Ain Shams University

Faculty of Engineering

Computer and Systems Engineering Department

XML Editor Documentation

Section	Code	Names
2	1700572	زياد مصطفى عبد العزيز مصطفى
3	1700860	عمر جمال حامد عجمي
3	1700948	عمرو محمد فتوح احمد
3	1700957	فادي أحمد مصطفى علي النشار

Background

XML stands for Extensible Markup Language. It is a markup language which aims to represent data in a way to be readable for both humans and machines. XML is used to store data rather than displaying it like HTML.
XML can be used for many things like exchanging data between systems, storing and arranging data. It also can be used to simplify creation of HTML documents. Any type of data can be represented using XML.(1)
XML is also used for better web search and automated tasks as it can restrict search results to a portion of results based on the type of search term due to its extensible property (i.e. you can write any word representing your data in its tags.(2)
XML parsers aim to convert the XML files into readable code after validating the written syntax. They can also do some operations on the XML files such as minification, prettifying, conversion to another data representing format, etc. (3)

Implementation Details

Data Structures

Due to the nature of XML files, the most important data structure used is the Tree data structure as it is the most similar data structure to the structure of XML. Using Tree data structure with the XML requires making a new data structure called Tree Node which represents the building blocks of the Tree data structure. Other data structures used are to represent the whole file at its different stages like the file being an input file or processed file or output file.

Tree Node

This is the building block of the tree data structure. It represents a single tag or a comment or a text in the XML file. It contains (value) property that holds the name of the tag or the content of the comment or text. It also contains vectors to hold its children Tree Nodes, attributes of a tag. It also contains boolean properties to indicate whether it is a comment or text or a tag.

Tree

This is the main data structure used to parse the XML file. It contains a single property which is a single tree node that holds the rest of its children tree nodes.

Input File

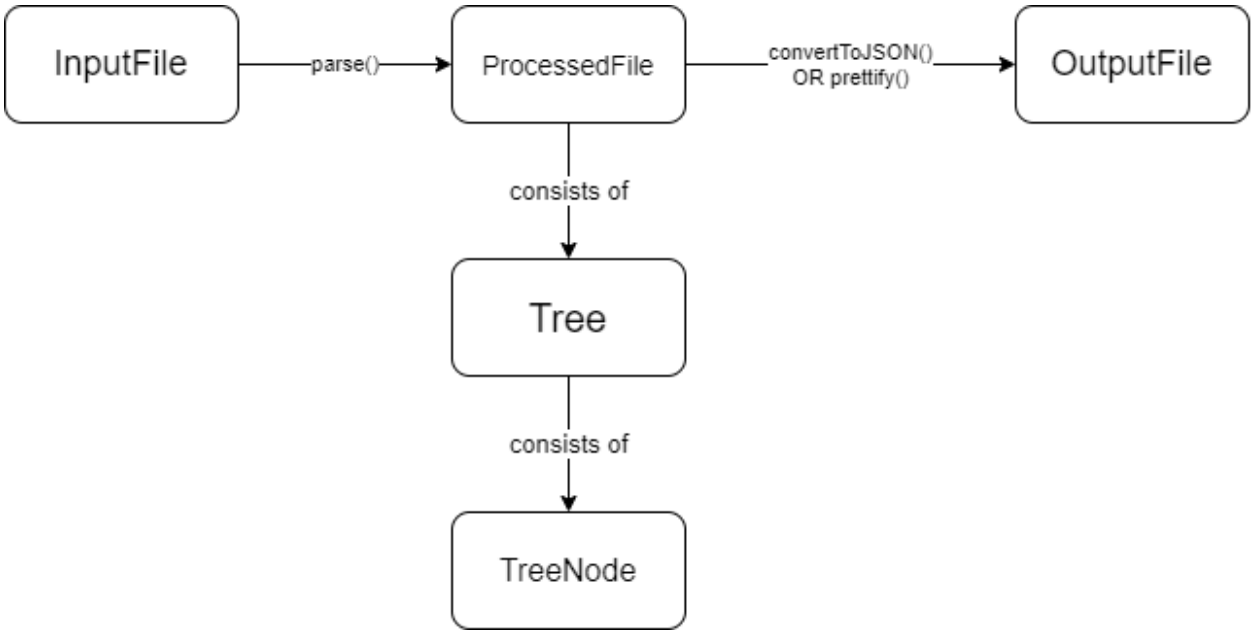
This is a data structure which represents the file which the user chooses at the beginning. It holds the file content in a property and has another property which holds the status of checking the file contents against the XML standard syntax rules.

Processed File

Represents an input file after processing using parse function. It consists of the XML tree and a group of XML declarations and comments.

Output File

Represents the file in the state before viewing to the user. Usually the functions which produce the final output viewed to the user deal with that data structure and their output is stored in it



Figure(1) shows the relation between classes in the project

Huffman Node

It's used in compression operation, as we use Huffman lossless compression technique.(4), it stores the value and its frequency in a text and has 2 pointers for the left and the right nodes.

Huffman Tree

We use a minimum heap to build a binary tree, in which at first all the nodes which hold the characters are leaf nodes then we pop from the priority queue to build internal nodes until we have one remaining node in the priority queue.

Implementations

Minifying

In the minifying function, we pass on the whole string removing all unnecessary spaces, tabs and newlines used for beautifying and readability so that the size of the file decrease

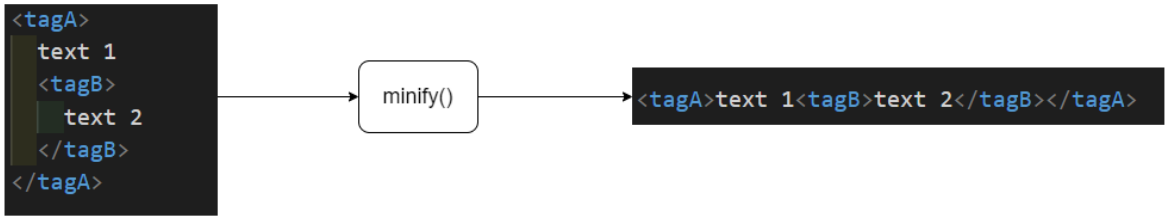
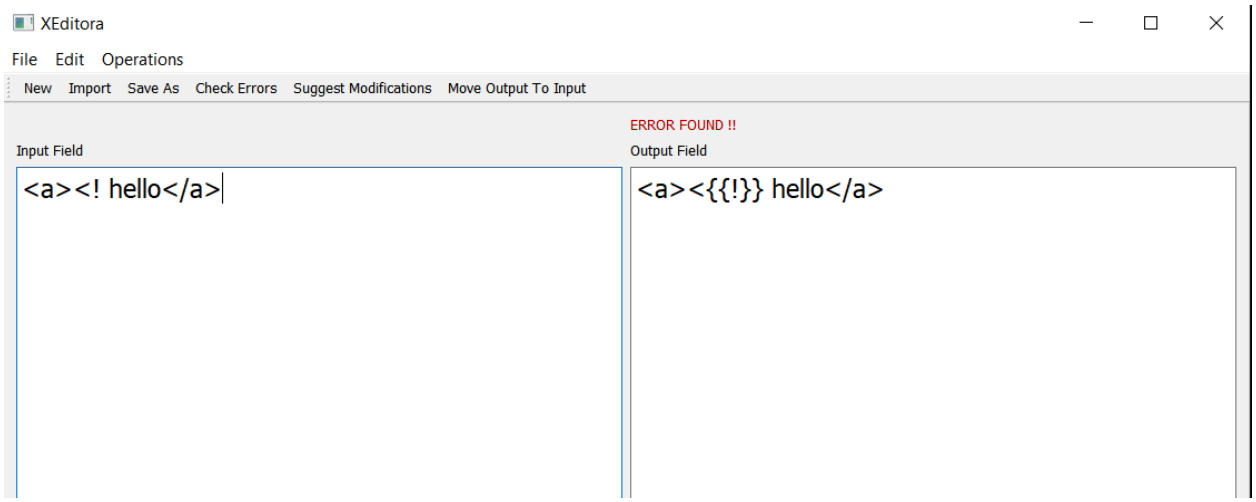


Figure (2) shows minification process

Checking

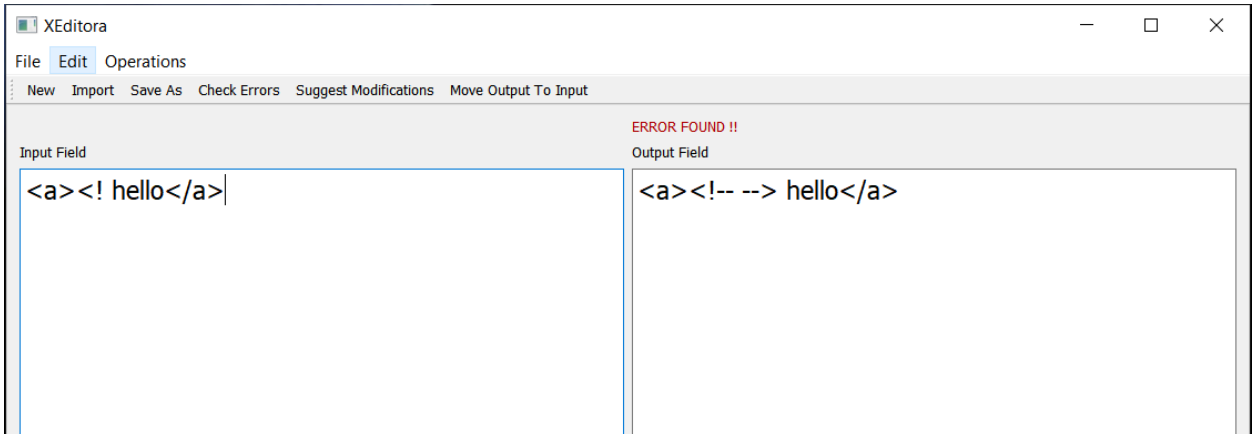
We check the input XML from the file and compare it against the standard rules of XML syntax. We mark the errors we find one by one



Checking example

Correction

This feature tries to make the XML code free from syntax errors. It tries to fix syntax errors especially those related to comments and closing tags. For example, it matches the mismatched opening and closing tags, close open comments, etc.



Correction example

Parsing

This function aims to convert the string representing the XML file to a tree structure to be easily processed. Other functions depend on the output of this function like prettify and convert to JSON function where they depend on the tree structure output by this function

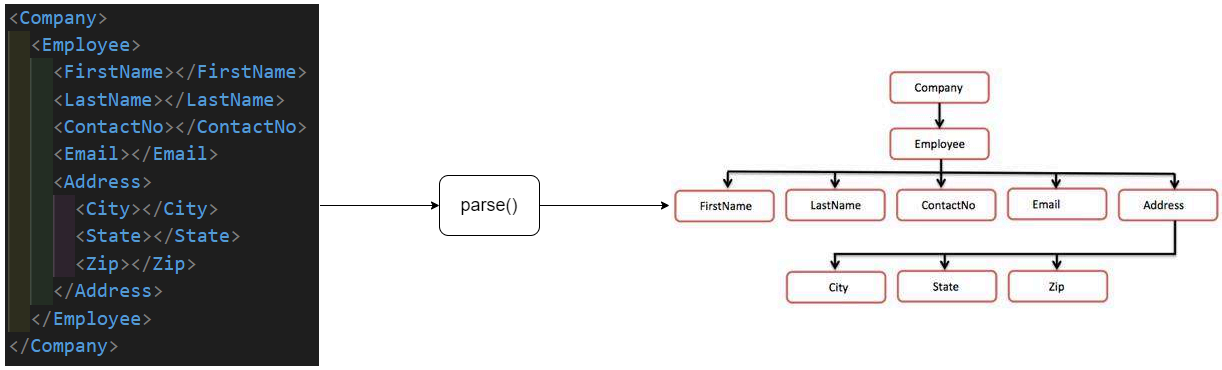
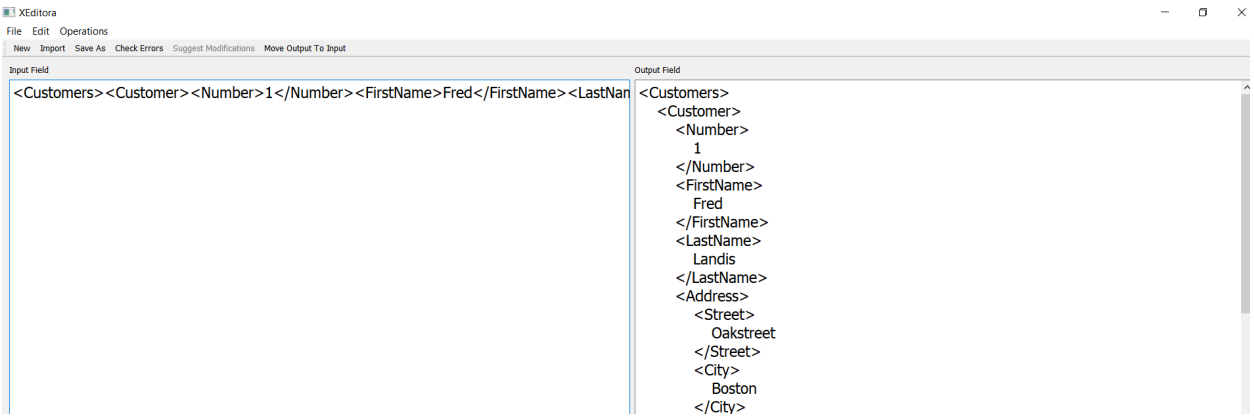


Figure (3) parsing process

Prettifying

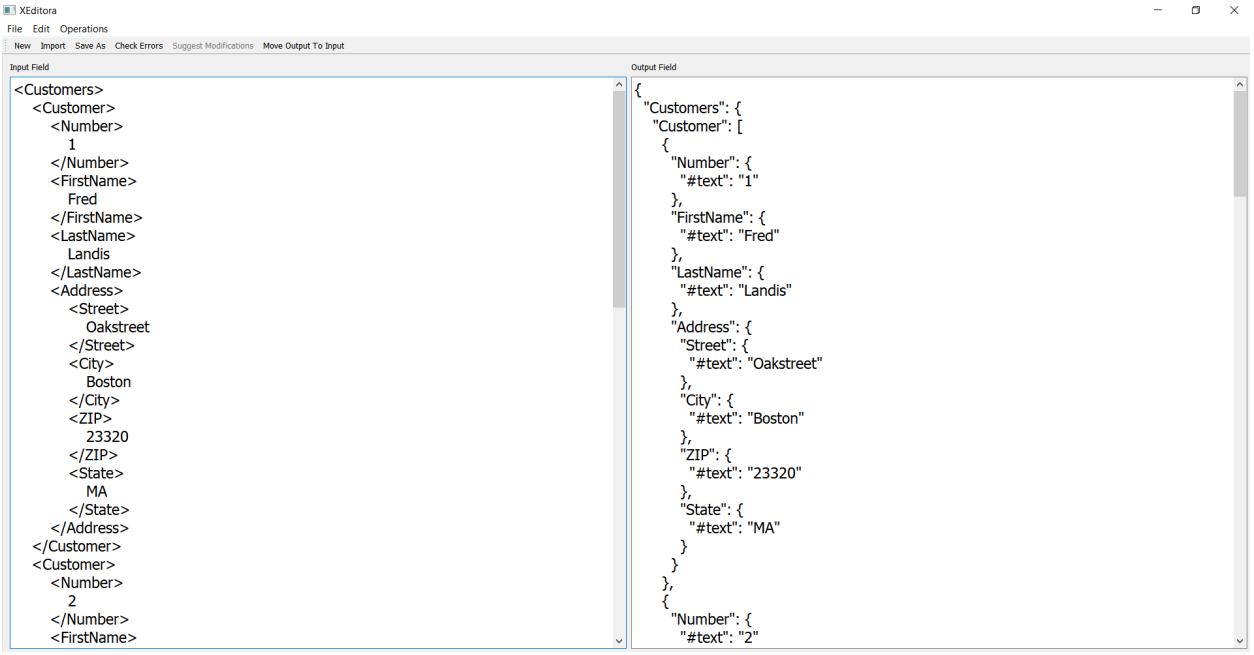
This function takes an XML string and outputs a prettified version of that string. It uses the parse function first to convert that string into a tree. It then works on that tree to reprint the XML string in a prettified way.



Prettifying example

Conversion to JSON

This function also depends on the parsing function and works on the output tree to print the corresponding JSON to the input XML file. Conversion from XML to JSON follows standard rules which can be found on a website in the references section.(5)

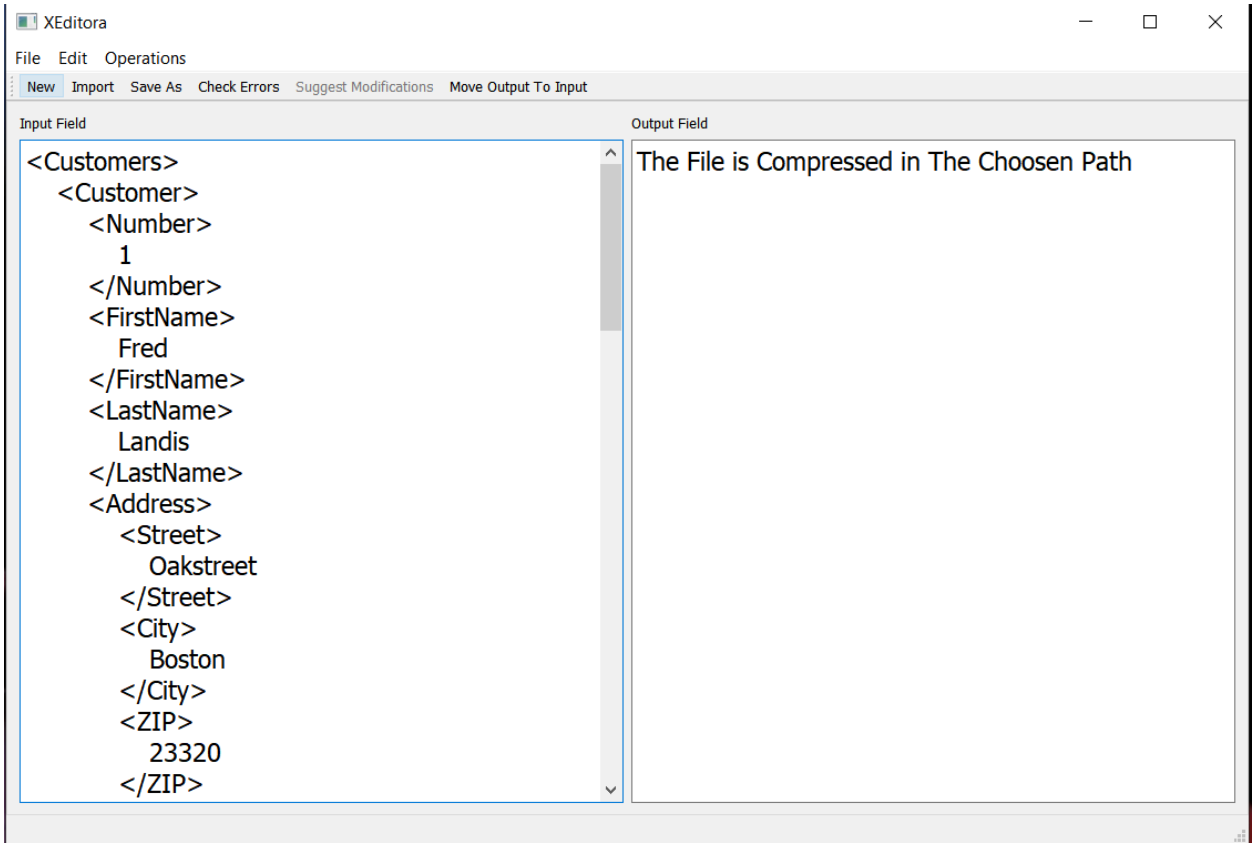


Conversion to JSON example

Compression

This function takes the text needed to be compressed and the file path to save the output, its output is a compressed file and another frequency table file used in decoding, the function checks that the text is not empty then it creates Huffman tree which

encodes the characters then encodes the whole text and saves it to a new file with frequency table file.



Compression example

UI

The GUI is implemented in C++ Qt Framework.(6), A simple UI is designed for easy access to the main features of the editor, where there is a menu bar which contains 3 main menu buttons (File, Edit and Operations).

- File menu: contains New, Import and Save As buttons.
- Edit menu: contains Move Input to Output button.
- Operations menu: contains Check Errors, Suggest Modifications, Prettify, Convert To JSON, Minify, Compress and Decompress buttons.

In addition, there is a toolbar for frequently used features which are:

- New: used to clear both input and output fields.
- Import: used to import file content to the program.
- Save As: used to save the Output Field content to a new file.
- Check Errors: used to check for XML errors on the content of Input Field.
- Suggest Modifications: used to suggest modifications on some errors which may occur.

- Move Output To Input: used for quick cutting the content of the Output Field to the Input field to do other operations on it as minification, prettifying and XML to JSON conversion.

Also, There are 2 separate fields (Input Field and Output Field) to show the data after each operation.

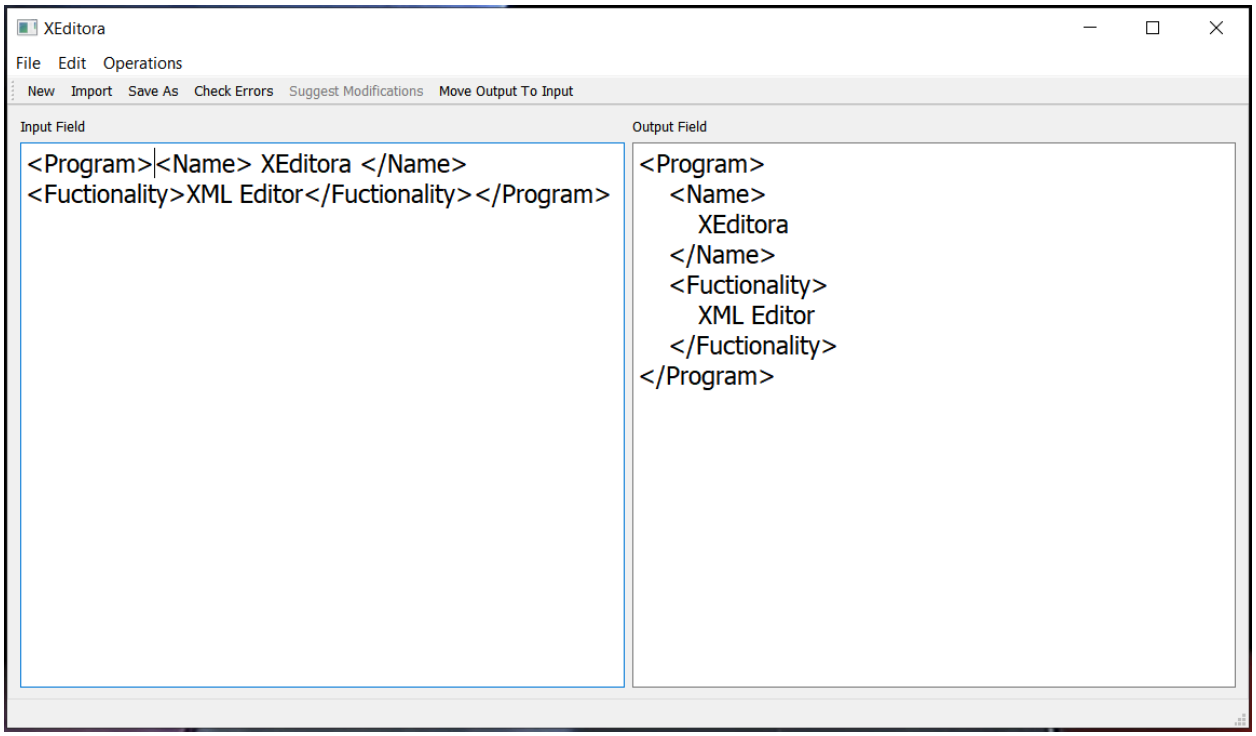


Figure (4) program UI

Complexity of Operations

In this section we discuss the analysis of complexity of operations in each major function in the program.

Note: complexity of find function is usually constant as the searched term is usually a few characters away from the index we start searching from, so it's usually ignored during complexity calculations.

Minifying

The function iterates over the entire string looking for (><!?) and removes empty spaces around them.
O(fileStringsize)

Checking

recCheck is a recursive function that iterates over number of valid attributes
and is invoked every opening tag
[O(number of opening tags * number of valid attributes)]
tagCheck function has recCheck included and iterates over every single character of the input string (file string)
O[characters in string +(number of opening tags * number of valid attributes)]

Correction

This function depends on checking and does additional operations that take constant time. O[characters in string +(number of opening tags * number of valid attributes)]

Parsing

The function iterates over the declarations, pushing them into their vector.
[O(declarations)]
The function also iterates over the comments, pushing them into their vector.
[O(comments)]
The function iterates over the tags of the XML file to store them, it also iterates over the attributes inside each tag. [O(tags * attributes)]
Total complexity: O(declarations + comments + tags * attributes)
Typically, tags dominates, so complexity equals O(nodes * attributes)

Prettifying

The function iterates over the declarations in the XML file. [O(declarations)]
The function also iterates over the comments before and after the XML tree.
[O(comments)]
The function iterates over all the nodes of the tree and for each node, it iterates over its attributes. [O(nodes * attributes)]

The overall complexity is the sum of all the previous complexities which equals O(declarations + comments + nodes * attributes)
In a typical XML file, the number of nodes dominates so the complexity would be O(nodes * attributes)

Conversion

The function “concatinate_childrens” that combines similar tags together into a vector, takes [O(number of nodes ^ 2)].
The function “toJson” that ,it takes [O(toJson)] * [O(concatinate_childrens)] = [O(tags * attributes)] * [O(number of nodes ^ 2)] = [O(n ^ 3)].
The function “string_to_json” that takes xml string and return json string ,it takes [O(parse)]+[O(toJson)]= [O(declarations + comments + tags * attributes)] + [O(n ^ 3)] = [O(n ^ 3)].

Compression

Compress function
Complexity of createHuffmanTree = O(chars) + O(n*freqTable)
Complexity of getEncodedString= O(chars)
Complexity of storeHuffmanFreqTable= O(freqTable)
Total complexity: O(chars) + O(n*freqTable)

Decompress function
Complexity of readHuffmanFreqTable = O(chars) + O(n*freqTable)
Complexity of createHuffmanTreeForDecoding= O(n*freqTable)
Complexity of getDecodedString= O(chars)

Assumptions

During project design, we assumed some behaviour to our functions in certain situations, here is a list of these assumptions.

Checking

- All inputs are in English Letters
- Version tag is checked same as attributes
- colon is accepted in attribute name
- allow <? ... to exist ?>

Parsing

- Xml file optionally begins with a group of declarations
- Each declaration begins with "<?" and ends with "?>"
- After that there can be a group of comments
- After that comes the root tag of the xml file
- Xml file can contain only 1 root tag
- Root tag can contain any number of sub tags, comments, text
- A tag can contain text, tag, and comment on the same level
- Comments can come after root node in xml file
- Input should be correct

Prettifying

Text would be put in a single line by itself not between opening and closing tags.

Conversion

- We assumed that xml comments will be parsed to json in the comments list with the same order of xml file.
- Text in tags will be converted in “#text” list.
- Input should be correct.

Compression

- The given path does not require administrative privileges.

UI

- The check errors button is initially off until the user enters input in the input field or imports a file.
- These buttons (Prettify, Convert to JSON) are initially off until the user presses the Check Errors button.
- Suggest Modification button is initially off until there is an error found by Check Errors.
- Save As button saves the content of the Output Field.
- Errors are shown sequentially and are removed after modification and repress the Check Errors button.
- When an error is found it is distinguished by putting the error between {{ }}, also an error notifier is shown above the output field.

References

(1) https://www.tutorialspoint.com/xml/xml_overview.htm
(2) https://www.ibm.com/docs/en/i/7.3?topic=introduction-uses-xml
(3) https://www.tutorialspoint.com/xml/xml_parsers.htm
(4) huffman
(5) https://www.xml.com/pub/a/2006/05/31/converting-between-xml-and-json.html
(6) https://doc.qt.io/