A blue and black logo with birds and text

Description automatically generated

**CSE331: Data Structure and Algorithms**

**Project report**

|  |  |
| --- | --- |
| Name | Id |
| Moustafa Ahmed Hashem | **2100467** |
| Michael George Naem | **2100709** |
| Moaz Ragab Abuelmagd | **2100938** |
| Ahmed Ashraf Ali | **2100255** |
| Karen Maurice William | **2100748** |
| Mohamed Waleed Elsayed | **2100623** |
| Mayar Walid Said | **2100953** |
| Kareem Ehab | **2100913** |
| Mohamed Hani Hamdi | **2100915** |
| Amr Essam Mahmoud | **2001089** |

**XML Editor and Social Network Visualizer**

**Table of Contents:**

Table of Contents

[**1. Background** 3](#_Toc185456067)

[**2. Implementation Details** 3](#_Toc185456068)

[**3. Complexity of Operations** 6](#_Toc185456069)

[**4. Conclusion** 7](#_Toc185456070)

[**5. References** 7](#_Toc185456071)

**1. Background**

**1.1 Introduction to XML**

Extensible Markup Language (XML) is a versatile markup language designed to store and transport data in a structured format. It is widely used in various applications for data interchange due to its ability to represent complex data structures in a human-readable format. However, working with XML files often requires specialized tools for parsing, analyzing, and visualizing data.

**1.2 Project Overview**

This project focuses on developing a desktop application that facilitates the parsing and visualization of XML files and many other operations like formatting, compression/decompression and conversion to JSON. Also, it represents user data in a social network. Each user within the XML file has attributes such as a unique ID, name, a list of posts, and followers. The application aims to provide various functionalities, enabling users to manipulate and analyze this data effectively.

**1.3 Key Features**

The application offers a comprehensive set of features that include:

* **XML Parsing and Verification:** Validates the structure of XML documents to ensure data integrity.
* **XML Formatting:** Prettifies XML files to improve readability.
* **XML to JSON Conversion:** Converts XML data into JSON format for compatibility with JavaScript applications.
* **File Minification:** Reduces file size by removing unnecessary whitespace and indentations.
* **Data Compression:** Compresses XML/JSON files using Byte Bair Encoding Technique.
* **Graphical Representation of User Relationships:** Visualizes user connections using graph data structures.
* **Network Analysis Tools:** Identifies influential users, mutual followers, and suggests connections.
* **Post Searching Capabilities:** Allows users to search posts by keywords or topics.

This application caters to both casual users through a GUI and advanced users via a command-line interface (CLI), enhancing accessibility and usability.

**2. Implementation Details**

**2.1 General Architecture**

The application consists of the following components:

1. **XML Processor:** Handles parsing, verifying, formatting, and minifying XML files.
2. **Data Converter:** Converts XML to JSON and vice versa.
3. **File Compressor:** Compresses and decompresses files using a custom or standard compression technique.
4. **Graph Module:** Builds and visualizes user relationships using a graph data structure.
5. **Network Analyzer:** Extracts insights such as the most influential user, the most active user, mutual followers, and follow suggestions.
6. **Search Module:** Allows searching posts by keywords or topics.

**2.2 Graphical User Interface (GUI)**

The GUI is designed for user-friendliness and supports:

* Browsing input files.
* Displaying outputs in a read-only text field.
* Saving output files to user-defined locations.
* Buttons for each supported operation.

**2.3 Command-Line Interface (CLI)**

The CLI allows advanced users to execute operations through commands like:

xml\_editor verify -i input\_file.xml -o output\_file.xml

xml\_editor json -i input\_file.xml -o output\_file.json

**2.4 Functionalities**

**2.4.1 XML Consistency Check**

The application validates XML structure by:

* Ensuring all opening tags have corresponding closing tags.
* Highlighting errors and their line numbers in case of inconsistencies.
* Offering an option to auto-correct errors and save the fixed file.

**2.4.2 Formatting (Prettifying)**

Improves XML readability by:

* Indenting nested elements.
* Removing unnecessary whitespace while maintaining structure.

**2.4.3 XML to JSON Conversion**

Converts XML data to JSON format for easier integration with JavaScript-based tools.

**2.4.4 Minifying XML**

Reduces file size by:

* Removing unnecessary whitespace and newlines.

**2.4.5 Compression and Decompression**

[Byte Bait Encoding](https://en.wikipedia.org/wiki/Byte_pair_encoding) is the algorithm used to compress any kind of text file. After compression, the program saves the key pairs of the encoding process followed by the encoded data.

Concerning the decompression operation, the program reads the file ready to fetch the pairs to be able to apply Byte Pair Decoding on the text that follows.

**2.4.6 Graph Representation**

Represents user relationships as a directed graph:

* Nodes represent users.
* Edges represent follower relationships.

**2.4.7 Network Analysis**

Analyzes the social network to:

* Identify the most influential user (maximum followers).
* Identify the most active user (maximum connections).
* Find mutual followers for a set of users.
* Suggest users to follow (followers of followers).

**2.4.8 Post Search**

Enables searching for posts containing specific keywords or topics.

**3. Complexity of Operations**

**3.1 XML Parsing and Validation**

* **Time Complexity:**
* **Space Complexity:**

**3.2 Formatting (Prettifying)**

* **Time Complexity:**
* **Space Complexity:**

**3.3 XML to JSON Conversion**

* **Time Complexity:**
* **Joining the input array into a single XML string:** Time Complexity: O(n) where n is the total number of characters in the input XML array.
* **Converting XML to JSON using XML.toJSONObject:** Time Complexity: O(m) where m is the size of the XML string.
* **Correcting the JSON structure: Time Complexity:** O(k) where k is the number of users in the "users" array (or the number of objects processed in the JSON).
* **Converting JSON to a string with 5 spaces for indentation:** Time Complexity: O(n\_json) where n\_json is the size of the resulting formatted JSON string.
* **Splitting the formatted JSON string by newlines:** Time Complexity: O(n\_json) where n\_json is the length of the formatted JSON string. Overall Time Complexity: O(n + m + k + n\_json) If n, m, k, and n\_json are all proportional to the size of the input,
* **The overall complexity can be approximated as: O(n)**
* **Space Complexity:**
* **Space for the XML string: Space Complexity:** O(n) where n is the total number of characters in the input XML array.
* **Space for the JSON object:** Space Complexity: O(m) where m is the size of the JSON object, corresponding to the size of the XML input.
* **Space for correcting the JSON structure:** Space Complexity: O(k) This space is required to store the modified JSON object with correctly structured "users", "posts", and "followers".
* *Space for the formatted JSON string:* Space Complexity: O(n\_json) where n\_json is the size of the formatted JSON string.
* **Space for the array of strings created by splitting the JSON string by newlines:** Space Complexity: O(n\_json) where n\_json is the length of the formatted JSON string.

**Overall Space Complexity:** O(n + m + n\_json) If n, m, and n\_json are proportional to the input size, the overall space complexity can be approximated as: O(n)

**3.4 Graph Representation**

* **Time Complexity:**
  + addVertex Method: Time Complexity: O(1) since HashMap's putIfAbsent operation is constant time.
  + buildGraphFromXML Method: Time Complexity: O(n) where n is the length of the XML string, as it needs to process each character for pattern matching.
  + getAdjacencyList Method: Time Complexity: O(1) for HashMap lookup using getOrDefault.
  + areConnected Method: Time Complexity: O(d) where d is the number of followers (edges) for a given user, as it needs to search through the adjacency list.
  + isFollowing Method: Time Complexity: O(d) where d is the number of followers, as it needs to search through the adjacency list.
  + Overall Time Complexity: O(n) for building the graph, and O(d) for most operations where d is the number of edges per vertex.
* **Space Complexity:**
  + Adjacency List Storage: Space Complexity: O(V + E) where V is the number of vertices (users) and E is the number of edges (follower relationships).
  + User Object Storage: Space Complexity: O(V) where V is the number of users, each storing their basic information.
  + buildGraphFromXML Method: Space Complexity: O(n) where n is the size of the XML string for pattern matching operations.
  + Pattern and Matcher Objects: Space Complexity: O(1) as they use constant space regardless of input size.
  + User Posts and Followers Lists: Space Complexity: O(P + F) where P is the total number of posts and F is the total number of follower relationships.
  + Overall Space Complexity: O(V + E + P + F) where: V = number of vertices (users) E = number of edges (follower relationships) P = total number of posts F = total number of followers

**3.5 Network Analysis**

* **Most Influential/Active User:**
  + **Time Complexity:**
* Most Influential:
* ReadFile(s): This function reads the data and presumably populates the users array. Its time complexity depends on the size of the file and the number of users it processes. Let's denote this complexity as O(F), where F is the size of the file.
* Both for loops: both loops iterates through the users array, its time complexity is O(n) for both where n is the number of users.
* Overall, the two loops combined run O(n)+O(n)=O(n).
* Overall complexity of function is O(F) + O(n) = O(F + n).
* Active User:
* ReadFile(s): This function reads the data and presumably populates the users array. Its time complexity depends on the size of the file and the number of users it processes. Let's denote this complexity as O(F), where F is the size of the file.
* Outer for loop: it iterates through all the vertices in graph (representing the users) so its time complexity is O(n) where n is the number of users.
* Inner for loop: the inner loop iterates through all vertices again to check connections. This gives an O(n) complexity for the inner loop.
* Overall complexity for this part: O(n^2)).
* Another loop: This loop iterates through all users once, running in O(n).
* Combining the above, the overall complexity is O(F + n^2)
  + **Space Complexity:**
* Most influential:
* Space for users: Assuming users is an array of User objects, the space used by users is proportional to O(n) , where n is the number of users.
* Space for mostInf: mostInf stores references to User objects. In the worst case, all users might have the same number of followers, so the space complexity for mostInf is O(n).
* Variables like index, mostfollowed, and the loop counters require O(1) space.
* Overall, the space complexity is O(n).
* Active User:
* Space for users and mostActive: The users array and mostActive list require O(n) space.
* Graph: Depends on the graph representation: if **Adjacency Matrix**: O(n^2). If **Adjacency List**: O(n+e) where e is the number of edges (connections).
* Variables like index, mostfollowed, and the loop counters require O(1) space.
* Overall, the space complexity is O(n^2) or O(n + e).
* **Mutual Followers:**
  + **Time Complexity:**
    - ReadFile(s): This function reads the data and presumably populates the users array. Its time complexity depends on the size of the file and the number of users it processes. Let's denote this complexity as O(F), where F is the size of the file.
    - Outer loop: This loop iterates through all users, so it runs O(n), where n is the number of users.
    - Inner loops: For each user, the inner loop iterates over the given ids. Let the number of IDs in the array ids be m, so its complexity is O(m).
    - add(user): O(1)
    - Overall, the complexity is O(F)+O(n⋅m).
  + **Space Complexity:**
    - Space for user and mutalIF: The users array and mutualF list require O(n) space.
    - Graph: Depends on the graph representation: if **Adjacency Matrix**: O(n^2). If **Adjacency List**: O(n+e) where e is the number of edges (connections).
    - Variables like follows and id require O(1) space.
    - Overall, the space complexity is O(n^2) or O(n + e).
* **Follow Suggestions:**

**Time Complexity:**

Let:

* n: The number of users in the graph.
* m: The number of followers (connections) across all users.

1. **Fetching the User Object**:
   * socialNetworkGraph.getUserById(userId) takes O(1) (assuming hash-based retrieval).
2. **Building Sets of Followers**:
   * user.followers (creating userFollowers and alreadyFriends) depends on the number of followers of the given user. Let this number be f.
   * This operation is O(f).
3. **Iterating Through Followers**:
   * For each follower of the user (f followers), you loop through their followers. If the average number of followers per user is k, this results in O(f \* k).
4. **Checking and Adding Friends**:
   * The operations to check membership and add elements to the friendsOfFriends set are O(1) on average (hash set operations).
   * The loop iterates O(f \* k) times.
5. **Converting IDs to User Objects**:
   * After finding f' suggested friends (friends of friends), converting these IDs to User objects involves O(f'). In the worst case, f' = f \* k.

**Overall Time Complexity**:

* O(f) + O(f \* k) + O(f \* k) = O(f \* k).
* f is the number of followers for the user, and k is the average number of followers per user.
* In the worst case, f and k can approach n, making the time complexity O(n^2).

**Space Complexity:**

1. **Space for Sets**:
   * userFollowers and alreadyFriends store O(f) elements.
   * friendsOfFriends can store up to O(f \* k) elements in the worst case.
2. **Suggested Users**:
   * suggestedUsers contains up to O(f \* k) elements.

**Overall Space Complexity**:

* O(f \* k) (dominated by friendsOfFriends and suggestedUsers).
* In the worst case, f \* k = n^2, making the space complexity O(n^2).

**Final Summary:**

* **Time Complexity**: O(f \* k), worst-case O(n^2).
* **Space Complexity**: O(f \* k), worst-case O(n^2).

**3.6 Compression and Decompression**

* **Minifying:**
  + **Time Complexity:** **O (N)**
  + **Space Complexity:** **O (N)**
* **Compression:**
  + **Time Complexity:** **O (N + t ×N + Lkeys + Lvalues)**
  + **Space Complexity:** **O (N + Lkeys + Lvalues)**

t represents the number of iterations of the outer while loop.

* **Decompression:**
  + **Time Complexity:** **O (M + k x N + N)**
  + **Space Complexity:** **O(k + N)**
    - M: Length of the hashmap string s[0].
    - k: Number of pairs in the hashmap.
    - N: Length of the encoded data s[1].

**4. Conclusion**

This application provides a robust solution for parsing, analyzing, and visualizing XML data representing social networks. The combination of GUI and CLI modes ensures accessibility for both casual and advanced users. By integrating network analysis and search functionalities, it goes beyond basic XML operations, offering insights into user relationships and activity.

**5. References**

* [**Byte pair encoding - Wikipedia**](https://en.wikipedia.org/wiki/Byte_pair_encoding)
* [**Byte-Pair Encoding tokenization - Hugging Face NLP Course**](https://huggingface.co/learn/nlp-course/en/chapter6/5)