Package 'AhoCorasickTrie'

February 5, 2025

Type Package

Title Fast Searching for Multiple Keywords in Multiple Texts

Version 0.1.3

Description Aho-Corasick is an optimal algorithm for finding many keywords in a text. It can locate all matches in a text in O(N+M) time; i.e., the time needed scales linearly with the number of keywords (N) and the size of the text (M). Compare this to the naive approach which takes O(N*M) time to loop through each pattern and scan for it in the text. This implementation builds the trie (the generic name of the data structure) and runs the search in a single function call. If you want to search multiple texts with the same trie, the function will take a list or vector of texts and return a list of matches to each text. By default, all 128 ASCII characters are allowed in both the keywords and the text. A more efficient trie is possible if the alphabet size can be reduced. For example, DNA sequences use at most 19 distinct characters and usually only 4; protein sequences use at most 26 distinct characters and usually only 20. UTF-8 (Unicode) matching is not currently supported.

License Apache License 2.0

URL https://github.com/chambm/AhoCorasickTrie

BugReports https://github.com/chambm/AhoCorasickTrie/issues

Encoding UTF-8

Imports Rcpp (>= 0.12.5)

LinkingTo Rcpp

Suggests microbenchmark, testthat

RoxygenNote 7.3.2

NeedsCompilation yes

Author Matt Chambers [aut, cre],
Tomas Petricek [aut, cph],
Vanderbilt University [cph]

Maintainer Matt Chambers <matt.chambers42@gmail.com>

Repository CRAN

Date/Publication 2025-02-05 00:30:02 UTC

Contents

`AhoCorasickSearch` *Fast searching for one or more keywords in one or more texts*

Description

Builds an Aho-Corasick trie from one or more keywords and uses it to search one or more texts. For a large number of keywords, Aho-Corasick is much faster than a naive approach (such as `lapply(keywords, gregexpr, text)`).

Use `AhoCorasickSearchList` instead of `AhoCorasickSearch` when you want to keep the matches of each input text separate. If the input texts have names, the resulting list of matches will include those names and non-matched texts will be excluded from the results. If the input texts do not have names, then the resulting list of matches will be in the same order as the input texts, and non-matched texts will be kept to preserve that order. Thus, it is more efficient to use named input texts (so non-matched texts can be dropped).

The default alphabet allows all 128 ASCII characters in the keywords and the texts. Characters outside this range will cause an error. A more efficient trie is possible if the alphabet size can be reduced. For example, DNA sequences use at most 19 distinct characters and usually only 4; protein sequences use at most 26 distinct characters and usually only 20. Set the `alphabet` parameter if a reduced alphabet is appropriate.

UTF-8 (Unicode) matching is not currently supported.

Usage

```
AhoCorasickSearch(
  keywords,
  text,
  alphabet = "ascii",
  groupByKeyword = FALSE,
  iterationFeedback = 0L
)
```

Arguments

`keywords`	Character vector of one or more keywords
`text`	Character vector of one or more texts to search
`alphabet`	Alphabet to use; one of `ascii`, `aminoacid`, or `nucleicacid`
`groupByKeyword`	If true, matches are grouped by keyword (instead of by text)
`iterationFeedback`	When set to a positive integer `i`, console output will indicate when searching every `i`th text

Value

List of matches, grouped by either text or by keyword

See Also

- Aho-Corasick string matching in C# for the article this package is based on
- `Biostrings` `matchPDict` for a more memory efficient, but DNA-only, implementation of the algorithm

Examples

```
listEquals = function(a, b) { is.null(unlist(a)) && is.null(unlist(b)) ||
                              !is.null(a) && !is.null(b) && all(unlist(a) == unlist(b)) }

# 1. Search for multiple keywords in a single text
keywords = c("Abra", "cadabra", "is", "the", "Magic", "Word")
oneSearch = AhoCorasickSearch(keywords, "Is Abracadabra the Magic Word?")
stopifnot(listEquals(oneSearch[[1]][[1]], list(keyword="Abra", offset=4)))
stopifnot(listEquals(oneSearch[[1]][[2]], list(keyword="cadabra", offset=8)))
stopifnot(listEquals(oneSearch[[1]][[3]], list(keyword="the", offset=16)))
stopifnot(listEquals(oneSearch[[1]][[4]], list(keyword="Magic", offset=20)))
stopifnot(listEquals(oneSearch[[1]][[5]], list(keyword="Word", offset=26)))

# 2. Search multiple named texts in a named list with keyword grouping and aminoacid alphabet
# * all matches to a keyword are accessed by name
# * non-matched keywords are dropped
proteins = c(protein1="PEPTIDEPEPTIDEDADADARARARARAKEKEKEKEPEPTIDE",
             protein2="DERPADERPAPEWPEWPEEPEERAWRAWWARRAGTAGPEPTIDEKESEQUENCE")
peptides = c("PEPTIDE", "DERPA", "SEQUENCE", "KEKE", "PEPPIE")

peptideSearch = AhoCorasickSearch(peptides, proteins, alphabet="aminoacid", groupByKeyword=TRUE)
stopifnot(listEquals(peptideSearch$PEPTIDE, list(list(keyword="protein1", offset=1),
                                                 list(keyword="protein1", offset=8),
                                                 list(keyword="protein1", offset=37),
                                                 list(keyword="protein2", offset=38))))
stopifnot(listEquals(peptideSearch$DERPA, list(list(keyword="protein2", offset=1),
                                               list(keyword="protein2", offset=6))))
stopifnot(listEquals(peptideSearch$SEQUENCE, list(list(keyword="protein2", offset=47))))
stopifnot(listEquals(peptideSearch$KEKE, list(list(keyword="protein1", offset=29),
                                              list(keyword="protein1", offset=31),
                                              list(keyword="protein1", offset=33))))
stopifnot(listEquals(peptideSearch$PEPPIE, NULL))

# 3. Grouping by keyword without text names: offsets are given without reference to the text
names(proteins) = NULL
peptideSearch = AhoCorasickSearch(peptides, proteins, groupByKeyword=TRUE)
stopifnot(listEquals(peptideSearch$PEPTIDE, list(1, 8, 37, 38)))
stopifnot(listEquals(peptideSearch$DERPA, list(1, 6)))
stopifnot(listEquals(peptideSearch$SEQUENCE, list(47)))
stopifnot(listEquals(peptideSearch$KEKE, list(29, 31, 33)))
```

`AhoCorasickSearchList` *Fast searching for one or more keywords in a list of texts*

Description

Builds an Aho-Corasick trie from one or more keywords and uses it to search a list of one or more texts. For a large number of keywords, Aho-Corasick is much faster than a naive approach (such as `lapply(keywords, gregexpr, text)`).

Use `AhoCorasickSearchList` instead of `AhoCorasickSearch` when you want to keep the matches of each input sublist separate. If the sublists of the input list have names, the resulting list of lists will use those names, but sublists with no matches will still be in the resulting list. If the texts of the sublists have names, the resulting sublists of matches will use those names, and the texts with no matches will be dropped. If the input texts do not have names, then the resulting sublists of matches will be in the same order as the input texts, and non-matched texts will be kept to preserve that order. Thus, it is more efficient to use named input texts (so non-matched texts can be dropped).

The default alphabet allows all 128 ASCII characters in the keywords and the texts. Characters outside this range will cause an error. A more efficient trie is possible if the alphabet size can be reduced. For example, DNA sequences use at most 19 distinct characters and usually only 4; protein sequences use at most 26 distinct characters and usually only 20. Set the `alphabet` parameter if a reduced alphabet is appropriate.

UTF-8 (Unicode) matching is not currently supported.

Usage

```
AhoCorasickSearchList(
  keywords,
  textList,
  alphabet = "ascii",
  groupByKeyword = FALSE,
  iterationFeedback = 0L
)
```

Arguments

`keywords`	Character vector of one or more keywords
`textList`	List of lists, each sublist with one or more texts to search
`alphabet`	Alphabet to use; one of `ascii`, `aminoacid`, or `nucleicacid`
`groupByKeyword`	If true, matches are grouped by keyword (instead of by text)
`iterationFeedback`	When set to a positive integer `i`, console output will indicate when searching every `i`th text

Value

List of lists of matches, grouped by either text or by keyword (each list of texts gets its own list of matches)

See Also

- Aho-Corasick string matching in C# for the article this package is based on
- `Biostrings` `matchPDict` for a more memory efficient, but DNA-only, implementation of the algorithm

Examples

```
listEquals = function(a, b) { is.null(unlist(a)) && is.null(unlist(b)) ||
                              !is.null(a) && !is.null(b) && all(unlist(a) == unlist(b)) }
keywords = c("Abra", "cadabra", "is", "the", "Magic", "Word")

# 1. Search a list of lists without names
# * sublists are accessed by index
# * texts are accessed by index
# * non-matched texts are kept (input index order is preserved)
listSearch = AhoCorasickSearchList(keywords,
                                   list(c("What in", "the world"),
                                        c("is"),
                                        "secret about",
                                        "the Magic Word?"))
stopifnot(listEquals(listSearch[[1]][[1]], list()))
stopifnot(listEquals(listSearch[[1]][[2]][[1]], list(keyword="the", offset=1)))
stopifnot(listEquals(listSearch[[2]][[1]][[1]], list(keyword="is", offset=1)))
stopifnot(listEquals(listSearch[[3]], list()))
stopifnot(listEquals(listSearch[[4]][[1]][[1]], list(keyword="the", offset=1)))
stopifnot(listEquals(listSearch[[4]][[1]][[2]], list(keyword="Magic", offset=5)))
stopifnot(listEquals(listSearch[[4]][[1]][[3]], list(keyword="Word", offset=11)))

# 2. Search a named list of named lists
# * sublists are accessed by name
# * matched texts are accessed by name
# * non-matched texts are dropped
namedSearch = AhoCorasickSearchList(keywords,
                                    list(subject=c(phrase1="What in", phrase2="the world"),
                                           verb=c(phrase1="is"),
                                           predicate1=c(phrase1="secret about"),
                                           predicate2=c(phrase1="the Magic Word?")))
stopifnot(listEquals(namedSearch$subject$phrase2[[1]], list(keyword="the", offset=1)))
stopifnot(listEquals(namedSearch$verb$phrase1[[1]], list(keyword="is", offset=1)))
stopifnot(listEquals(namedSearch$predicate1, list()))
stopifnot(listEquals(namedSearch$predicate2$phrase1[[1]], list(keyword="the", offset=1)))
stopifnot(listEquals(namedSearch$predicate2$phrase1[[2]], list(keyword="Magic", offset=5)))
stopifnot(listEquals(namedSearch$predicate2$phrase1[[3]], list(keyword="Word", offset=11)))
```

`AhoCorasickTrie` *AhoCorasickTrie: fast searching for multiple keywords in multiple texts*

Description

Builds an Aho-Corasick trie from one or more keywords and uses it to search one or more texts. For a large number of keywords, Aho-Corasick is much faster than a naive approach (such as `lapply(keywords, gregexpr, text)`).

Author(s)

Maintainer: Matt Chambers <matt.chambers42@gmail.com>

Authors:

- Tomas Petricek [copyright holder]

Other contributors:

- Vanderbilt University [copyright holder]

See Also

Useful links:

- https://github.com/chambm/AhoCorasickTrie
- Report bugs at https://github.com/chambm/AhoCorasickTrie/issues

Index