HTML Parser with Tag and Attirbute Spelling Correction

Chadi Helwe[1], John Abou Jaoudeh[1]
[1]American University of Beirut, Beirut, Lebanon

As HTML invades the world wide web, more and more HTML pages are being written. And, almost all published websites online do not comply with W3 standards. Several factors cause HTML invalidity, some of which are: incorrect placement of tags, invalid attributes for tags, misspelled tags... Invalid HTML leads to issues such as: a decrease in loading DOM content, decrease in page performance, incorrect display of content, and many others. To minimize these errors, we propose an algorithm for HTML spelling correction, and an enhancement to missing tag correction. We believe that this would highly minimize the errors in HTML pages, increase DOM content load time, and improve the performance of web pages especially on devices with limited resources such as mobile devices.

***Index Terms*—HTML, Parser, Tag Correction, Spelling Correction.**

INTRODUCTION

The Internet is an enormous library of information. This information is mainly presented to people using websites. Information on websites is displayed using HTML tags, and attributes. Web browsers take this HTML code, and parse it to produce a user friendly layout for non-computer specialists; in other words, the information becomes readable by everyone. A single web page usually contains at least a hundred HTML tags, each HTML tag may include several attributes. Those tags are written by developers.

If a person is writing an English essay, one would notice that there would be several typing mistakes. Developers tend to do spelling mistakes as well, in this incident what is misspelled would be the HTML code: tags, and attributes. We bet you did not notice attribute was misspelled in the title of this proposal, or did you? When a web browser encounters a misspelled tag, an error is not thrown; browsers continue parsing the rest of the page ignoring this tag. In this case, the output presented by the web browser is not equivalent to the one intended by the developer. It is frequent that developers do not notice this kind of errors.

Incorrect output is not the only problem caused by misspelling words. If we take a closer look on how a browser renders HTML code into graphics, we would notice that misspelled tags, and attributes may also show a decline in a web page's performance mainly on devices with limited resources such as smartphones; currently, the performance of HTML websites on mobile devices is a hot topic, especially when comparing HTML5 mobile application development to native application development.

What if a web browser can overcome misspelled words? We propose an HTML parser with word correction. Parser spelling mistakes auto correction would improve a web page's performance, and correctness.

HTML KNOWN ISSUES

One of the main components of a web browser is the web browser engine[1]; this component is a program which transform content written in markup languages into the graphics we see when we open a website.

The web browser engine consists of several modules, one of which is the XML parser; this module parses the HTML file to generate the DOM tree, and the render tree. The render tree is then given to the rendering engine module, and is converted into layers of paint, events, and event listeners. An invalid HTML file would generate an incorrect render tree, in other words, not the expected output.

Shan Chen et al. [2] state that only 5% of webpages are valid according to HTML standards, thus generating incorrect render, and DOM trees.

Incorrectly structured render and DOM trees would delay the load time of the page, and would also cause decrease in performance. One of the factors when measuring the performance is the frames per second of a web page. In most cases, for a page to run smoothly, fps should not drop below 30. This is mainly a big issue on mobile devices since more scrolling is required, and every time a user scrolls, the web browser engine needs to repaint; the paint is dependent on the render tree. This becomes even more problematic when CSS, and JavaScript since both need the DOM tree.

Another problem which invalid HTML is that each browser displays it differently, mainly because each browser uses a different web browser engine; Google Chrome and Opera use Blink, Microsoft's Internet Explorer and Spartan uses Trident on Windows (Tasman on Macintosh), Firefox uses Gecko, Safari uses WebKit, and Opera uses Presto. Each layout engine has it's own advantages, and disadvantages[3]. To the best of our knowledge, none of these engines has HTML spelling correction; but some of them do contain missing tag correction, but not powerful enough to fix a lot of cases.

RELATED WORK

The problem of correcting HTML to become validwas tackle by a lot of researchers, each one having their own approach.

Samimi *et al.*[4] suggested two plugins for Eclipse to solve this problem for the PHP language. Those tools can help the developer create valid HTML codes when it is generated by PHP.
The first tool is PHPQuickFix, a plugin that can statically check for incorrect HTML structure such as open, and not ending tags. This tool uses a stack as a data structure to store the open elements, and to pop them when PHPQuickFix finds the ending tag. This approach cannot find, and correct all HTML errors. One case is when HTML errors occur in a PHP file where the data is acquired from an external data source, such as a database.
The second tool provided by the authors will solve the problem we have with PHPQuickFix. The approach of PHPRepair is based on Input-Output based repair. This approach considers that we have a collection of PHP files. The program tests the output of the PHP file, and compares with the expected output; if it passes the test that means the output generated by the PHP script is correct.

$$t = (p, o)$$

t stands for the test, p stand for the actual output, and o stands for the expected output. If the test fails, we need to generate a correct PHP file that gives us the output we want. We say that p and p' are *repair convertible* if one can be generated from the other by adding, removing or modifying HTML code created by string literal in PHP like *print* or *echo*. According to the paper, "a *repair problem* consists of a program p and a set T of tests. A solution of the repair problem is a program p' such that p and p' are repair convertible, and p' passes all tests in T"[4]. The output of the actual PHP file is characterized as the concatenation of all HTML codes printed as a literal string in PHP, $c_1....c_n = output$.
According to this paper, those tools are efficient because they can find and correct most of the HTML codes generated by the PHP file. Those tools did not provide a way to solve misspelled HTML's keywords and is restricted to the PHP language; language like Java, ASP or Ruby that can generate HTML documents will not be able to benefit from those plugins.

Moller and Schwarz[2] based their algorithm on abstract versions of SGML by removing some features.The algorithm considers that we have a DTD for one of the versions of the HTML language. DTD is a formal structure of how a document can be parsed.In this research, a DTD is described as a content model automaton.
The algorithm uses a stack that called context stack to store the state of the parser.According to the paper, the set of possible contexts is described as

$$H = E * Q * P(E) * P(E)$$

($P(E)$ denotes the powerset of E)

The context is defined as $c_n = (a, q, i, n)$ is at top of the stack $c_1...c_n \in H^*$; a refers to the current context, q refers to the current state, i refers to the current inclusion, and n refers to the current exclusion.We say that an element b is permitted in the current context if (a, q, i, n) if $d(q, b) \neq undefined$. If we have a tag b that is above the tag a in the context state that means that a is the parent of b.In the algorithm, we have two functions called OmitStart and OmitEnd. OmitStart may remove a start tag if this exclusion does not cause ambiguity, and OmitEnd may remove an end tag if it is followed by an end tag of another element or if we have a new start tag.
For example, if the parser found a start tag like $< a >$ which is permitted in the current context, It will be pushed onto the stack. Later on, if the parser find an end tag $< /a >$and matches a; the current context will be popped off the stack.During the parsing, if we got *an error* result it means that it is a parsing error because the parser found an end tag and its stack is empty.
The second approach given by the authors is based on the algorithm given above, but instead of working only on HTML documents they presented an algorithm that work on the grammar of the language.The grammar of the language is represented as a context-free grammar. The idea of their algorithm is to generate constraints and then to solve those constraints. According to the authors, their algorithm will terminate successfully if and only if a solution exists to the constraints that mean they have a valid context-free grammar. This paper provided us with two algorithms to generate valid HTML documents. The first algorithm works on the language itself. The second algorithm works on the grammar of the HTML language. According to the experimental results done by the authors their algorithms found most of the errors in an HTML document.Those two algorithms did not take into consideration the problem to solve misspelled HTML's keywords. Our tool will be based on the first algorithm done by Moller and Schwarz to solve missing tags in HTML.

OUR APPROACH

Our approach is to create a parser that can generate a valid HTML code by correcting or suggesting the developer of missing tags and misspelling tags or attributes. Our suggestion can be better than the ones proposed in the papers because we are taking into consideration the issue of correcting misspelled tags. The solution is divided into two phases: the first phase of the parser is to correct HTML keywords and the second phase is to check for missing tags.

A. Correction of HTML Keyword

This phase of the tool is responsible to correct misspelled tags and attribute in the HTML language.We propose to use the *edit distance*[5] algorithm to solve this problem. *Edit distance* algorithm is widely used in information retrieval to check for the correct misspelled keyword by choosing the nearest one.
The *edit distance* algorithm works as follow; it takes two strings and checks for the minimum *edit operations* to transform the first string to the second string.The edit operations are inserting a character, substituting a character or deleting a character.For example, if the developer wrote <hml> instead of <html> the edit distance between them is 1; our tool will able into transform <hml> to <html>. To

calculate the edit distance between two strings we need to use dynamic programming; this algorithm is discussed by Manning, Raghavan, and Schutze(2008). In addition to the algorithm, we need to use a set as a data structure to store all the HTML keywords. We need to check if a tag or attribute t of the HTML document is found in the set. if it is in the set, it is a correct keyword; if not we need to compute the edit distance t with all the keywords in the set and we need to transform t to the keyword that have the smallest edit distance. If we have more than one small equal edit distance with different keywords, we need to suggest to the developer the different possibilities.

Algorithm 1 Correction of HTML Keyword

Require: HTML document, Set s, Stack s_1
  while loop till the end of the HTML document **do**
    t is an HTML's token
    if t is not found in the set s **then**
      while loop each HTML's keyword t_1 of the set s **do**
        min = EditDistance(t,t_1)
        if we have more than one min **then**
          push min into the stack
        else
          we have only one min
        end if
      end while
      if s_1 is empty **then**
        correct t acording to the $t1$ with min edit operations
      else
        suggest to the developer $t1$ with the same min edit operations from the stack
      end if
    end if
  end while

Algorithm 2 Edit Distance

Require: String s and String s_1
  int $m[i, j]$ = 0
  for i = 1 to $|s_1|$ **do**
    $m[i, 0]$ = i
  end for
  for j = 1 to $|s_1|$ **do**
    $m[j, 0]$ = j
  end for
  for i = 1 to $|s_1|$ **do**
    for j = 1 to $|s_2|$ **do**
      $m[i, j]$ = min $\{$ $m[i-1, j-1]$ + if($s[i] = s_1[j]$) then 0 else 1,
      $m[i-1, j] + 1$, $m[i, j-1] + 1\}$
    end for
  end for
  return $m[|s|, |s_1|]$

B. Correction of Missing Tag

This phase of the tool is responsible to check for missing tags in the HTML language. Our approach is based on the HTML parser algorithm proposed by Moller and Schwarz (2011). The algorithm uses a stack as a data structure.The stack stores the open tag elements.For example, each time the parser finds a new open tag like <*p*> it will push it into the stack.The algorithm will continue scanning until the end of the HTML document.When the algorithm finds the closing tag </p> it will pop the first element of the first stack.If the algorithm finds a different closing element, that does not match the first element of the stack; the algorithm will add the appropriate closing tag and repeat the process of scanning from the beginning of the HTML documennt to check if the HTML document is valid. If the parser finds an ommision of the start tag it will add the open tag element and repeat the scanning. Adding open and close tag elements in the HTML document is based on the DTD of the HTML language.

Algorithm 3 Missing Tag Correction

Require: DTD, HTML document, Stack s
  push the first the token <html> into s
  while loop till the end of the HTML document **do**
    t is an HTML's keyword
    if t is an open tag **then**
      push t into s
    else if t is an close tag AND it matches the first element in s **then**
      pop the first element from s
    else if t is an close tag AND it does not match the first element in s **then**
      add the appropriate close tag by checking the DTD
      retry the parsing with the new HTML document
    else if we have an ommission of the open element AND we have the close element **then**
      add the appropriate open tag by checking the DTD
      retry the parsing with the new HTML document
    end if
  end while

CONCLUSION

We reviewed some papers that talk about HTML correction, but most of them did not take into consideration the correction of misspelled HTML keyword. We proposed an HTML parser that could create a new HTML document by correcting or suggesting to the developer errors that were found by our tool. Our proposed parser was not yet created because of the time constraint. We are thinking in the future to build our suggested parser by adding some improvements to the three algorithms we talked about in this paper.

REFERENCES

[1] A. Grosskurth and M. W. Godfrey, "A reference architecture for web browsers," in *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*. IEEE, 2005, pp. 661–664.

[2] M. Schwarz *et al.*, "Html validation of context-free languages," in *Foundations of Software Science and Computational Structures*. Springer, 2011, pp. 426–440.

[3] Wikipedia, "Comparison of layout engines (html5) — wikipedia, the free encyclopedia," 2014, [Online; accessed 12-April-2015]. [Online]. Available: http://en.wikipedia.org/w/index.php?oldid=640303446

[4] H. Samimi, M. Schäfer, S. Artzi, T. Millstein, F. Tip, and L. Hendren, "Automated repair of html generation errors in php applications using string constraint solving," in *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 2012, pp. 277–287.
[5] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1.