Webpage Recognition using ANTLR

Python 3.10

Alexander Popov - 4776313
Viktor Kurtev
AUT
13747
23.06.2024

Objective

The purpose of this project is to find out the possibility of building a tool that recognizes the topic of a webpage, based on the html input it receives. The input for this tool should be HTML code. Based on this input and the generated tree from ANTLR, the logic will recognize whether or not a webpage is about a specific topic. Such a tool could be of great importance to a lot of people. A big benefit that it could bring is time-saving. Because of the functionality of such a tool, it would be easy to copy the HTML code and paste it as input, in order to determine if the webpage is related to a desired topic or not. All of this would be possible without even reading a single word, and that saves a lot of time. One could go through a lot of webpages fast, until finding material of the desired topic. An example scenario where such a tool could have been useful and we have experienced personally is searching for internships. All internships have a description that sometimes can be very long and not of great use. One can spend hours looking through a couple of internship posts and then finding nothing useful. Instead, using such a tool can help with finding the appropriate post, and then instead of wasting hours searching for the right post, one can quickly find the right position, and instead, he/she can spend that time preparing for the post.

Keywords

Python | Automata | ANTLR | Web | HTML | Web scraping

Introduction

Web page recognition can be done with the help of language recognition tools as in its core web pages consist mainly of three types of files: HTML, CSS and javascript. We are going to concentrate on HTML parsing. For this, we are going to use ANTLR (ANother Tool for Language Recognition). The complex structure of a web page consists of nested elements, scripts, links, images… By defining a crude version of this structure we can extract the required data with the use of ANTLR.

Parr, T. (2013). *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf.

Materials and Methods

We are going to use Python version 3.10. We chose this language for its simplicity of use and popularity among developers. This means that we are more likely to find help online in case we get stuck with a problem. Another reason is that we both have experience with it.

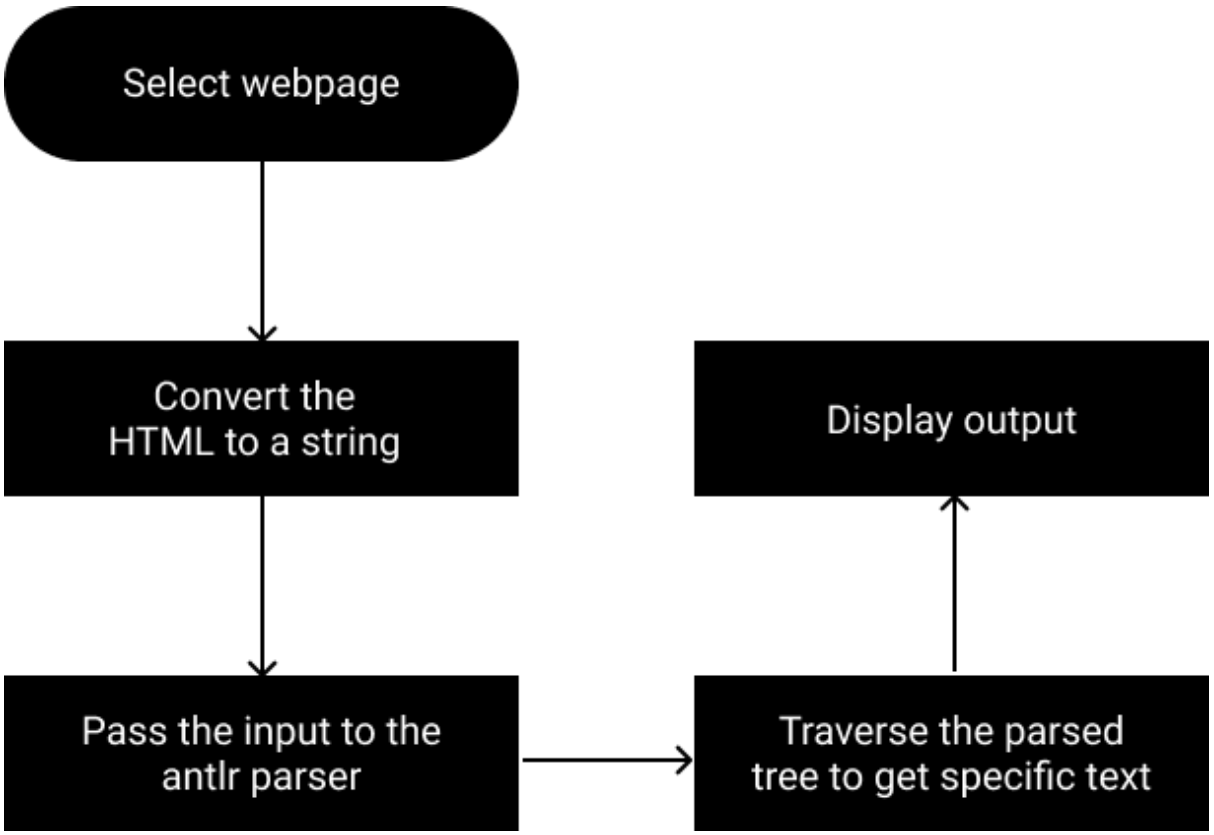

Figure 1

S no.	Title	Method	Description	Result Analysis
1	Strings	ANTLR Grammar Definition	Create grammar rules to recognize and parse string literals	Analyzes the development of precise rules for string recognition and parsing
2	Expression	ANTLR Grammar Definition	Create grammar rules to recognize and parse mathematical expressions	Evaluates the complexity and efficiency of grammar rules for mathematical expressions
3	Variables	ANTLR Grammar Definition	Create grammar rules to recognize and parse variable declarations	Examines the flexibility and accuracy of rules in handling different variable types
4	Conditions	ANTLR Grammar Definition	Create grammar rules to recognize and parse conditional statements	Assesses the robustness of grammar in processing nested and complex conditional statements

5	Functions	ANTLR Grammar Definition	Create grammar rules to recognize and parse function definitions	Evaluates the comprehensiveness of grammar in accurately identifying and parsing function constructs

HTML parse grammar description

We created a simple grammar for parsing general html elements and content.

Parser rules:

htmlDocument:

- This is the top-level rule for the grammar, representing an entire HTML document.
- It consists of zero or more element rules (element*) followed by the end-of-file (EOF)

element:

- Represents a generic HTML element, which includes a start tag (startTag), optional content (content), and an end tag (endTag)
- The content can either be another element or plain text (TEXT)

startTag:

- Defines the rule for an opening HTML tag
- It begins with an opening angle bracket <, followed by one or more alphabetic characters ([a-zA-Z]+) representing the tag name
- Optionally followed by zero or more attributes (attribute*)
- Ends with a TAG_END token (>)

endTag:

- Represents a closing HTML tag
- Starts with </, followed by one or more alphabetic characters ([a-zA-Z]+) representing the tag name
- Ends with a closing angle bracket >

content:

- Defines what can be inside an HTML element
- It can either be:
    - Another element (nested structure)
    - Or plain text (TEXT), which is any sequence of characters that does not include <, >, carriage return, or newline (~[<>\r\n]+)

attribute:

- Represents an attribute within an HTML tag
- Defined by one or more alphabetic characters ([a-zA-Z]+) for the attribute name, followed by an equals sign =, a double quote ", any characters (.*?) within the quotes, and then another double quote " (ATTRIBUTE)

Lexer rules(tokens):

OPEN_TAG:

- Token for opening HTML tag (< followed by one or more alphabetic characters)

CLOSE_TAG:

- Token for closing HTML tag (</ followed by one or more alphabetic characters >)

TAG_END:

- Token for end of an HTML tag (>)

TAG_SELF_CLOSE:

- Token for self-closing HTML tag (/>)

ATTRIBUTE:

- Token for HTML attribute (name="value" format).

TEXT:

- Token for text content within HTML, excluding tags (~[<>\r\n]+)

WS:

- Whitespace token to be ignored ([\t\r\n]+ -> skip)

Result

With this defined grammar we are successfully able to parse a lot of different html content. It is not very complex, and it does not cover everything within the HTML structure, but it does provide a good starting point and a solid base to build upon. Such a grammar could effectively parse an html text structure into an asynchronous search tree, which could be used for different purposes. Some examples could be: listing every different element and its occurrence count, describing the structure of the webpage, describing the topic of it, and perhaps even building the webpage without using HTML. You can see the result in Figure 2 as we could create a parse tree using antlr and grun.

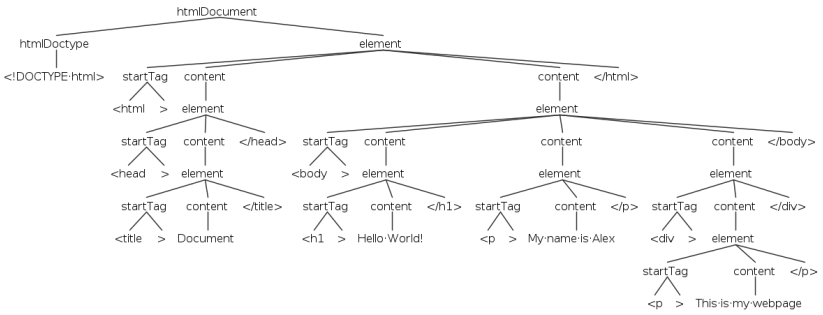

Figure 2

Discussion

During the lectures and assignments of this course, we gained valuable insight into the topics of automata and language parsing. The practice gained from the assignments will serve as guidelines on how to develop such a tool. Due to the limited time frame provided for this paper, the execution and development of the project was not possible. Therefore our future plans consist of further research and will mostly concentrate on the development and maintenance of the tool.

Conclusion

In this project, we explored creating a tool to recognize the topic of a webpage using its HTML input. We used ANTLR to define a grammar that parses HTML content into a structured format.

Our work showed that it is possible to parse and analyze HTML documents to extract useful information about their content and structure. Although our current grammar does not cover every aspect of HTML, it provides a solid starting point.

The knowledge gained from our lectures and assignments on automata and language parsing was essential in developing this tool. Future work will focus on expanding and refining the grammar and parser to handle more complex HTML structures.

References

fwouts. (2019, February 28). A quick intro to ANTLR4. *Medium*. https://medium.com/@fwouts/a-quick-intro-to-antlr4-5f4f35719823
Hardy, J. (n.d.). Lexical analysis. Retrieved from https://www.irisa.fr/caps/people/hardy/m1comp/doc/lexer.html
isetitra. (2020, July 12). Step-by-step creation of a simple compiler using ANTLR4. *Medium*. https://medium.com/@isetitra/step-by-step-creation-of-a-simple-compiler-using-antlr4-9285755cf943

Fontys
Alexander Popov
Viktor Kurtev