



Interaction

- [Help](#)
- [About Wikipedia](#)
- [Community portal](#)
- [Recent changes](#)
- [Contact page](#)

Tools

- [What links here](#)
- [Related changes](#)
- [Upload file](#)
- [Special pages](#)
- [Permanent link](#)
- [Page information](#)
- [Wikidata item](#)
- [Cite this page](#)

Print/export

- Create a book
- Download as PDF
- Printable version

Languages

- Čeština
- Deutsch
- Español
- فارسی
- Français
- 한국어
- 日本語
- Polski
- Português
- Русский
- Српски / srpski
- Українська

 Edit links

[Article](#) [Talk](#) [Read](#) [Edit](#) [View history](#)  Search

From Wikipedia, the free encyclopedia  
(Redirected from [Aho–Corasick string matching algorithm](#))

This article includes a [list of references](#), related reading or [external links](#), **but its sources remain unclear because it lacks [inline citations](#)**. Please [improve](#) this article by introducing more precise citations. *(February 2013)*

In [computer science](#), the **Aho–Corasick algorithm** is a [string searching algorithm](#) invented by [Alfred V. Aho](#) and Margaret J. Corasick.<sup>[1]</sup> It is a kind of dictionary-matching algorithm that locates elements of a finite set of strings (the "dictionary") within an input text. It matches all patterns simultaneously. The [complexity](#) of the algorithm is linear in the length of the patterns plus the length of the searched text plus the number of output matches. Note that because all matches are found, there can be a quadratic number of matches if every substring matches (e.g. dictionary = a, aa, aaa, aaaa and input string is aaaa).

Informally, the algorithm constructs a [finite state machine](#) that resembles a [trie](#) with additional links between the various internal nodes. These extra internal links allow fast transitions between failed pattern matches (e.g. a search for `cat` in a trie that does not contain `cat`, but contains `cart`, and thus would fail at the node prefixed by `ca`), to other branches of the trie that share a common prefix (e.g., in the previous case, a branch for `attribute` might be the best lateral transition). This allows the automaton to transition between pattern matches without the need for backtracking.

When the pattern dictionary is known in advance (e.g. a [computer virus](#) database), the construction of the automaton can be performed once off-line and the compiled automaton stored for later use. In this case, its run time is linear in the length of the input plus the number of matched entries.

The Aho–Corasick string matching algorithm formed the basis of the original [Unix command fgrep](#).

- 1 Example
- 2 See also
- 3 References
- 4 External links

### Example [\[edit\]](#)

In this example, we will consider a dictionary consisting of the following words: {a,ab,bab,bc,bca,c,caa}.

The graph below is the Aho–Corasick data structure constructed from the specified dictionary, with each row in the table representing a node in the trie, with the column path indicating the (unique) sequence of characters from the root to the node.

The data structure has one node for every prefix of every string in the dictionary. So if (bca) is in the dictionary, then there will be nodes for (bca), (bc), (b), and (). If a node is in the dictionary then it is a blue node. Otherwise it is a grey node. There is a black directed "child" arc from each node to a node whose name is found by appending one character. So there is a black arc from (bc) to (bca). There is a blue directed "suffix" arc from each node to the node that is the longest possible strict suffix of it in the graph. For example, for node (caa), its strict suffixes are (aa) and (a) and (). The longest of these that exists in the graph is (a). So there is a blue arc from (caa) to (a). The blue arcs can be computed in linear time by repeatedly traversing the blue arcs of a node's parent until the traversing node has a child matching the character of the target node.

There is a green "dictionary suffix" arc from each node to the next node in the dictionary that can be reached by following blue arcs. For example, there is a green arc from (bca) to (a) because (a) is the first node in the dictionary (i.e. a blue node) that is reached when following the blue arcs to (ca) and then on to (a). The green arcs can be computed in linear time by repeatedly traversing blue arcs until a filled in node is found, and memoizing this information.

**Dictionary {a, ab, bab, bc, bca, c, caa}**

--	--	--	--

Path	In Dictionary	Suffix Link	Dict Suffix Link
( )	–		
(a)	+	( )	
(ab)	+	(b)	
(b)	–	( )	
(ba)	–	(a)	(a)
(bab)	+	(ab)	(ab)
(bc)	+	(c)	(c)
(bca)	+	(ca)	(a)
(c)	+	( )	
(ca)	–	(a)	(a)
(caa)	+	(a)	(a)

At each step, the current node is extended by finding its child, and if that doesn't exist, finding its suffix's child, and if that doesn't work, finding its suffix's suffix's child, and so on, finally ending in the root node if nothing's seen before.

When the algorithm reaches a node, it outputs all the dictionary entries that end at the current character position in the input text. This is done by printing every node reached by following the dictionary suffix links, starting from that node, and continuing until it reaches a node with no dictionary suffix link. In addition, the node itself is printed, if it is a dictionary entry.

Execution on input string `abccab` yields the following steps:

Analysis of input string `abccab`

Node	Remaining String	Output:End Position	Transition	Output
( )	abccab		start at root	
(a)	bccab	a:1	( ) to child (a)	Current node
(ab)	ccab	ab:2	(a) to child (ab)	Current node
(bc)	cab	bc:3, c:3	(ab) to suffix (b) to child (bc)	Current Node, Dict suffix node
(c)	ab	c:4	(bc) to suffix (c) to suffix ( ) to child (c)	Current node
(ca)	b	a:5	(c) to child (ca)	Dict suffix node
(ab)		ab:6	(ca) to suffix (a) to child (ab)	Current node

See also [\[edit\]](#)

Other multiple-pattern string search algorithms are:

- [Commentz-Walter](#)
- [Rabin–Karp](#)



References [\[edit\]](#)

- ↑  **Aho, Alfred V.**; Corasick, Margaret J. (June 1975). "Efficient string matching: An aid to bibliographic search". *Communications of the ACM* **18** (6): 333–340. doi:10.1145/360825.360855





## External links [\[edit\]](#)

- [Set Matching and Aho–Corasick Algorithm](#) , lecture slides by Pekka Kilpeläinen
- [Aho–Corasick entry](#)  in NIST's [Dictionary of Algorithms and Data Structures](#)
- [Aho-Corasick implementation in C++](#) 

Categories: [String matching algorithms](#)

This page was last modified on 27 June 2015, at 22:50.

Text is available under the [Creative Commons Attribution-ShareAlike License](#); additional terms may apply. By using this site, you agree to the [Terms of Use](#) and [Privacy Policy](#). Wikipedia® is a registered trademark of the [Wikimedia Foundation, Inc.](#), a non-profit organization.

[Privacy policy](#) [About Wikipedia](#) [Disclaimers](#) [Contact Wikipedia](#) [Developers](#) [Mobile view](#)

