Project

Phase 1&2

Data Structures and Algorithms

Name	ID
Omar Mohamed Diaaeldin Ibrahim Elsayed	1802932
Ahmed Magdy Fahmy Mohamed Ahmed	1805862
Ahmed Magdi Mostafa Hosni	1808714
Ahmed Ayman Abd-Alaziz Sharf	1806171

Report

- GitHub Repo Link
- Drive Link (includes video + pdf file + source codes)
- GUI Option 1 Video Link For Phase 1
- GUI Option 2 Video Link For Phase 1
- GUI Option 1 Video Link For Phase 2

Name	ID	Contribution
Omar Mohamed Diaaeldin Ibrahim Elsayed	1802932	PDF Formatting, XML.Py (collecting all files in one for GUI), GUI Option 2
Ahmed Magdy Fahmy Mohamed Ahmed	1805862	Format and Fix XML,convert to json , XML_tree (used in converting to JSON)
Ahmed Magdi Mostafa Hosni	1808714	Encode/Decode – Validate XML (check and fix errors)
Ahmed Ayman Abd-Alaziz Sharf	1806171	GUI Option 1 / Graph

1. Introduction:

In this project, we will make an XML parser that checks consistency (validation) of an XML file (whether it is encoded or a normal XML file) to check the validation of an XML file , if it has errors in it , produce an error with the tag name , line number , and the type of error. Then fix it and format the XML file, by adding the required indentation spaces to make the file in the appropriate hierarchical XML file format.

Second, if the user wants to convert it to JSON format, pass the file after validation and formatting, convert it to JSON using generic <Tree> algorithm approach (general case).

Third, if the user wants to compress/decompress the file after validation,

produce an encoded/decoded file

```
Encoded file is made of two parts head and body
head contains all unique tags separated by ; and terminated with :
ex : users;user;id;followers;:
then the body:
    suppose we use the following xml :
        <users>
            <user>
                <id>50</id>
                <empty></empty>
                <name>Ahmed</name>
            <user>
        </users>

    and head is as follows (not in order of appearance and will explain why):
        id;name;user;users;empty;:
        0   1   2   3       4

    then we get an encoding as follows:
        3>2>0>50<4:2>Ahmed<<<

        each tag replaced by its number
        > means we go down one level
        < means we go up one level (closing tag we then use stack to reconstruct it)
        : means we are on same level (happens with empty tags only)

    so full encoding of previous xml is :
        id;name;user;users;empty;:3>2>0>50<4:2>Ahmed<<<
```

In old validate (consistency) algorithm i was expecting a perfect tags list where least common tags take higher numbers so we get the best efficiency but since its no longer used we are using alternative function which will give us also good but not best efficiency least common tags will still get highest numbers but when u go down u don't always get best numbers but this won't be a problem at all and won't differ compared to old case until u have xml file with like over 18 different tags which is uncommon and even with 18 unique tags the difference in efficiency compared to old algorithm is significantly small so I didn't try to remake the old algorithm which is possible but will consume an O(n) time so I followed the concept of make common case fast

This encoding algorithm has a great advantage which is that it scales with files, the larger the file the smaller the result compared to original file because we replace all closing tags with single '<' and all tags with 2 chars 'i>' where 'i' is the index, it can be of 2 chars for tags with index > 15 bec we use hex number for index.

Finally, as a first option we made a desktop compatible application with the functionalities provided using Tkinter library across platform GUI toolkit.

We made it applicable with the user so that the user won't have to use console to run the program, export it as an .exe file to work with any other environment.

In addition, we make a bonus a "slider" and a window to show user's input file and another window to show output of each button also the user can save the file in the second window whenever he wants with small .exe file size and applicable interface.

As a second option we made a desktop compatible application with the functionalities provided using pyqt5 library across platform GUI toolkit.

We made it applicable with the user so that the user won't have to use console to run the program, export it as an .exe file to work with any other environment.

A. XML.py:

Class name: XML

Parameters:

file_name = “” ; // takes the file name as an input path

file_data=”” ; // represents the data inside the file

isValid=false; // checks if the file is valid or not

tags = []; // used in encoding and decoding

elements = []; // used in validation, formatting and converting to JSON

- **Function name:** __init__

Input: file_name = none, self //object of class XML

Output: None

Explanation: used as a constructor, assigning file_name after reading it to a variable called file, and then assign it to file_data

After removing spaces using function trim_spaces.

Time & Space Complexity: O(n)

- **Function name:** close_tag

Input: opening //string array initialized to take all data after closing tag.

Output: </ + opening //return closing element + data.

Explanation: defines closing tag of an element.

Time & Space Complexity: O(1)

- **Function name:** open_tag

Input: closing //string array initialized to take all data after opening tag.

Output: < + closing //return closing element + data.

Explanation: defines opening tag of an element.

Time & Space Complexity: O(1)

- **Function name:** is_array

Input: -elements // list of array data.

-element // an element to check whether it exists in the elements array or not.

Output: //return closing element + data.

Explanation: defines opening tag of an element.

Time & Space Complexity: O(n)

- **Function name:** is_singular

Input: -plural //expected plural

-singular //expected singular

Output: //return Boolean whether 2nd param is single of 1st

Explanation: Compares two strings (will trim any '<', '>' or '/') then check if 2nd param is singular of 1st , this will help is choosing correct position if we have a tag like posts and post, users and user, books and book, ….. etc.

Time & Space Complexity: O(1)

- **Function name:** found_b4

Input: -elements //elements list

-element //closing tag

Output: //return Boolean

Explanation: Loop through the elements array to check if opening of element is found in elements array .

Time & Space Complexity: T (n) = O(n), for n is size of elements, S(n) = O(1)

- **Function name:** validate

Input: -filename //name of the file

Output: //return tuple (elements, errors) where elements is the reconstructed xml elements, and errors is array of errors each error is a tuple of (line number, tag with the problem, type of error)

Explanation: validates the xml file and producing a fixed version

We are working in general but the validator will fix any error in it's perfect place except for strange tags they will be closed and ignored so we handle all cases except strange tags yet those tags are balanced too

It normally can fix multiple errors but when they are too much it sometimes gives balanced but inaccurate result and rarely can crash

Time & Space Complexity: O(n)

Next is validating test (it fixes too)

Before with errors

```
<users>
    <user>
    <id>1</id>
        <name>Ahmed Ali </name>
        <posts>
            <post>
                <body>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
                <topics>
                <topic>economy</topic>
                    <topic>finance </topic>
                </topics>
            </post>
            <post>
                <body>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
                <topics>
                    <topic>solar_energy</topic>
                </topics>
            </post>
        </posts>
        <followers>
            <follower>
```

After validating

```
<users>
    <user>
        <id>1</id>
        <name>Ahmed Ali</name>
        <posts>
            <post>
                <body>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
                <topics>
                    <topic>economy</topic>
                    <topic>finance</topic>
                </topics>
            </post>
            <post>
                <body>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
                <topics>
                    <topic>solar_energy</topic>
                </topics>
            </post>
        </posts>
        <followers>
            <follower>
```

- **Function name:** fix

Input: self //object of class XML

Output: return self.format_xml()

Explanation: calls function format_xml which in return produces a formatted xml file

Time & Space Complexity: O(n)

- **Function name:** validate_encoded

Input: self //object of class XML

Output: self.tags

Explanation: check validation of encoded files by putting all tags in an array called tags.

Time & Space Complexity: O(n)

- **Function name:** encode

Input: self //object of class XML

Output: result //encoded file with an extension .enc

Explanation: reduce size of the original XML file by getting rid of spaces and tags i.e. making it encrypted in a single crypted line.

Time & Space Complexity: O(n)

- **Function name:** decode

Input: self //object of class XML

Output: decoded //decoded file which represents the original file

Explanation: decrypting the output of encoding, by reversing the process made in it i.e. decrypting the single crypted line back to the original form.

Time & Space Complexity: O(n)

- **Function name:** scrap_data

Input: self //object of class XML

Output: elements //array of data (list) used in encoding, formatting, converting to json.

Explanation: takes an XML file as a string, then scrap the data from it into a list.

Time & Space Complexity: $O(n^2)$

- **Function name:** get_tags

Input: self //object of class XML

Output: result //array of data (list) which encapsulates the tags inside it which in turn is reversed.

Explanation: scrap the tags between < and > in the opening tags and then append it in the result array.

Time & Space Complexity: O(n)

- **Function name:** format_xml

Input: self //object of class XML

Output: result //formatted xml file in text form

Explanation: //just formats the file

Time & Space Complexity: O(n)

- **Function name:** to_json

Input: self //object of class XML

Output: json //converted xml file to json in text form

Explanation: //main function that will prepare the beginning and end of json file and start recursion for

Time & Space Complexity: O(n)

- **Function name:** int_or_str

Input: data //input data is in the form of integer or string

Output: data //returns data

Explanation: if input is int will be returned, otherwise if its string will be returned with " " around it .

Time & Space Complexity: O(1)

- **Function name:** array_check

Input: node //instance of class Node in xml_tree

Output: //Boolean

Explanation: returns true if parent of this node is array of this node like users is array of user otherwise return false

Time & Space Complexity: O(1)

- **Function name:** node_to_json

Input: node //instance of class Node in xml_tree

Output: string // part of xml converted to json

Explanation: the main function for creating json we handle 4 cases

1) Array of leaves like topic
2) Array of non leaves like user
3) Non array of leaves like id and name
4) Non array of non leaves like posts with single post

We use recursion with the cases of leaves are the base cases where recursion stops

Time & Space Complexity: O(n)

- **Function name:** trim_spaces

Input: text

Output: text after trimming all spaces

Explanation: remove all spaces In text to make easier for other functions to work with

Time & Space Complexity: O(n)

B. XML_tree.py:

Class name: Node

- **Function name:** __init__

Input: data , self //object of class Node

Output: None

Explanation: used as a constructor.

Time & Space Complexity: O(1)

- **Function name:** add_child

Input: child //any datatype , self //object of class Node

Output: None

Explanation: used to add children nodes to the existing nodes.

Time & Space Complexity: O(n)

- **Function name:** print_tree_elements

Input: level =0 // start of printing tree elements in indented, self //object of class Node

Output: None

Explanation: used to print xml tree elements. Recursive function used to add new elements.

Time & Space Complexity: $T(n) = aT\left(\frac{n}{b}\right) + O(n^d)$

A=1 , b=1 , d=1

So the time complexity is O(nlogn)

- **Function name:** add_child

Input: child //any datatype, self //object of class Node

Output: None

Explanation: used to add children nodes to the existing nodes.

Time & Space Complexity: O(n)

- **Function name:** get_depth

Input: self //object of class Node

Output: level //the depth of the tree measured from this node

Explanation: measures the depth of a tree from a certain node

Time & Space Complexity: O(n)

- **Function name:** create_xml_tree

Input: elements //list of all data

Output: root node

Explanation: takes xml file in the form of a list and use to create the XML tree

Time & Space Complexity: O(n) //n: size of elements list

GUI:

Option 1:(using tkniter)

C. GUI.py:

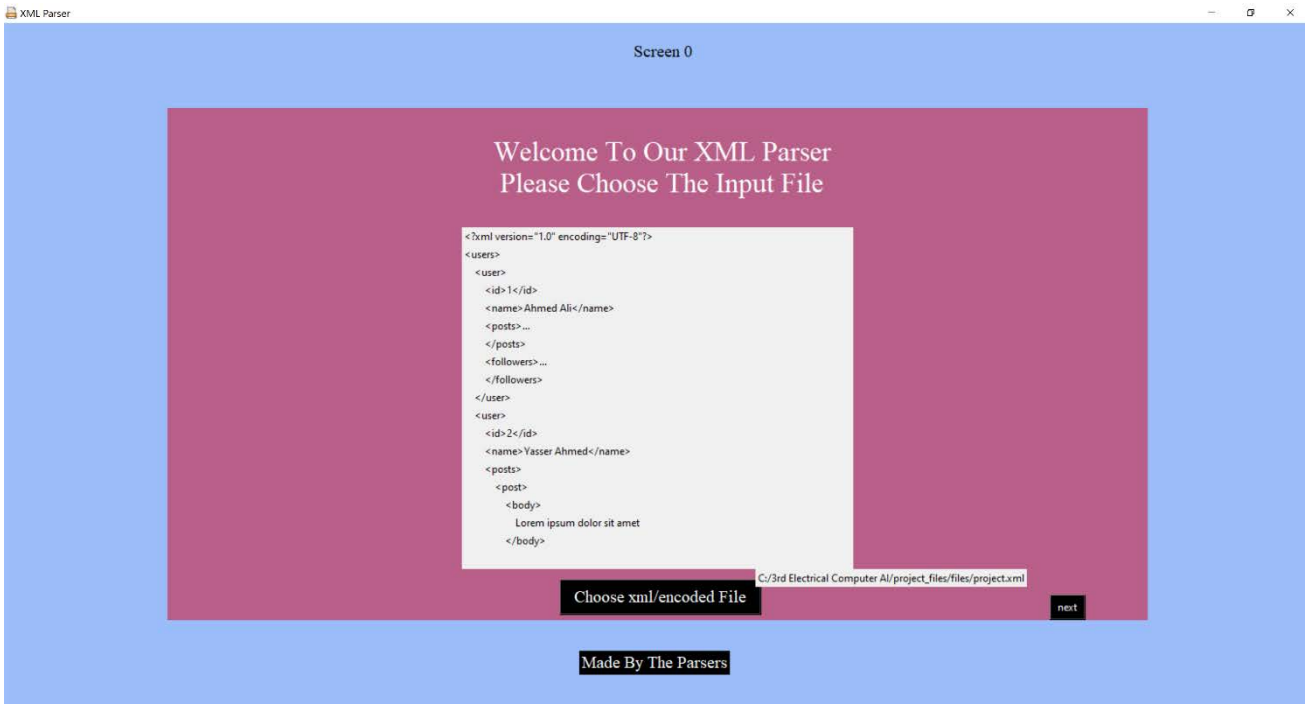

Figure 1:GUI Option 1

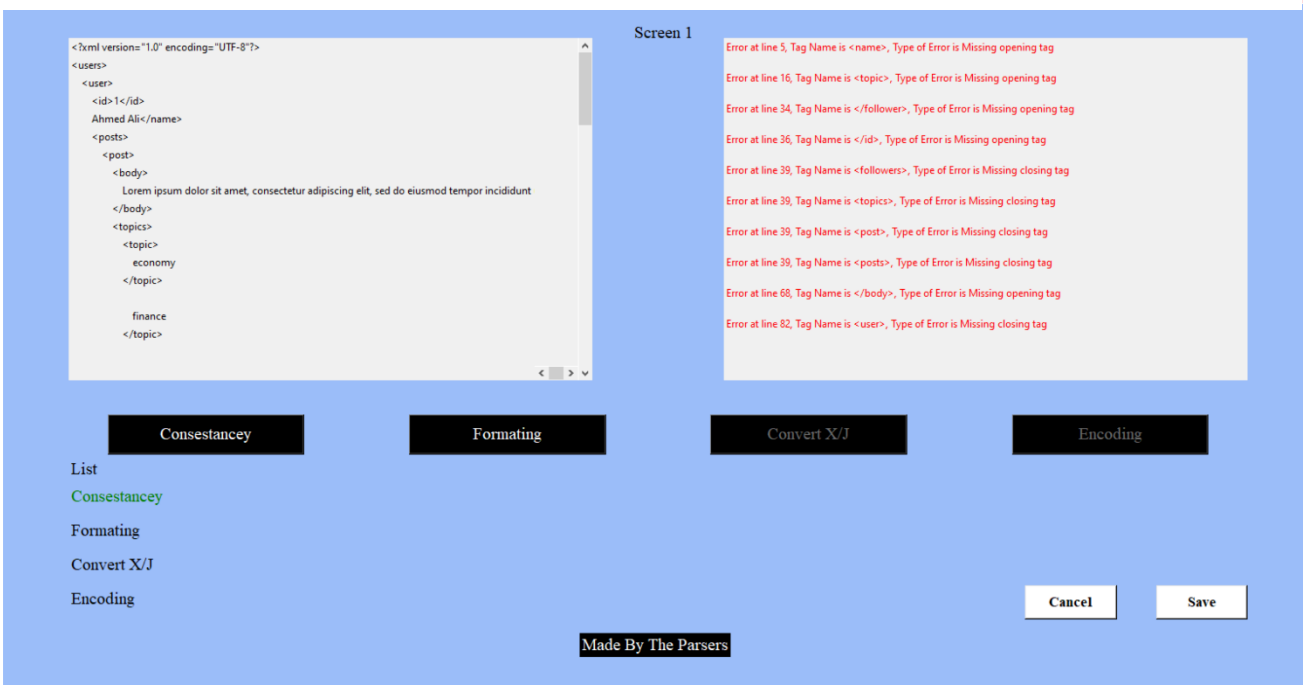

Figure 2: GUI Option 1

Buttons:

Choose xml/encoded File Load XML/encoded file.

Next Go to Screen 1.

Cancel close the program.

Consistency Check if the code is error free or not.

Format XML fix error if found in xml file and then format it.

Convert to JSON convert XML file into .json file format.

Encoding Encode xml-file into .enc file.

Decoding decode encoded-file into xml-file.

Option 2:(using pyqt5)

D. XML_Parser.py:

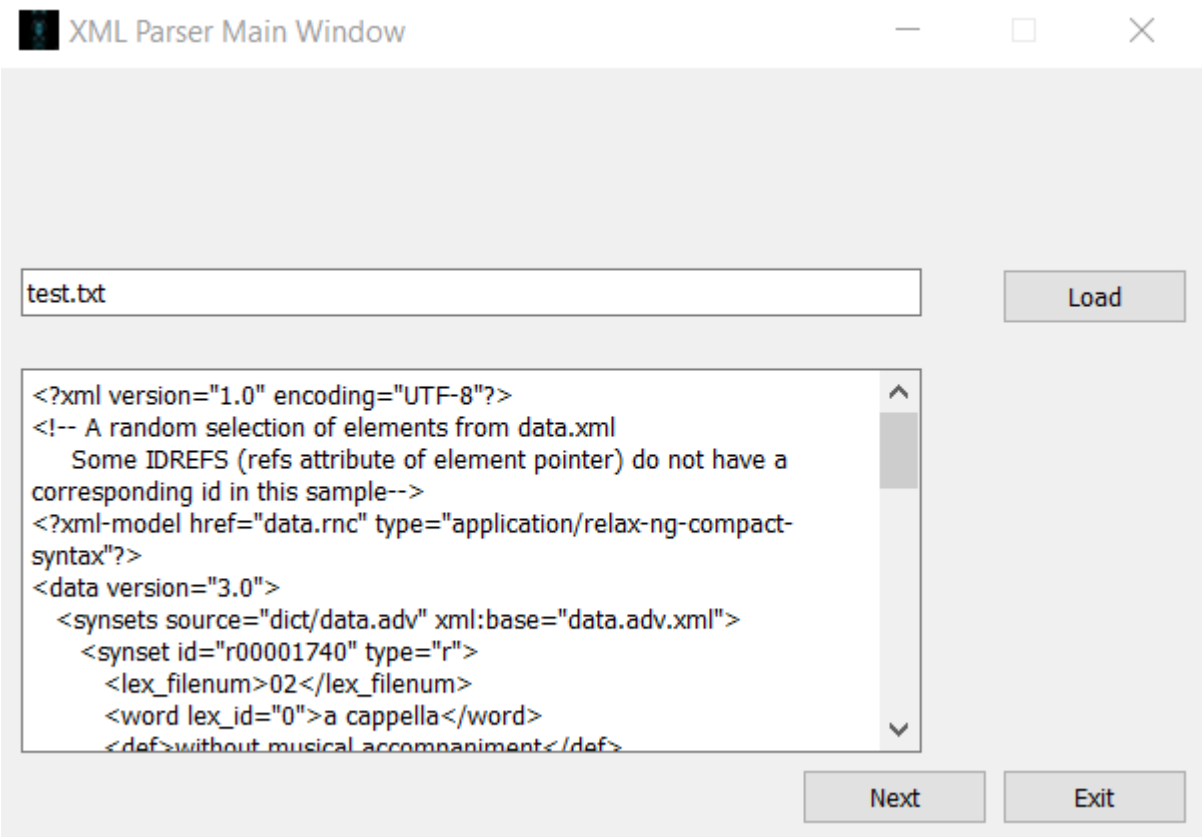

Figure 2:GUI Option 2

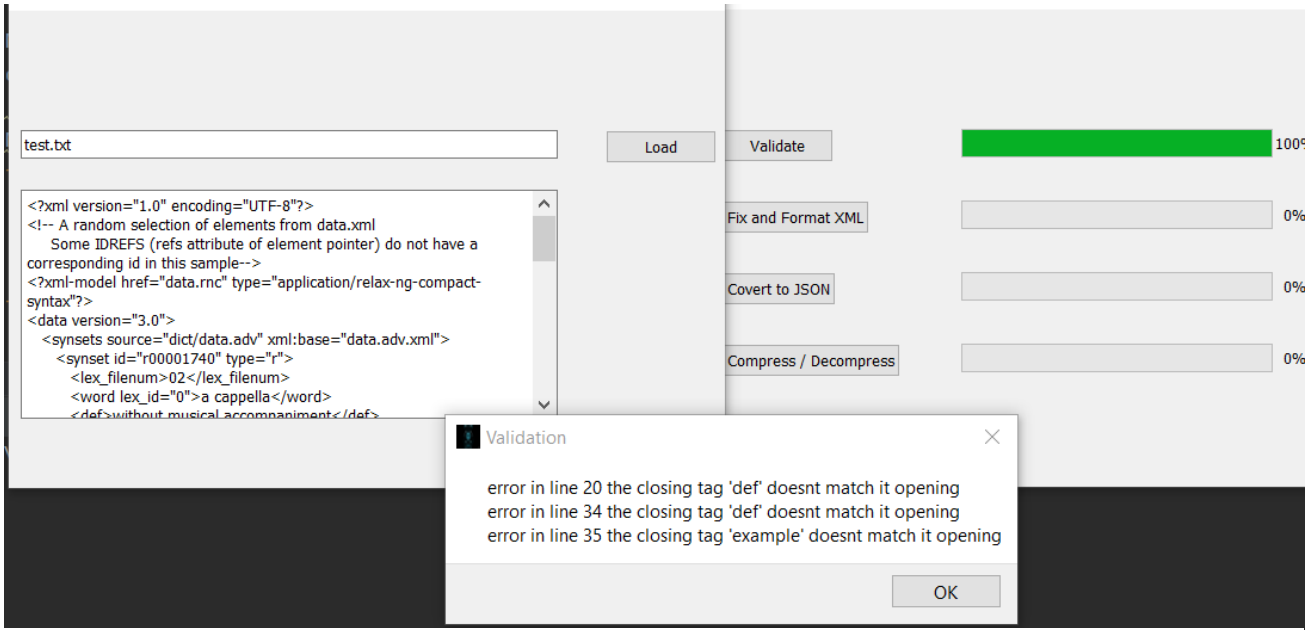

Figure 2:GUI Option 2

Buttons:

Load Load XML file

Next Switch to the functionalities window

Exit close the entire file

Validate Check if the code is error free or not.

Fix and Format XML fix error if found in xml file and then format it.

Convert to JSON convert XML file into .json file format

Compress/Decompress encode/decode file into the desired user format.

Phase 2

Function name: graph

Input: self

Output: adj matrix,marker array

Explanation: We Traverse The Tree of The XML to find for each user his ID and Followers then build ADJ_Matrix upon number of users and each user follower

Time and space complexity: $O(n^2)$ where n is number of users

For space complexity= O(x.y) where x is the size of adj matrix and y is length of bytes of each element

Function name: find_followers_and_id

Input: root//the root of the XML tree

Output: list of followers and ids refernces

Explanation: at XML tree we traverse on each user to make 2 lists one list for ids' references and the other for the followers' references and it is passed to the graph function to make the adj matrix

Time and space complexity: O[x*y] where x is number of users and y is number of children of each user

For space complexity : = O(x.y) where x is the size of array and y is length of bytes of each element

Function name: printer

Input: one element of the 2 reference lists ,L//empty list

Output: L// after the data(id) get collected in it

Explanation: traverse through XML tree and get leaf nodes data

Time and space complexity: we will use the a master theorem $aT(n/b)+O(n^d)$ it will be log(n) as we traverse inside a tree and n is the entire number of elements

For space complexity : = O(1) auxiliary space

- Note we used numpy to manipulate matrices and Matplotlib to plot the graphs and we used networkx to display the graph
- **GUI Added Buttons**
    1) Show graph: to show the graph
    2) Back: to return to the previous screen
    3) Close: to close the program

- **Screesshots**

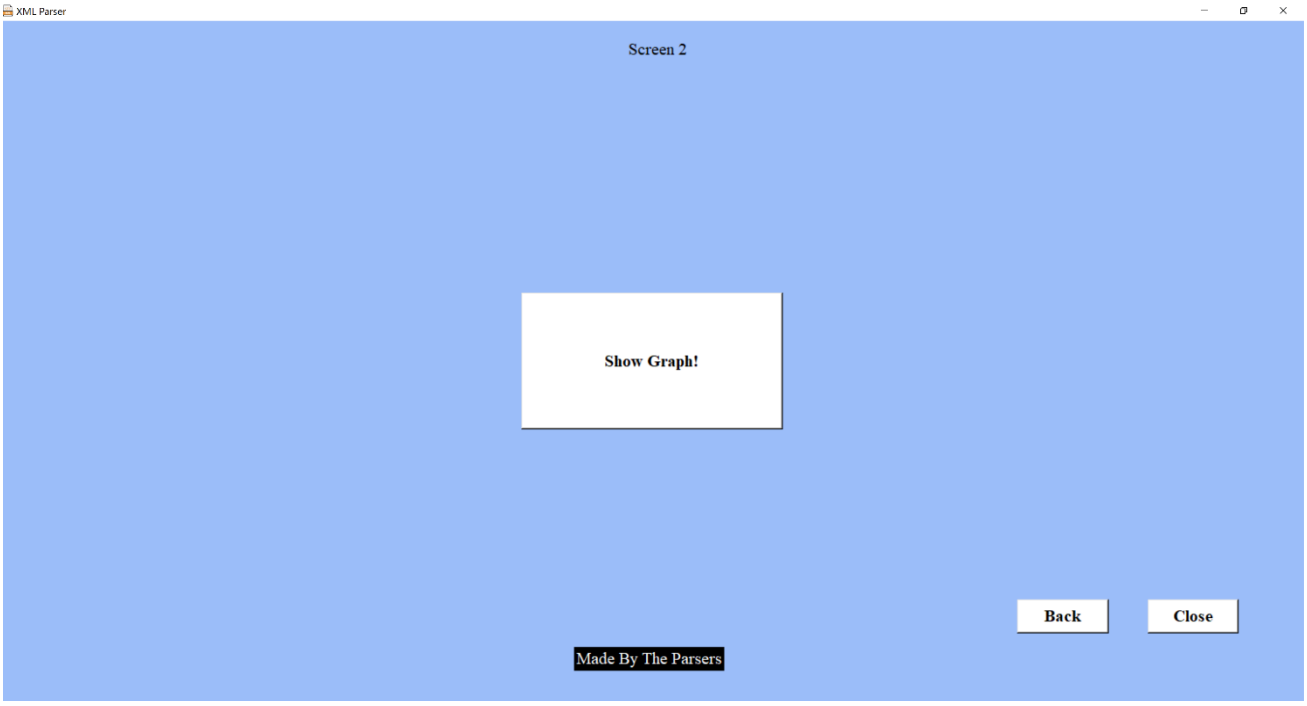

Figure 3(GUI screen to deaw the graph)

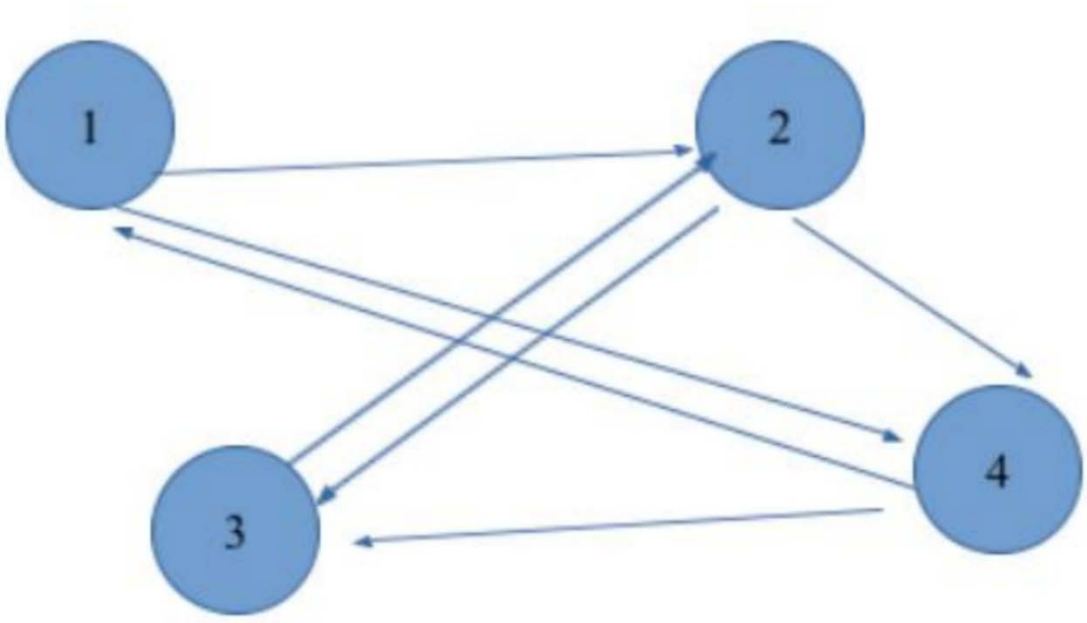

Figure 5(required shape of the graph)

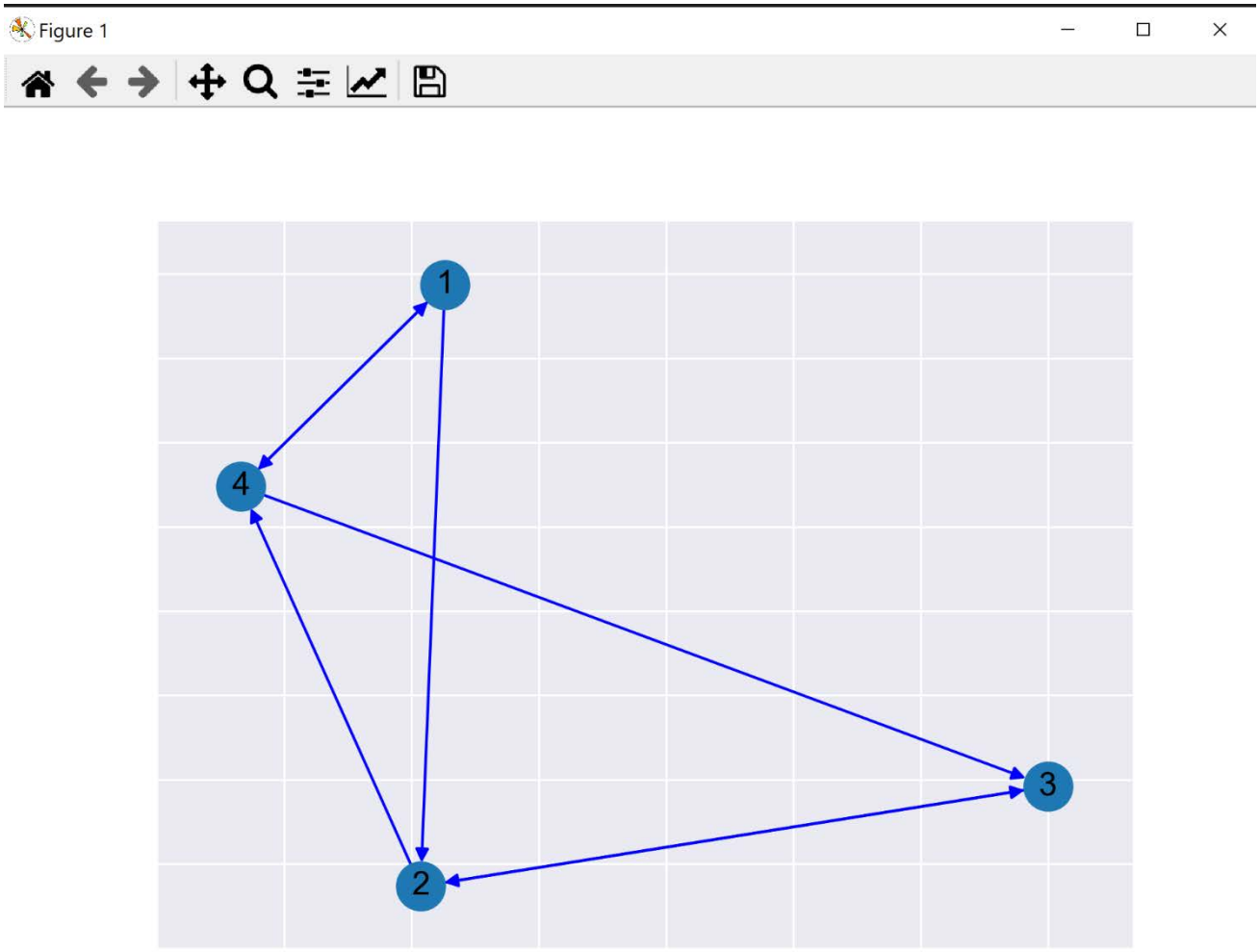

Figure 3(the shape of the graph we produced))