this is `""` (the default), the tree will be written on the standard output.
`append`	If this is `FALSE` (the default), the output file (if it exists already) is truncated before writing trees (i.e. overwritten). If this is `TRUE`, the trees are appended at the end of the output file.
`nwka`	If this is `TRUE` (the default), the tree will be written in Newick-with-Attributes (NWKA) format. Otherwise, the tree will be written in Newick format (and attributes that cannot be represented in this format will be lost).
`quotes`	If `nwka = FALSE`, this argument determines whether names in the tree file will be enclosed within single quotes (if this is `TRUE`) or not (if this is `FALSE`).

Details

All of the available attributes are written to the file if `nwka = TRUE`. Otherwise, (if available) the tip names and lenghts are always written, as well as the internal nodes' lenghts and support values. If nodes have a name, this is only included if they do not have a support value as well.

The tip names can be specified either as a `Name` element in the tree's `tip.attributes` element, or as the `tip.label` element of the tree. If both are specified, the values stored in the `Name` attribute take precedence (this allows backward compatibility for trees created using `ape`).

The node names and support values can similarly be specified either with a `Name` or `Support` element in the tree's `node.attributes`, or as the tree's `node.label`. If all the node labels can be parsed as numbers, they will be assumed to contain support values; otherwise, they will be assumed to contain node names. If the `node.attributes` already contain a `Name` or `Support` element, the node labels will be ignored.

No attempt is made to fix problematic labels. Thus, if the tip or node names contain special characters, an invalid output may be produced. For example, if the labels contain spaces or commas and `enwk` and `quotes` are both `FALSE`, the output tree may not be parsed correctly. If you wish to produce a tree conforming to the Newick format while fixing problematic tip labels, you should look into the `write.tree` function of the `ape` package.

Author(s)

Giorgio Bianchini

References

https://github.com/arklumpus/TreeNode/blob/master/NWKA.md

See Also

ape, write.tree

Other functions to write trees: keep_writing_binary_trees(), write_binary_trees(), write_nwka_nexus()

Examples

```
# Parse a tree string
# Topology from https://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi?id=207598
tree <- read_nwka_tree(text="(((('Homo sapiens'[rank=species])'Homo'[rank=genus])
'Hominina'[rank=subtribe],(('Pan paniscus'[rank=species],'Pan troglodytes'
[rank=species])'Pan'[rank=genus])'Panina'[rank=subtribe])'Hominini'[rank=tribe],
(('Gorilla gorilla'[rank=species],'Gorilla beringei'[rank=species])'Gorilla'
[rank=genus])'Gorillini'[rank=tribe])'Homininae'[rank=subfamily];")

# Print the tree to the standard output in NWKA format
cat(write_nwka_tree(tree))

# Print the tree to the standard output in Newick format without quotes
cat(write_nwka_tree(tree, nwka = FALSE))

# Print the tree to the standard output in Newick format with quotes
cat(write_nwka_tree(tree, nwka = FALSE, quotes = TRUE))
```

Index