# English hyphenation algorithm in PHP


| | |
|---|---|
| **Description** | Implementing English hyphenation algorithm in PHP. |
| **Period** | Summer 2013 |
| **Languages & Libs** | PHP |
| **Tags** | Hyphenation, Blog, GitHub, Data Structures |
| **GitHub** | nikonyrh/hyphenator-php |

A good presentation about hyphenation in HTML documents can be seen here, but it is client side (JavaScript) oriented. Basically you shouldn't use justified text unless it is hyphenated, because long words will cause huge spaces between words to make the line stretch out the whole width of the element. I found a few PHP scripts such as phpHyphenator 1.5, but typically they weren't implemented as a single stand-alone PHP class. Since the underlying algorithm is fairly simple, I decided to write it from scratch.

The algorithm itself has been known since 1980s, and it is based on pre-calculated patterns, which indicate how fragments of words could be hyphenated. These are fairly simple to understand and debug, and can most likely correctly hyphenate any new previously unseen words as well. The algorithm scans the inputted word to check which patterns can be found, and this is illustrated in Table 1. Here the word to be hyphenated is "algorithm", from which the following patterns can be found: `4l1g4`, `lgo3`, `1go`, `2ith` and `4h1m`. During pattern matching these numbers are ignored, and when matches are found then all existing numbers are injected between corresponding characters. Once this is done, the highest numeric value is defined for each column. If the highest number is odd, it indicates that this is a possible position for hyphenation. Typically in TEX some additional rules are enforced, such as the minimum allowed length for a syllable. In this example, the hyphenation "algorith-m" doesn't make much sense.

Table 1: Example result of hyphenation algorithm on the word "algorithm".

| a | | l | | g | | o | | r | | i | | t | | h | | m |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4 | l<br>l | 1<br><br>1 | g<br>g<br>g | 4 | o<br>o | 3 | | 2 | i | | t | 4 | h<br>h | 1 | m |
| a<br>a | 4 | l<br>l | 1<br>– | g<br>g | 4 | o<br>o | 3<br>– | r<br>r | 2 | i<br>i | o | t<br>t | 4 | h<br>h | 1<br>– | m<br>m |

The open source project phpHyphenator has a very good support for multi-byte characters because of the extensive use of `mb_`-prefix functions such as `mb_substr`. On a negative side it was not object oriented, uses `$GLOBALS` for configuration, somewhat inefficient pattern searching and a bit gimmicky handling of HTML tags. Many of these are most likely due to the usage of `mb_substr` instead of *regular expressions*.

In contrast my `Hyphenator` is a stand-alone class with two methods, the `__construct` which takes the pattern list and the hyphen symbol as a parameter, and `hyphenate` which returns the hyphenated string. The constructor uses provided patterns to build a trie data structure, which is a tree-like data structure that enables a very efficient string searching, especially in an incremental case. It also makes pattern searching code more clean, when we aren't forced into calculating odd substring offsets and avoiding indexing beyond the string length.

Text hyphenation begins by splitting the string into chunks using sequences of non-alphabetical characters as delimiters. Then these are iterated through, and a counter is used to keep track of the number of open HTML tags. This ensures that we don't add hyphenations into for example CSS class names. When a hyphenation pattern is observed in the string at some position, "hyphenation numbers" are stored into a separate table at corresponding "gap indexes". After all maximum numbers are found, the word is iterated character-by-character and hyphenations are inserted into the output when possible.