WIKIPEDIA
The Free Encyclopedia

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
Français
Русский
Српски / srpski
Українська
Edit links

Boyer–Moore–Horspool algorithm

From Wikipedia, the free encyclopedia

It has been suggested that *Raita Algorithm* be merged into this article. (Discuss) *Proposed since March 2015.*

In computer science, the **Boyer–Moore–Horspool algorithm** or **Horspool's algorithm** is an algorithm for finding substrings in strings. It was published by Nigel Horspool in 1980.[1]

It is a simplification of the Boyer–Moore string search algorithm which is related to the Knuth–Morris–Pratt algorithm. The algorithm trades space for time in order to obtain an average-case complexity of O(N) on random text, although it has O(MN) in the worst case, where the length of the pattern is M and the length of the search string is N.

Contents [hide]

Description [edit]

Like Boyer–Moore, Boyer–Moore–Horspool preprocesses the pattern to produce a table containing, for each symbol in the alphabet, the number of characters that can safely be skipped. The preprocessing phase, in pseudocode, is as follows (for an alphabet of 256 symbols, i.e., bytes):

```
function preprocess(pattern)
    T ← new table of 256 integers
    for i from 0 to 256 exclusive
        T[i] ← length(pattern)
    for i from 0 to length(pattern) - 1 exclusive
        T[pattern[i]] ← length(pattern) - 1 - i
    return T
```

Pattern search proceeds as follows.[*disputed – discuss*] The procedure `search` reports the index of the first occurrence of `needle` in `haystack`.

```
function search(needle, haystack)
    T ← preprocess(needle)
    skip ← 0
    while length(haystack) - skip ≥ length(needle)
        i ← length(needle) - 1
        while haystack[skip + i] = needle[i]
            if i = 0
                return skip
            i ← i - 1
        skip ← skip + T[haystack[skip + length(needle) - 1]]
    return not-found
```

Performance [edit]

The algorithm performs best with long needle strings, when it consistently hits a non-matching character at or near the final byte of the current position in the haystack and the final byte of the needle does not occur elsewhere within the needle. For instance a 32 byte needle ending in "z" searching through a 255 byte haystack which does not have a 'z' byte in it would take up to 224 byte comparisons.

The best case is the same as for the Boyer–Moore string search algorithm in big O notation, although the constant overhead of initialization and for each loop is less.

The worst case behavior happens when the bad character skip is consistently low (with the lower limit of 1 byte movement) and a large portion of the needle matches the haystack. The bad character skip is only low, on a partial match, when the final character of the needle also occurs elsewhere within the needle, with 1 byte movement happening when the same byte is in both of the last two positions.

The canonical degenerate case similar to the above "best" case is a needle of an 'a' byte followed by 31 'z' bytes in a haystack consisting of 255 'z' bytes. This will do 31 successful byte comparisons, a 1 byte comparison that fails and then move forward 1 byte. This process will repeat 223 more times (255 - 32), bringing the total byte comparisons to 7,168 (32 * 224).

The worst case is significantly higher than for the Boyer–Moore string search algorithm, although obviously this is hard to achieve in normal use cases. It is also worth noting that this worst case is also the worst case for the naive (but usual) memmem() algorithm, although the implementation of that tends to be significantly optimized (and is more cache friendly).

See also [edit]

- Boyer–Moore string search algorithm

References [edit]

1. ^ R. N. Horspool (1980). "Practical fast searching in strings". *Software - Practice & Experience* **10** (6): 501–506. doi:10.1002/spe.4380100608. CiteSeerX: 10.1.1.63.3421.

External links [edit]

- Description of the algorithm

Categories: String matching algorithms