Data structures

Xml Project

Team members	ID
Omar Muhammed Mahmoud Al-Tahan	2001277
Mohamed Ali Fadl Kassem	2000965
Marwan Wael Mahmoud Abbas	2001244
Ahmed Eissa Mahfouz Eissa	2001725
Mohamed Ayman Mohamed soliman	2001048
Mohamed Shawky Mohamed abdelzim	2001687
Ahmed Khamis Said Khames	2001489

Video Link:

GitHub Repository:

Background:

We have made a GUI Application to do some basic operations on xml files such as:

- Ensuring Consistency.
- Modifying text and Uploading files.
- Correcting XML.
- Formatting XML.
- Minifying XML.
- Transforming XML into JSON.
- Compressing and decompressing files.
- Generating a social network graph of users and their followers in an XML file.
- Graph Visualization
- Undo and Redo
- Searching Posts

Implementation Details:

Checking Consistency:

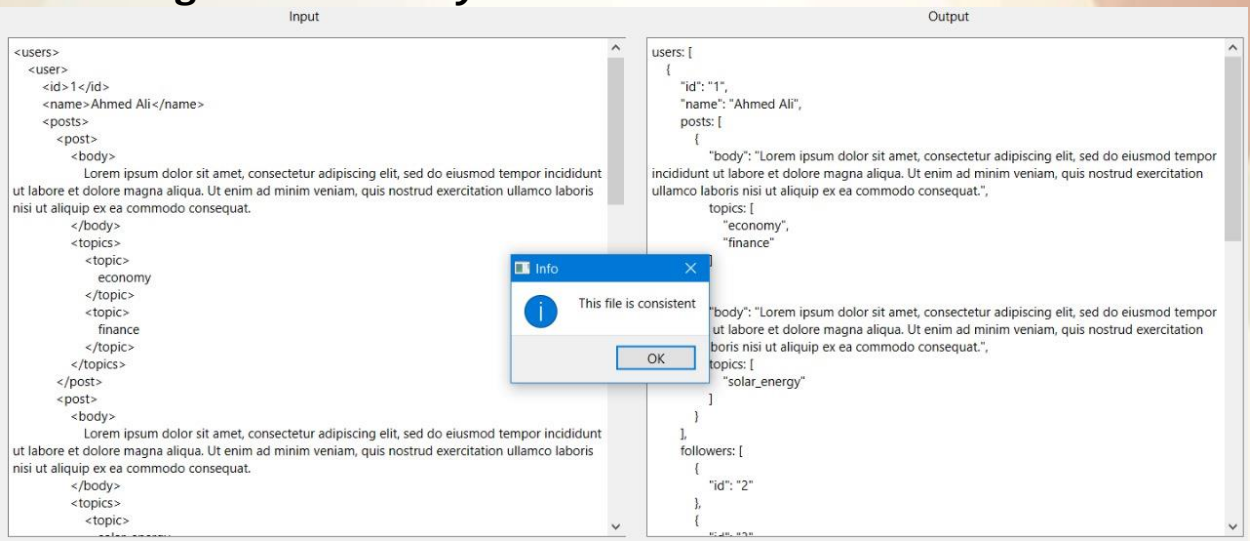


Fig. 1: Checking Consistency of a consistent file

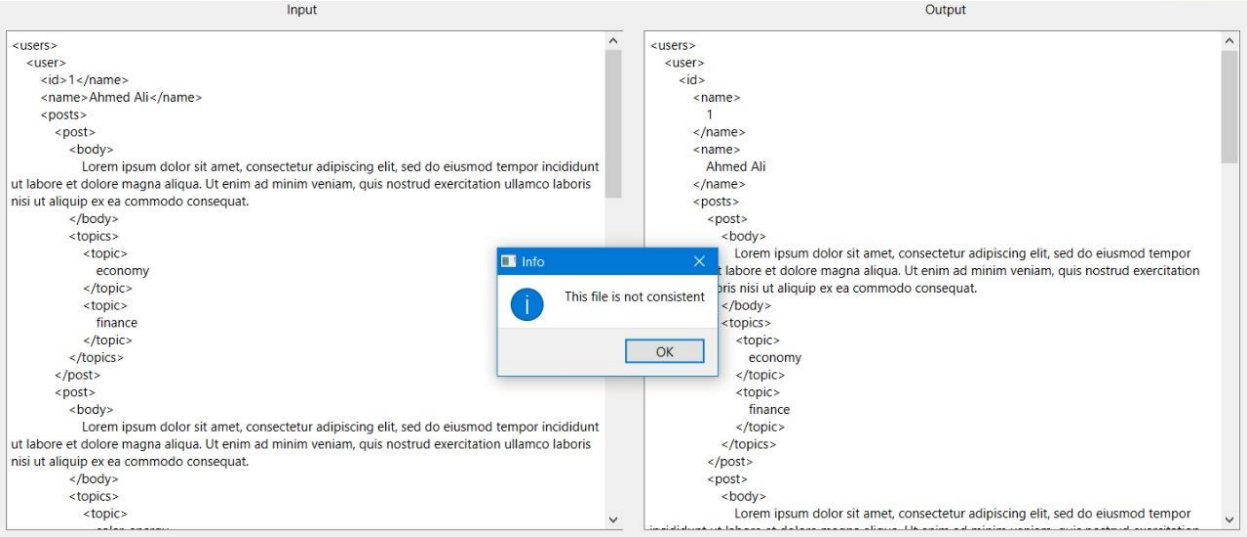


Fig. 2: Checking Consistency of a non-consistent file

C: > Users > user > Downloads > checkconsistency.cpp

```cpp
#include "checkconsistency.h"
QString checkconsistency(QString filedata) {
    filedata=diff_lines(filedata);

    QTextStream in(&filedata);
    QString line;
    stack<Node> node_stack;
    Node imaginary_firstnode;
    node_stack.push(imaginary_firstnode);
    int cnt_line_number=0;
    while (in.readLineInto(&line)) {
        line = line.trimmed();
        if (line.isEmpty()) {
            continue;
        }
        cnt_line_number++;
        if (line.contains("<")) {
            QString tag = line.mid(line.indexOf("<") + 1, line.indexOf(">") - line.indexOf("<") - 1);
            if (tag[0] == '/') {
                if (node_stack.size()<2) {
                    return "not consistent";
                } else {
                    Node child = node_stack.top();
                    if (child.name != tag.mid(1)) {
                        return "not consistent";
                    } else {
                        node_stack.pop();
                        Node& parent = node_stack.top();
                        parent.children.push_back(child);
                    }
                }
            } else {
                Node node;
                node.name = tag;
                node_stack.push(node);
            }
                    } else {
                        Node node;
                        node.name = tag;
                        node_stack.push(node);
                    }
                } else {
                    if(node_stack.size()<2){
                        return "not consistent";
                    } else {
                        Node& node = node_stack.top();
                        node.value += line;
                    }
                }
            }
            if (node_stack.size() > 1) {
                return "not consistent";
            }


            return " consistent";
    }
```

Consistency Check:

After processing all lines, it checks if there are remaining nodes in the node_stack. If there are more than one node remaining, it implies an inconsistency in the structure of the markup, so it returns "not consistent". Otherwise, if only one node remains (presumably the initial imaginary node), it returns "consistent".

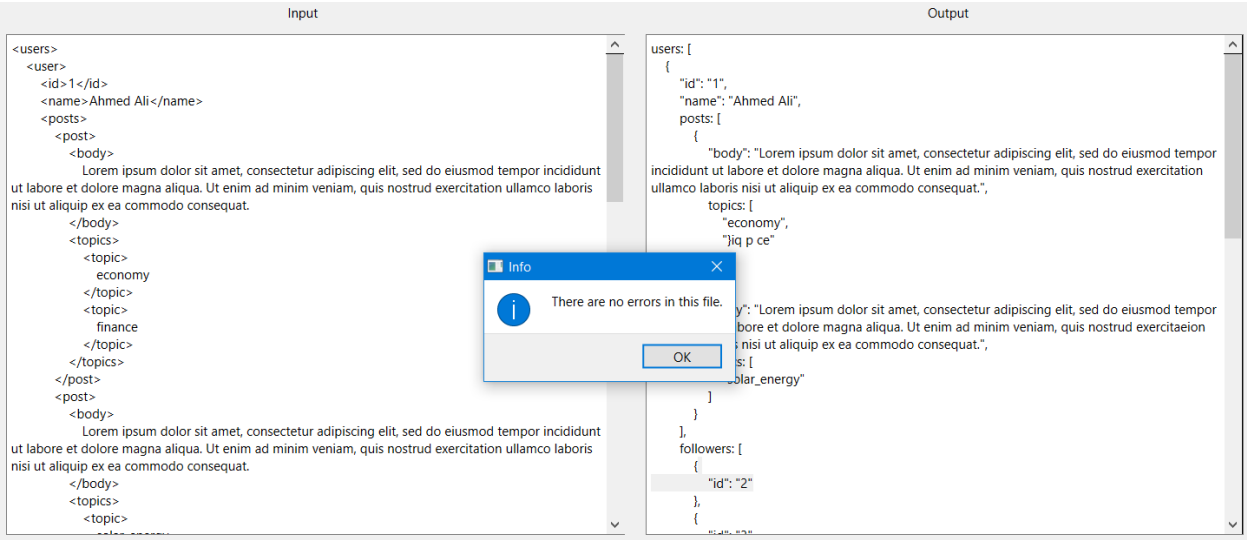
Input Processing:

The function takes a QString filedata and processes it by performing a function called diff_lines that appears to handle some line-based operations and formatting.
Tag Consistency Checking:

It initializes a QTextStream in to read from filedata.
It uses a stack of Node structures named node_stack to keep track of the structure of the markup.
It initializes an imaginary first node and pushes it onto the stack.
It iterates through each line in the input file represented by filedata.
For non-empty lines, it trims the line and proceeds with further operations.
Line Processing:

If a line contains a '<' character (indicating a tag in the markup):
If the tag starts with '/', it implies a closing tag. It checks whether the closing tag matches the top node's name in the node_stack. If not consistent (mismatch), it returns "not consistent". Otherwise, it pops the top node from the stack and appends it as a child to its parent.
If it's an opening tag, it creates a new Node and pushes it onto the node_stack.
If the line doesn't contain a '<' character (presumably representing content within tags), it appends the line's content to the value of the top node in the node_stack.

Correcting XML:



```cpp
#include "correct.h"
String correct(QString filedata) {

    filedata= diff_lines(filedata);
    stack<Node> node_stack;
    QString line;
    Node imaginary_firstnode;
    node_stack.push(imaginary_firstnode);
    int cnt_line_number=0;
    QVector<QPair<int,QString>> solved;
    QTextStream stream(&filedata);
    while (!stream.atEnd()) {
        line = stream.readLine().trimmed();
        if (line.isEmpty()) {
            continue;
        }
        cnt_line_number++;
        if (line.contains("<")) {
            QString tag = line.mid(line.indexOf("<") + 1, line.indexOf(">") - line.indexOf("<") - 1);
            QString s='<'+tag+'>';
            solved.push_back({cnt_line_number,s});

            if (tag[0] == '/') {
                if (node_stack.size()<2) {
                    //   cout << "Error: Closing tag without opening tag: " << tag <<" at line : "<<cnt_line_number<< endl;
                    if(solved.size()==1){
                        solved[solved.size()-1].first++;

                        solved.push_back({cnt_line_number,'<'+tag.mid(1)+'>'});
                        cnt_line_number++;
                        Node node;
                        node.name=tag.mid(1);
                        node_stack.push(node);

                    }else{

                        QString q=solved[solved.size()-2].second;
                        if (q[0]!='<'){
                            solved[solved.size()-2].first++;
                            solved[solved.size()-1].first++;

                            solved.push_back({cnt_line_number-1,'<'+tag.mid(1)+'>'});
                            cnt_line_number++;
                            Node node;
                            node.name=tag.mid(1);
                            node_stack.push(node);
                            //  continue;
                        }else{
                            solved[solved.size()-1].first++;
                            solved.push_back({cnt_line_number,'<'+tag.mid(1)+'>'});
                            cnt_line_number++;
                            Node node;
                            node.name=tag.mid(1);
                            node_stack.push(node);
                        }
                    }
                }
                Node child = node_stack.top();
                if (child.name != tag.mid(1)) {
                    //cout << "Error: Mismatched tags: " << child.name << " and " << tag <<" at line : "<<cnt_line_number<< endl;
                    QString q=solved[solved.size()-2].second;
                    if (q[0]!='<'){
                        solved[solved.size()-2].first++;
                        solved[solved.size()-1].first++;
                        solved.push_back({cnt_line_number-1,'<'+tag.mid(1)+'>'});
                        cnt_line_number++;
                        Node node;
                        node.name=tag.mid(1);
                        node_stack.push(node);
                        child=node_stack.top();
                    }else{
                        solved[solved.size()-1].first++;
                        solved.push_back({cnt_line_number,'<'+tag.mid(1)+'>'});
                        cnt_line_number++;
                        Node node;
                        node.name=tag.mid(1);
                        node_stack.push(node);
                        child=node_stack.top();
                    }
                }
                node_stack.pop();
                Node& parent = node_stack.top();
                parent.children.push_back(child);
            }else{
                Node node;
                node.name = tag;
                node_stack.push(node);
            }
        }else {
            solved.push_back({cnt_line_number,line});
            if(node_stack.size()<2){
                cout<<"no tags opened"<<" at line : "<<cnt_line_number<< endl;
            }
            Node& node = node_stack.top();
            node.value +=line;
        }
    }

```

Parsing and Correction:

It maintains a stack (node_stack) of Node objects to keep track of the tag structure.

While iterating through the processed lines, it analyzes lines containing '<' to identify tags.

It attempts to correct issues related to missing opening or closing tags by adjusting the tag structure and adding necessary tags based on certain conditions.

It keeps track of identified issues and their corrected versions in a vector (solved) to modify the content later.

After processing, it deals with any remaining unclosed tags by creating corresponding closing tags.

Finally, it sorts the identified issues and their corrected versions and reconstructs the modified content.

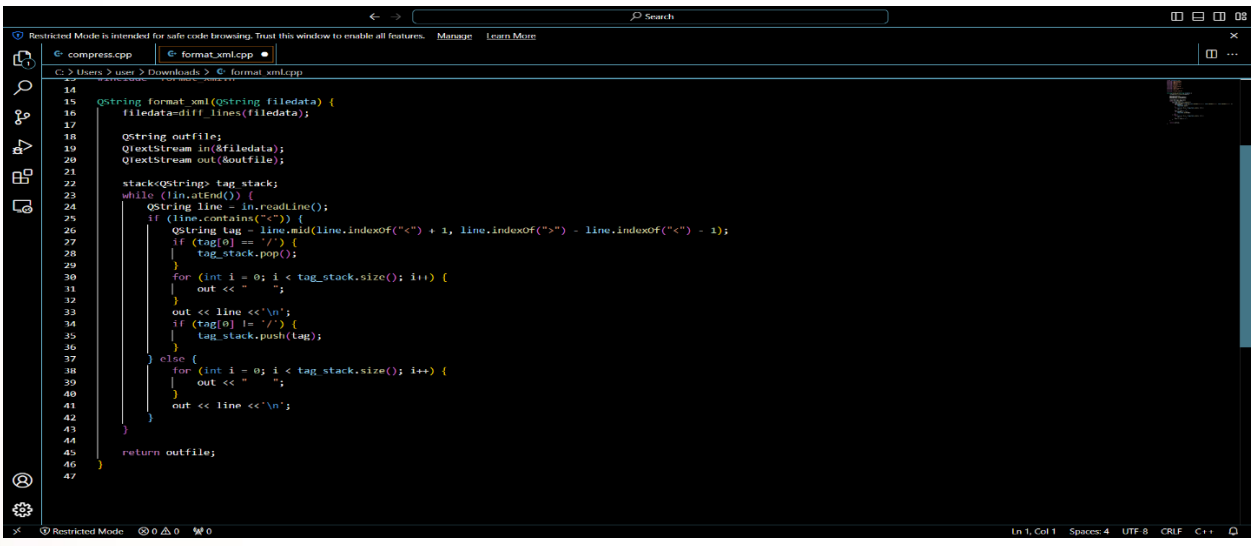
Formatting XML:

Input

```
<users>
  <user>
    <id>1</id>
    <name>Ahmed Ali</name>
    <posts>
      <post>
        <body>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
        </body>
        <topics>
          <topic>
            economy
          </topic>
          <topic>
            finance
          </topic>
        </topics>
      </post>
      <post>
        <body>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
        </body>
        <topics>
          <topic>
```

Output

```
<users>
  <user>
    <id>
      1
    </id>
    <name>
      Ahmed Ali
    </name>
    <posts>
      <post>
        <body>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
        </body>
        <topics>
          <topic>
            economy
          </topic>
          <topic>
            finance
          </topic>
        </topics>
      </post>
      <post>
        <body>
          Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation
```



formatting we worked on

two ways:

- The first way is coloring text if it's tag.
- The second way is by adding indentation to each line.

Minifying XML:

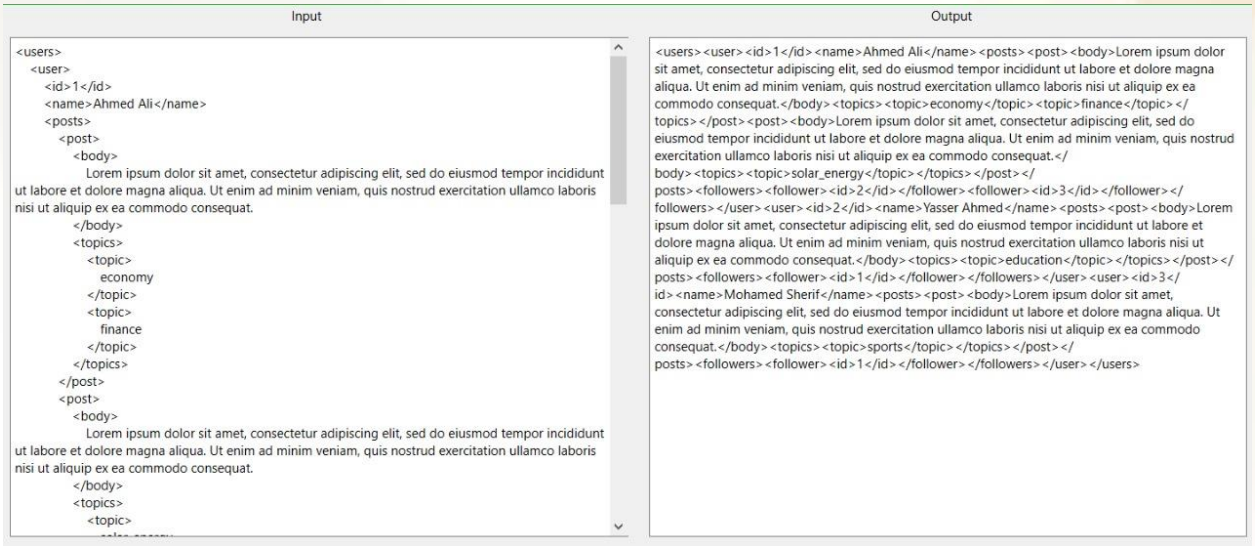


Fig. 5: Minify XML

```cpp
#include "minify_xml.h"
QString minify_xml(QString filedata) {
    QString outfile;
    QTextStream in(&filedata);
    QTextStream out(&outfile);

    while (!in.atEnd()) {
        QString line = in.readLine().trimmed();
        for (int i = 0; i < line.length(); i++) {
            if (line[i] != '\n' && line[i] != '\r' && line[i] != '\t') {
                out << line[i];
            }
        }
    }



    return outfile;

}
```

Minifying is done by removing all spaces then removing all "\n".

XML->JSON:

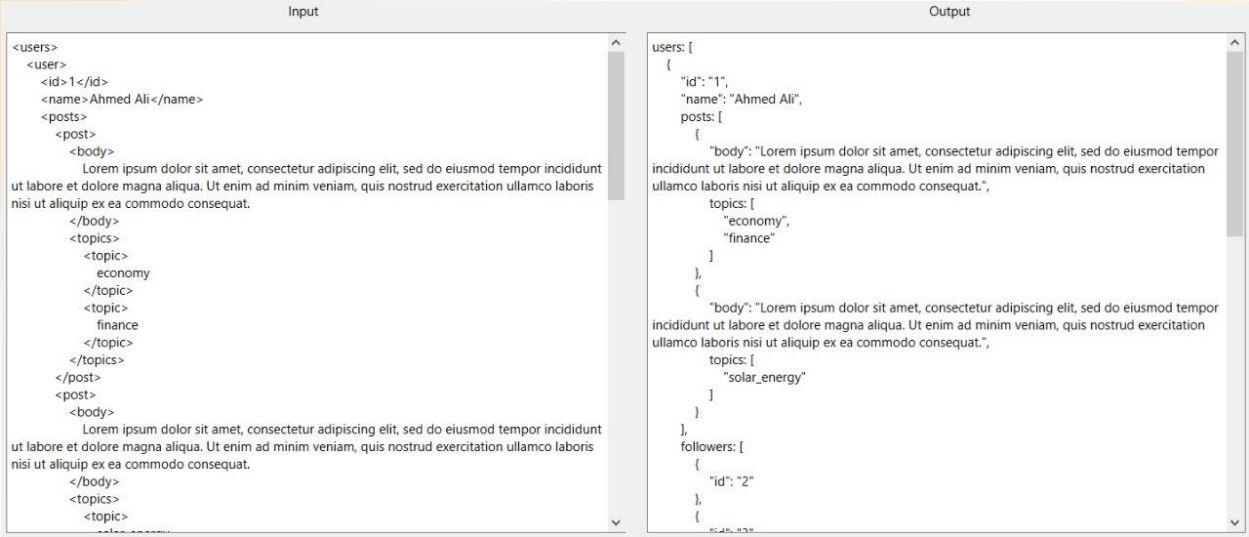


Fig. 6: Converting XML to JSON

```
QString xml_to_json(QString filedata) {
    filedata=diff_lines(filedata);

    stack<Node> node_stack;
    Node imaginary_firstnode;
    node_stack.push(imaginary_firstnode);
    int cnt_line_number = 0;
    QTextStream in(&filedata);
    while (!in.atEnd()) {
        QString line = in.readLine().trimmed();
        if (line.isEmpty()) {
            continue;  // Skip empty lines
        }
        cnt_line_number++;

        if (line.contains("<")) {
            QString tag = line.mid(line.indexOf("<") + 1, line.indexOf(">") - line.indexOf("<") - 1);

            if (tag[0] == '/') {

                Node child = node_stack.top();

                node_stack.pop();

                Node& parent = node_stack.top();
                parent.children.push_back(child);
            } else {

                Node node;
                node.name = tag;
                node_stack.push(node);
            }
        } else {
            if (node_stack.size() < 2) {
                qDebug() << "no tags opened" << " at line : " << cnt_line_number;
            }
            Node& node = node_stack.top();
```

convert XML to Json, we need first to build tree from the XML file then giving the root of the tree to *jsonStringBuilder()* to get the Json object then appending parenthesis to it.

Compressing & Decompressing Files:

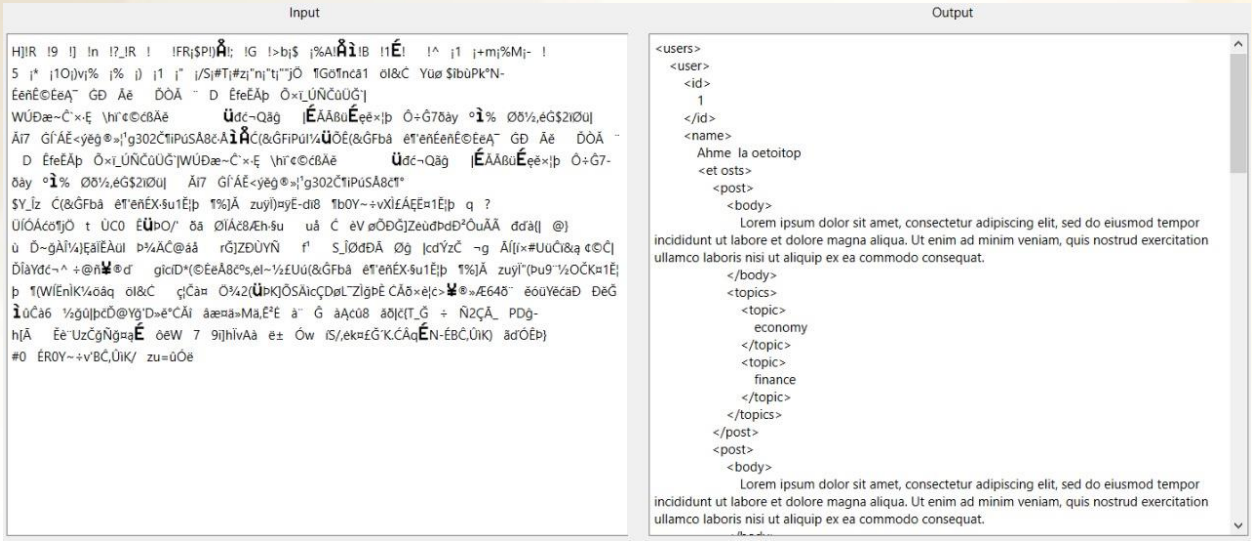


Fig. 7: decompressing a file.

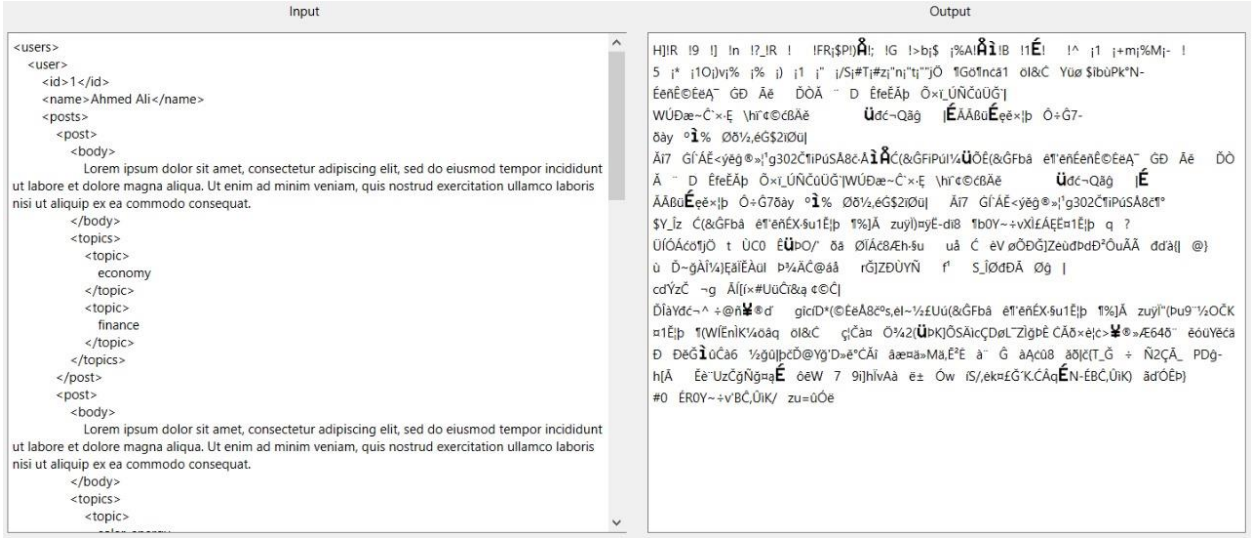


Fig.8: Compressing a file.

```
QString compress(QString filedata) {
    QString compressedfile="";
    filedata=minify_xml(filedata);
    QString input = filedata;

    QMap<QChar, int> frequencyTable;
    for (int i = 0; i < input.length(); i++) {
        frequencyTable[input[i]]++;
    }
    int x =frequencyTable.size() + 33;
    compressedfile=compressedfile+QChar(x);

    //QMap<QChar, bool> visited;
    bool ivisited[2000] = {false};
    for (int i = 0; i < input.length(); i++) {
        if (!ivisited[input[i].toLatin1()]) {
            ivisited[input[i].toLatin1()] = true;

            compressedfile=compressedfile+QChar(input[i].unicode() + 33);

            int x = frequencyTable[input[i]];
            int num1, num2;
            convertDecimalToBinary(x, num1, num2);
            if (num2 <= 33) {
                num2 = num2 + 33;
                num1 = num1 | (1 << 7);
            }
            compressedfile=compressedfile+ QChar(num1 + 33)+QChar(num2);

        }
    }
    huffman_Node* root = buildHuffmanTree(frequencyTable);
    QMap<QChar, QString> huffmanTable;
    QMap<QString, QChar> opphuffmanTable;
    generateHuffmanCodes(root, "", huffmanTable, opphuffmanTable,false);
    QString compressedInput = "";
    for (int i = 0; i < input.length(); i++) {
```

input Preparation:

Initializes compressedfile to store the compressed data.

Utilizes minify_xml to condense the XML data.

Constructs a frequency table (frequencyTable) to count the occurrences of each character in the input.

Encoding Process:

Iterates over the input data and creates a unique representation for each character in the compressed file.

Utilizes a unique number to represent the total number of distinct characters in the input file plus 33.

Creates a compact representation for each character:

Encodes characters and their frequencies into a specific format.

Constructs a Huffman tree based on character frequencies and generates Huffman codes.

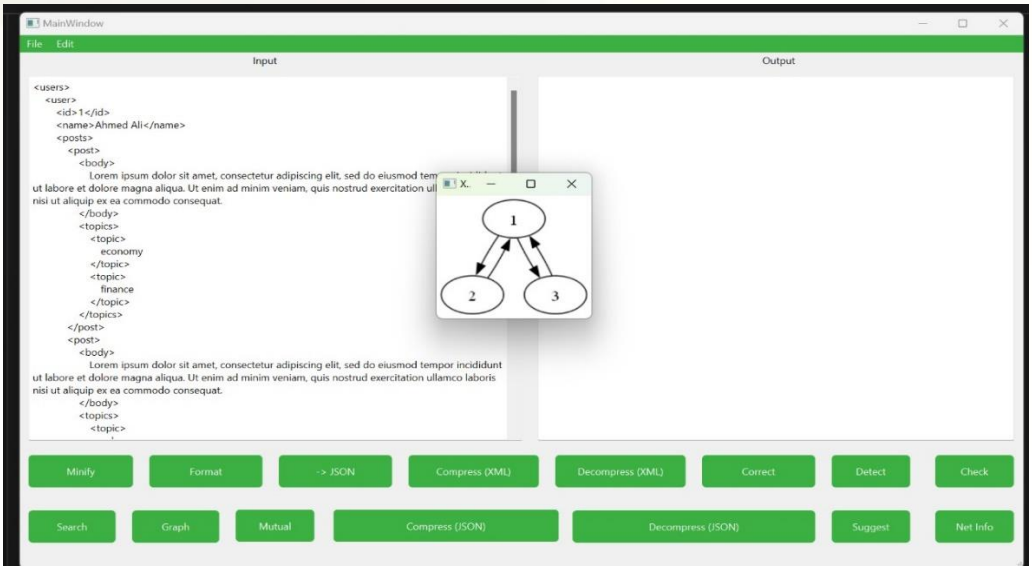
Encodes the original input using Huffman codes.

Encodes the encoded input into a binary format with 8-bit representation and appends it to compressedfile.

Network Analysis:

a) Graph Visualization

The graph is in adjacency list form so we loop through each node and graph a circle then we go through each edge of each node and draw an arrow



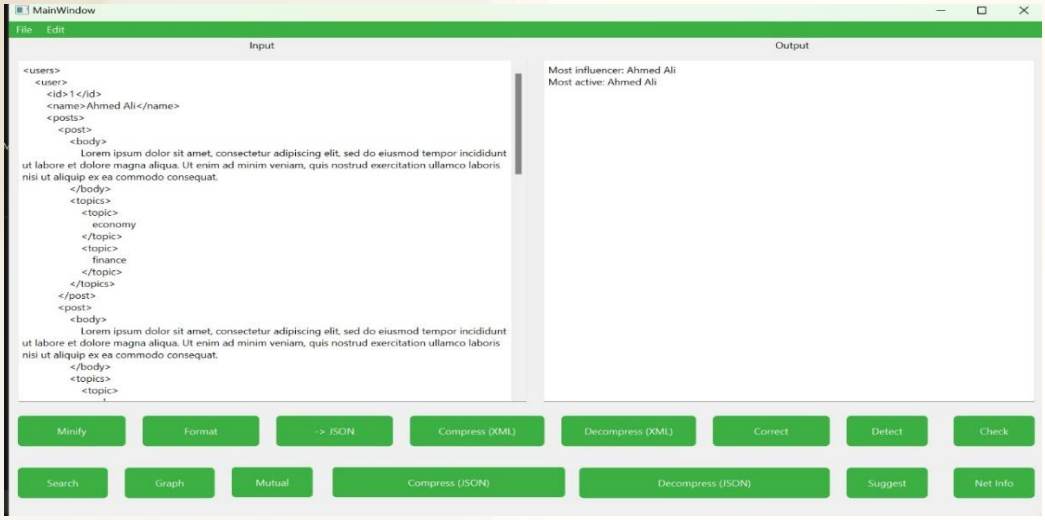
b) Network Statistics

- Most active user:

since we're using the adjacency list form of the graph we simply loop through the connection list of each user and see which is following more users

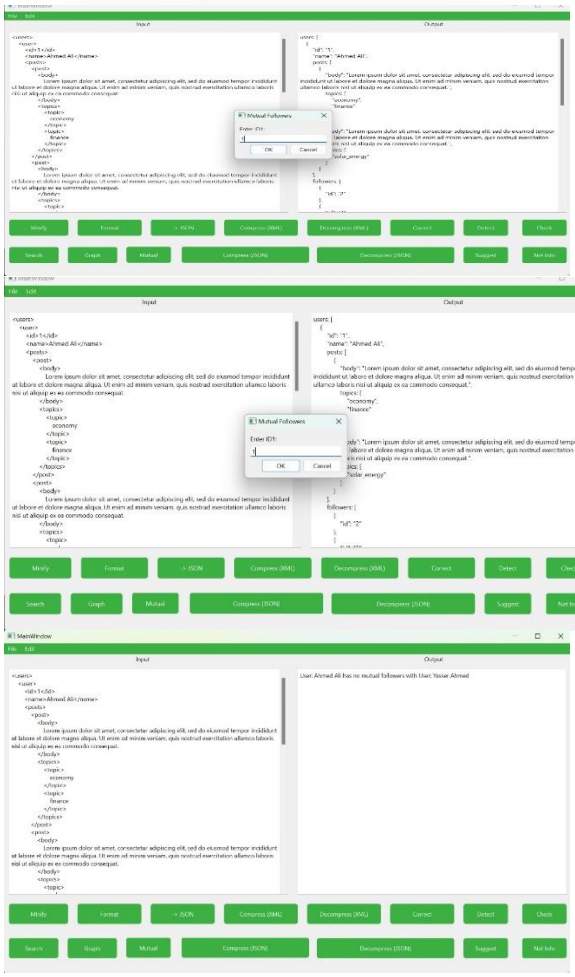
- Most Influencer:

we can't simply loop through all users and find the one that's following the most users because the user with the most followers might not be following anyone so we need to loop through all the users and count up the follower count of each and then decide which user has the largest follower count



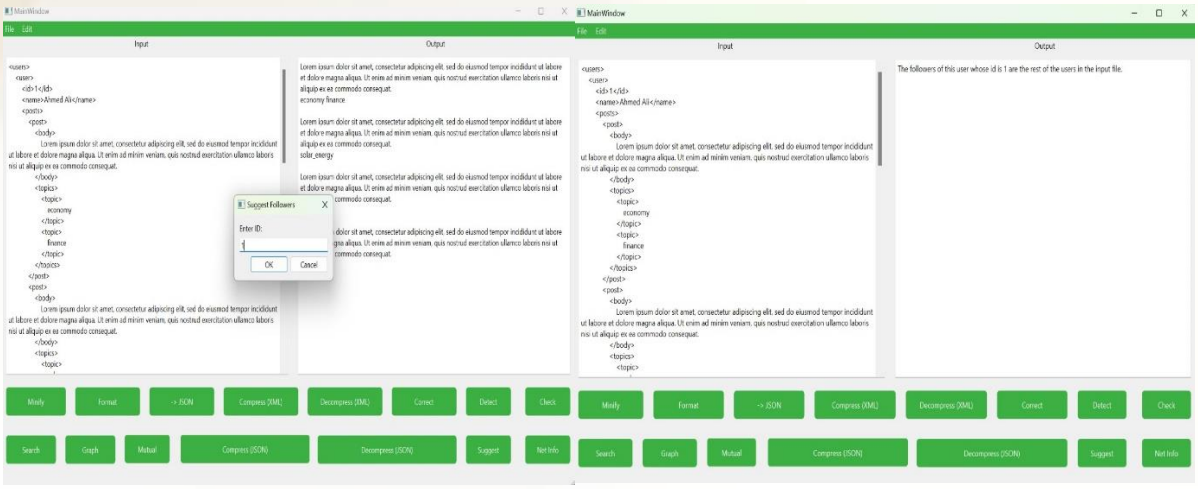
- Mutual Followers:

We loop through the connections of each node and check if both have a node in common (user)



- Suggested Followers:

For the user we loop through his adjacency list and for each entry we go through its adjacency list and add any node the isn't the user and hasn't been added



Undo & Redo:

Undo & Redo Operations are done through using two stacks for pushing old data and on undo it's popped and pushed into redo stack.

Searching:

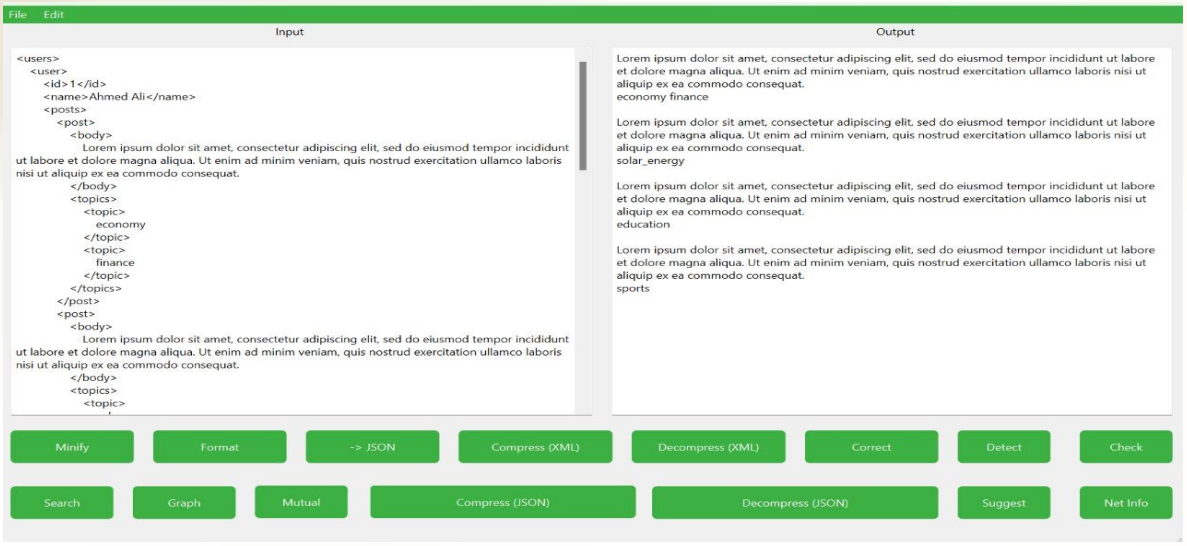


Figure 13: Searching for body of posts.

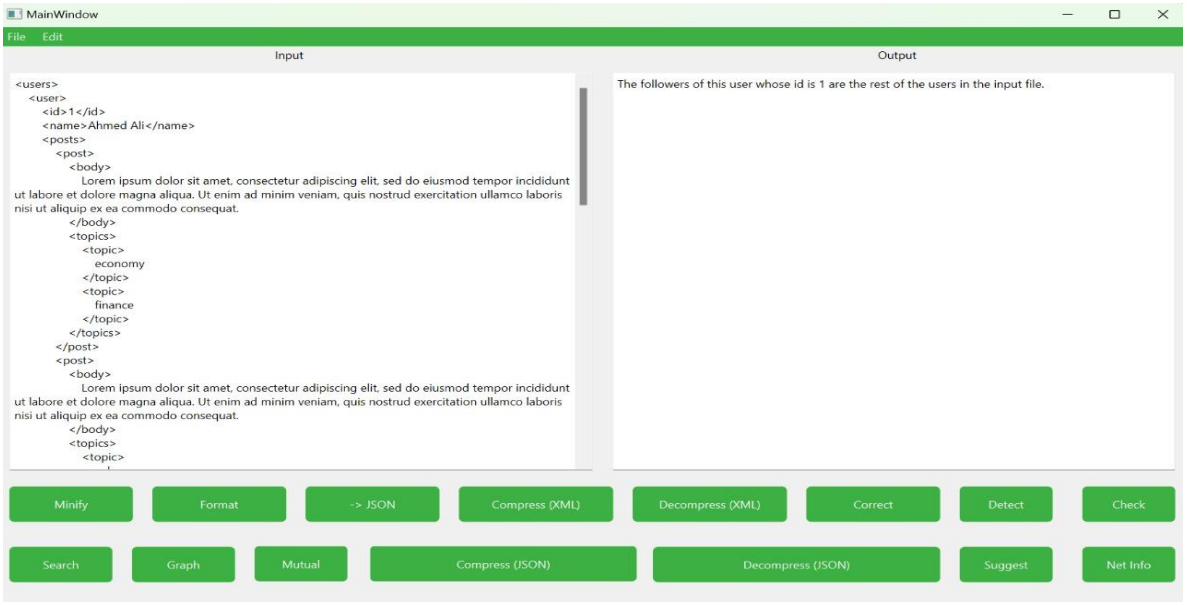


Figure 14: Searching for topics of posts.

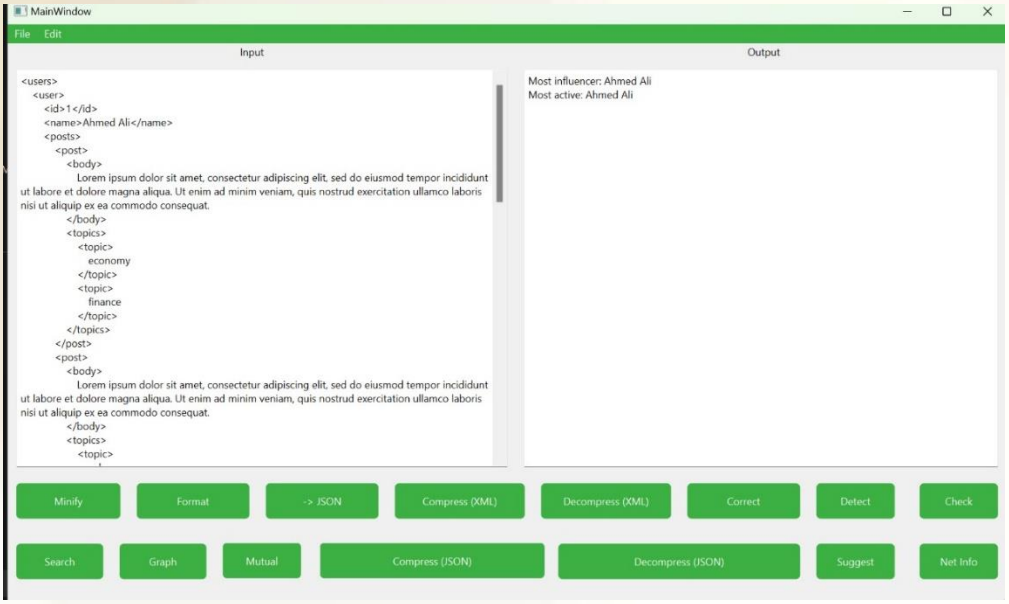


Figure 15: No posts found.

Searching for topics of posts of each user by removing spaces from edges then checking if string contains the topic and doing the same algorithm for searching for body.

Complexity:

Operation	Time Complexity
Checking consistency	O(n*m) : the number of lines in the file is 'n' and the average length of a line is 'm'
Correcting XML	O(n*m) : where n is the number of lines in the processed data, and m is the maximum size of the node_stack or the solved vector
Formatting XML	O(n * m) : where n is the number of lines in the input filedata, and m is the average length of each line
Minifying XML	O(n * m): where n is the number of lines in the input filedata, and m is the average length of each line.
XML->JSON:	O(n) : where n is the number of lines in the processed filedata.

Graph Visualization and Network Analysis	Visualization: O(V*E) Most active user: O(V) Most Influencer: O(V*E) Mutual Followers: O(V) Suggested Followers: O(V^2) Where V is the number of vertices/nodes an E is the number of edges
Compressing File	O(n log n) : where 'n' is the number of distinct characters
Decompression File	O(n log n): where 'n' is the size of the input data or the number of unique characters present
Undo & Redo	O(1)
Searching for post	O(N * M * K): where 'N' is the number of users, 'M' is the average number of posts per user, and 'K' is the average length of posts or topics.
Mutual influencer	O(vertexNum): where 'vertexNum' is the number of users in the graph.
Most active user	O(vertexNum^2): where 'vertexNum' represents the number of users in the graph.
Most influencer	O(vertexNum): where 'vertexNum' represents the number of users in the graph.
Followers suggestion	O(N^2 * M log M): where 'N' is the average number of followers and 'M' is the average number of followers of each follower.

References:

Referenced Item	Location of Reference
Huffman Coding for compression and decompression	https://en.m.wikipedia.org/wiki/Huffman_coding
C++ Documentation	https://stackoverflow.com/questions/4396573/tools-rules-or-processes-for-checking-internal-consistency-of-xml-config-files https://codebeautify.org/xmlvalidator https://www.w3schools.com/xml/xml_validator.asp https://jsonformatter.org/xml-to-json https://stackoverflow.com/questions/1082285/best-compression-algorithm-for-xml