String searching algorithm

From Wikipedia, the free encyclopedia
(Redirected from Substring search)

In computer science, **string searching algorithms**, sometimes called **string matching algorithms**, are an important class of string algorithms that try to find a place where one or several strings (also called patterns) are found within a larger string or text.

Let Σ be an alphabet (finite set). Formally, both the pattern and searched text are vectors of elements of Σ. The Σ may be a usual human alphabet (for example, the letters A through Z in the Latin alphabet). Other applications may use *binary alphabet* (Σ = {0,1}) or *DNA alphabet* (Σ = {A,C,G,T}) in bioinformatics.

In practice, how the string is encoded can affect the feasible string search algorithms. In particular if a variable width encoding is in use then it is slow (time proportional to N) to find the Nth character. This will significantly slow down many of the more advanced search algorithms. A possible solution is to search for the sequence of code units instead, but doing so may produce false matches unless the encoding is specifically designed to avoid it.

Basic classification [edit]

The various algorithms can be classified by the number of patterns each uses.

Single pattern algorithms [edit]

Let *m* be the length of the pattern and let *n* be the length of the searchable text.

Algorithm	Preprocessing time	Matching time[1]
Naïve string search algorithm	0 (no preprocessing)	Θ((n−m) m)
Rabin–Karp string search algorithm	Θ(m)	average Θ(n+m), worst Θ((n−m) m)
Finite-state automaton based search	Θ(m |Σ|)	Θ(n)
Knuth–Morris–Pratt algorithm	Θ(m)	Θ(n)
Boyer–Moore string search algorithm	Θ(m + |Σ|)	Ω(n/m), O(nm)
Bitap algorithm (*shift-or*, *shift-and*, *Baeza–Yates–Gonnet*)	Θ(m + |Σ|)	O(mn)

[1]Asymptotic times are expressed using O, Ω, and Θ notation

The **Boyer–Moore string search algorithm** has been the standard benchmark for the practical string search literature.[1]

Algorithms using a finite set of patterns [edit]

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

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
Bahasa Indonesia
Italiano
Basa Jawa
日本語
Русский
Српски / srpski
Srpskohrvatski / српскохрватски
Suomi
Türkçe
Українська
中文
Edit links

- Aho–Corasick string matching algorithm
- Commentz-Walter algorithm
- Rabin–Karp string search algorithm

Algorithms using an infinite number of patterns [edit]

Naturally, the patterns can not be enumerated in this case. They are represented usually by a regular grammar or regular expression.

Other classification [edit]

This section **does not cite any references or sources**. Please help improve this section by adding citations to reliable sources. Unsourced material may be challenged and removed. *(July 2013)*

Other classification approaches are possible. One of the most common uses preprocessing as main criteria.

Classes of string searching algorithms[2]

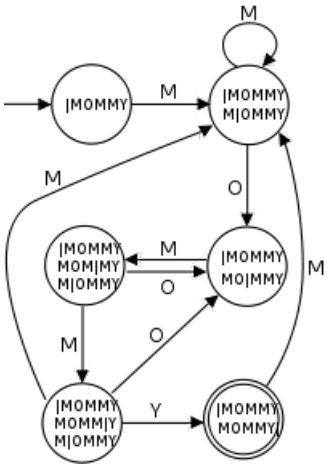
	Text not preprocessed	Text preprocessed
Patterns not preprocessed	Elementary algorithms	Index methods
Patterns preprocessed	Constructed search engines	Signature methods

Naïve string search [edit]

A simple but inefficient way to see where one string occurs inside another is to check each place it could be, one by one, to see if it's there. So first we see if there's a copy of the needle in the first character of the haystack; if not, we look to see if there's a copy of the needle starting at the second character of the haystack; if not, we look starting at the third character, and so forth. In the normal case, we only have to look at one or two characters for each wrong position to see that it is a wrong position, so in the average case, this takes O(n + m) steps, where n is the length of the haystack and m is the length of the needle; but in the worst case, searching for a string like "aaaab" in a string like "aaaaaaaaab", it takes O(nm)

Finite state automaton based search [edit]

In this approach, we avoid backtracking by constructing a deterministic finite automaton (DFA) that recognizes stored search string. These are expensive to construct—they are usually created using the powerset construction—but are very quick to use. For example, the DFA shown to the right recognizes the word "MOMMY". This approach is frequently generalized in practice to search for arbitrary regular expressions.



Stubs [edit]

Knuth–Morris–Pratt computes a DFA that recognizes inputs with the string to search for as a suffix, Boyer–Moore starts searching from the end of the needle, so it can usually jump ahead a whole needle-length at each step. Baeza–Yates keeps track of whether the previous j characters were a prefix of the search string, and is therefore adaptable to fuzzy string searching. The bitap algorithm is an application of Baeza–Yates' approach.

Index methods [edit]

Faster search algorithms are based on preprocessing of the text. After building a substring index, for example a suffix tree or suffix array, the occurrences of a pattern can be found quickly. As an example, a suffix tree can be built in $\Theta(n)$ time, and all z occurrences of a pattern can be found in $O(m)$ time under the assumption that the alphabet has a constant size and all inner nodes in the suffix tree know what leaves are underneath them. The latter can be accomplished by running a DFS algorithm from the root of the suffix tree.

Other variants [edit]

Some search methods, for instance trigram search, are intended to find a "closeness" score between the search string and the text rather than a "match/non-match". These are sometimes called "fuzzy" searches.

See also [edit]

- Sequence alignment
- Pattern matching
- Compressed pattern matching
- Approximate string matching
- String metric

Academic conferences on text searching [edit]

- Combinatorial pattern matching (CPM), a conference on combinatorial algorithms for strings, sequences, and trees.
- String Processing and Information Retrieval (SPIRE), an annual symposium on string processing and information retrieval.
- Prague Stringology Conference (PSC), an annual conference on algorithms on strings and sequences.
- Competition on Applied Text Searching (CATS), an annual series of evaluations of text searching algorithms.

References [edit]

1. ^ Hume; Sunday (1991). "Fast String Searching". *Software: Practice and Experience* **21** (11): 1221–1248. doi:10.1002/spe.4380211105.
2. ^ Melichar, Borivoj, Jan Holub, and J. Polcar. Text Searching Algorithms. Volume I: Forward String Matching. Vol. 1. 2 vols., 2005. http://stringology.org/athens/TextSearchingAlgorithms/.

- R. S. Boyer and J. S. Moore, *A fast string searching algorithm*, Carom. ACM 20, (10), 262–272(1977).
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Chapter 32: String Matching, pp.906–932.

External links [edit]

- Huge (maintained) list of pattern matching links Last updated:12/27/2008 20:18:38
- StringSearch – high-performance pattern matching algorithms in Java – Implementations of many String-Matching-Algorithms in Java (BNDM, Boyer-Moore-Horspool, Boyer-Moore-Horspool-Raita, Shift-Or)
- Exact String Matching Algorithms — Animation in Java, Detailed description and C implementation of many algorithms.
- Boyer-Moore-Raita-Thomas
- (PDF) Improved Single and Multiple Approximate String Matching
- Kalign2: high-performance multiple alignment of protein and nucleotide sequences allowing external features
- C implementation of Suffix Tree based Pattern Searching

Categories: String matching algorithms