DOCUMENTATION

HTML Parsing

Advanced Algorithms

Louis ARCELON, Théo VOREAUX, Raphaël KERVAON and Rémi PICHOT

HTML Parsing

Introduction :

What the program should achieve:

1. Analysing input HTML code, identifying tags, attributes and their structure.
2. Reporting structural errors such as incorrectly closed tags, missing tags, etc.
3. Providing a clear and understandable error report, indicating the exact locations of errors in the source code.

Roles :

Louis ARCELON : Helmsman
In charge of keeping tracks of the process, along with task distributing

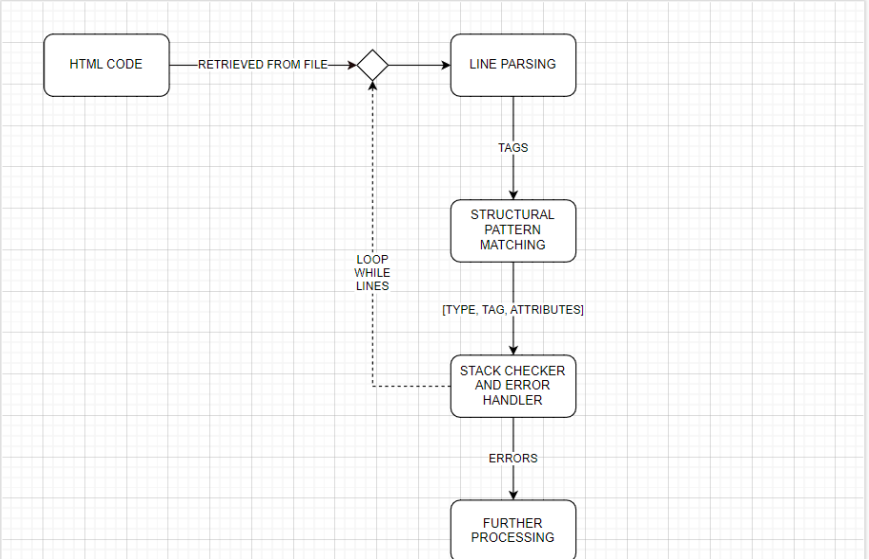
Raphaël KERVAON :
He doesn't understand

Théo VOREAUX :
Test case analysis (Wikipédia)

Rémi PICHOT : Spokesperson
Explain everything at the end

Global idea of the code:

Overall, this code will attempt to parse HTML code line by line.
The first part of the program will read line by line the given file and extract the tags. It will check for errors such as nested tags (tag opened inside one another), unclosed and tags that were never opened.
Then from this list of tags, we can first deduce their nature with python's structural pattern matching, then the program will use a stack to check whether the html structure is valid across the file.
At the end, this code will report errors such as unmatched tags or unclosed tags and report to the user the error and where the error is located.



1. Check_tag_inline: (data harvesting)

In this function, the code will check the tag-related errors possible such as tags that were never closed or opened with “<” and “>” characters, and nested tags.
We will check the tags for each line, if we find ourselves in a tag the code will “analyse” it and stock the tags alone (without the content in between the tags). This function will return a list of strings . Returning a list of tags as strings allows further processing of the output, and specifically feeds them into ‘check_tag_content()’ to get their nature. This is what will allow the structural analysis further down the line (in Check_tag_content).

Input : “<head id=”test”></head>”
Output: [“<head id=’test’>”, “</head>”]

2. Check_tag_content: (data identifying)

This function will retrieve a tag and check if it’s an open or closed one (or a comment).
It uses the keyword “match” in python to compare the structure of the tag against different patterns, it then returns the corresponding type (OPEN, CLOSE or COMMENT).

We verify every case in a specific order to ensure that no mismatch happens : from the most specific to the most global one (like an upside down funnel).

Input : “<head id=’test’>”
Output : [“OPEN”, “head id=’test’”]

3. Global_parsing: (data analysis)

This function will use a wrapper to prevent the program from crashing according to Louis. (but I will not explain it)

This function will parse the HTML file we have given it,
It will check what type of tags it comes across, analyse it if it's an OPEN or CLOSE tag. It utilises a stack data structure to keep track of open tags and when it comes across a corresponding closing tag it will stop keeping track of it.

It will also check for mismatching closing tags (if the opening tags don't correspond to the closing tag).

Problem with this version :

This version 1 is not capable of handling self-closing customised tags.

From what I understood from what Louis has explained to me, the program doesn't specifically check if the tags are "correct" when it comes to their identifier. Instead, if the tags are not in the "self-closing" list it will always await a closing tag further down the line. But in the case of self-closing custom tags (or unspecified self-closing tags) it will await closing tags that will never occur. Hence it is bound to find a mismatch when there should be none.

I thought about a solution though I'm not sure it could work. At the end we will end up with a mismatch error. We create a list of every possible tag. We can then compare it to the list created before. If there is no match, we can therefore conclude that it's a customised tag and treat it separately (delete it from the list and consider it as never closed tag)

Another problem with this version is that in a HTML code, we can find JS code or another language which doesn't use the same data structure as HTML and therefore provoke a false error.

One solution I thought that could work is that we will have to ignore content in between style or script tags.

And the last problem for the code to work is that the opening and closing tags would have to be on the same line.

The solution I came up with is that the code should not analyse line by line but character by character

TEST :

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Void's Website</title>

    <link href="style.css" rel="stylesheet" type="text/css">

</head>

<body style="margin:0;">

    <!------------SCRIPT PART------------->

    <script src="script.js"> </script>

</body>

</html>
```