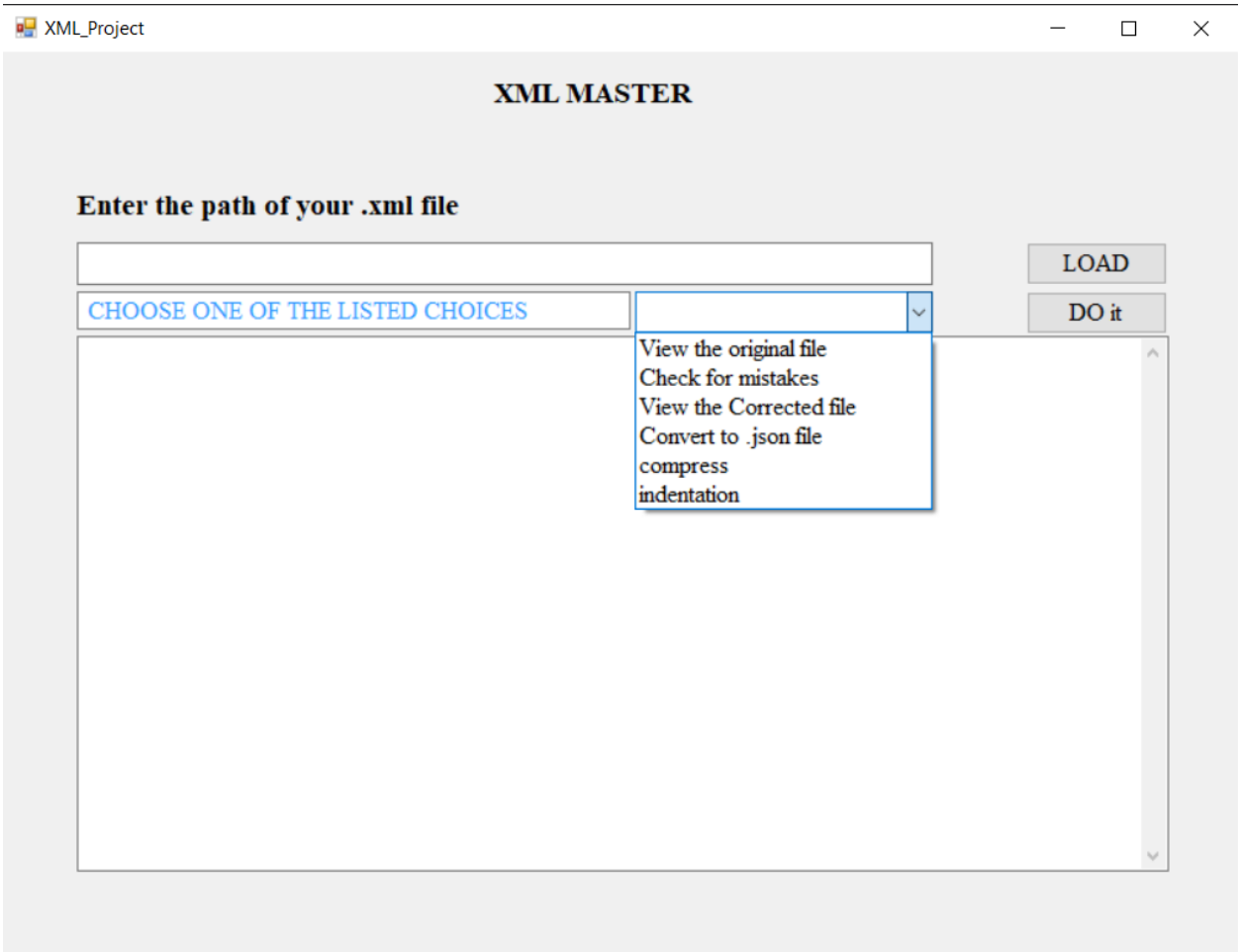
Computer and Systems Engineering

Data Structures and Algorithms (CSE331)

(Project report)

1- Mohamed Khaled Mohamed Ahmed Ghoz. 1802656
2- Ali Abdelfattah Shehata Ibrahim Ahmed. 1803168
3- Ahmed Khaled Mohamed Esmail ElGazzar. 1805106
4- Ahmed Fakreldin Sayed Ahmed. 1809285

GUI:



The GUI consists of a space to enter the path of the XML file to be processed then we press the LOAD button to load the file before choosing a specified task to be performed on the file and those tasks are:

1- View the original file:

It displays the original file entered only.

2- Check for mistakes:

It displays the line number with the error and the numbers of errors in the XML file.

3- View corrected file:

It displays the corrected file.

4- Convert this file to JSON:

It converts a valid XML file to JSON.

5- Compress the file:

It compresses the XML file and displays it on screen.

6- Indentation:

It forms the XML file properly with indentation.

Data structures used:

We used a tree data structure to store the XML tags to be able to traverse it.

We used stack data structure to push the XML tags and check correctness of the file.

We used vector in various scenarios in code.

functions used:

getTreeNode():

It converts the XML file to a tree and it uses some other functions to obtain its use and these functions are:

sys_std() , removeSpaces(), getTag(), erase().

This function makes the first tag in the file the root and each tag it contains one of its children and the information between the tags are the leaves.

This function makes it easier for the program to deal with the XML file by storing it in a tree and when you need to read the file you can traverse through the tree.

Iscorrect():

It checks if the XML file is valid or not and if there are any errors its checks them and displays the number of errors and the line with the error. It also corrects the XML file with errors and save it in a file. The function reads the file line by line and then it searches for opening or closing tags in the line, when it finds an opening tag it pushes that tag into a stack and if it's a closing tag it compares it to the top of the stack, If the top of the stack equals the closing tag it prints the closing tag in the file. If the closing tag and the top of the stack aren't equal it prints the closing tag in the top of the stack and then pops it and repeat that every line. When the file ends the function checks the stack, if the stack is empty then it returns if the stack is not empty the function prints the closing tags in the stack until the stack is empty then returns. If there is a closing tag while the stack is empty the function ignores it.

This function is a general case for all the XML files but it contains a special case for the given sample to print the closing tags in their places in some cases.

xml_to_JSON_converter():

it converts the tree obtained with getTreeNode() function into JSON format using tree traversing using recursive calls, it checks if the node is the root in this case it opens a "{" ,calls its children then after returning it closes with "}".

Then it checks if the node is a leaf node or not, if it is a leaf node it checks if the node is unique or there are other siblings nodes that has the same tag name, if there are nodes with the same tag names it opens an array and the last node with the same tag name closes it.

If the node is not a leaf node it checks if the node is unique or has the same tag name as other nodes, if it is unique it opens "{" and calls for children then closes with "}" when it returns, and if the node has the same tag name as other siblings it opens an array after "{"and the last node with that name closes it before "}".

Every node that is not the only node and not the last node should put "," after it.

xml():

It is used to convert the XML tree obtained with getTreeNode() function into an XML file with proper indentation.

Every node type it's name between "<" and ">" , calls for its children then close the tag with "</" and ">".

compress():

It decreases the size of the XML file by removing the new lines and spaces in the beginning of the lines and replacing the tags with symbols to decrease the number of characters in the file. This function reads the file line by line, replaces

the tags by symbols, removes the spaces and then print the line in a new file with all the lines in one line.

graph ():

In this method we loop through the file and get it line by line and for each line, we loop through it, character by character.

when it finds a closing tag, it gets the tag and store it in a string, and by that it is able to build a 2d Array that represents the adjacency matrix of the graph that shows the relations between the users in the xml file.

To visualize the graph, we used a library called ***"Allegro"*** which is a software library for video game development. The functionality of the library includes support for basic 2D graphics, image manipulation and text output.

Time complexity:

getTreeNode():

it loops over the file reading character by character and after getting a line it removes spaces using a while loop.

So, if S stands for the size of the file and L for the length of the line the function will have O(S*L).

sys_std():

It has O(1).

removeSpaces():

it has O(n) where n is the length of the string.

getTag():

It has O(n) where n is length of the string.

erase():

It has O(n) where n is the length of the string.

Iscorrect():

It has a nested for loop with overall time complexity of $O(S * L^2)$, where S is the size of the file and L is the length of the line.

xml_to_JSON_converter():

It traverses the tree so it has O(n) where n is the number of tags in the XML file.

xml():

It traverses the tree so it has O(n) where n is the number of tags in the XML file.

compress():

it has $O(S*L^2)$ where S is the size of the XML file and L is the length of the line.

graph ():

it has O(S) where S is the size of the XML file (No. of characters including spaces, new lines)

References:

1. https://docs.microsoft.com/en-us/dotnet/api/system.io.streamreader?view=net-6.0
2. https://docs.microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=net-6.0
3. https://www.cplusplus.com/reference/string/string/replace/
4. https://docs.microsoft.com/en-us/cpp/dotnet/walkthrough-compiling-a-cpp-program-that-targets-the-clr-in-visual-studio?view=msvc-170

GitHub repo link:

https://github.com/MedoGhoz/data-structures-project.git