Project

Data Structure

Ahmed Essam El-Din Mohammed	1900731
Muhamed Hassan Mohamed	1900819
Omar Mohamed Moustafa Mohamed	1900953

Link on Repo: Link

Demo Video: Link

Contribution Table:

Ahmed Essam El-Din Mohammed	Validation Fixing Analysis functions for phase 2
Muhamed Hassan Mohamed	JSON Prettify GUI
Omar Mohamed Moustafa Mohamed	Compression & Decompression Minify Extract user data and phase 2 helper functions

Functions:

Phase One:

1) Validation:

Algorithm

- Read file line by line.
- Extract tag names from the line if exist by getting index of open tag and close tag and get the name between them.
- If there isn't a tag name between then push it as error of missing tag name.
- If there:
  - In case of open tag:
    push it to the stack and save its name in pushedTags(string).
  - In case of close tag:
    If close tag is matching the last open tag then pop from the stack.
    Else: Check if the close tag has an opening tag in pushedTags.
    - If it is found, then pop from the stack till the matched open tag for this closed tag and all pushed tags from the stack are pushed to the errors for having missed close tags.
    - Else if not found, then this close tag doesn't have an open tag and is added to errors.
  - After finish reading the file, check if stack isn't empty pop all tags from it and push them to errors as having a missed closed tag.

Time Complexity: O(N^2) **Space Complexity:** O(N)

2) Fixing:

Algorithm

Iterate over the errors array which is filled in the validation, and fixing the file by adding the missed tags from the errors and remove empty tags.

Time Complexity: O(N^2) **Space Complexity:** O(N)

3) Compression & Decompression:

Algorithm

Using Huffman algorithm, I first read the sample.xml as a single string then passed it to function ENCODE, inside ENCODE:

- A frequency array that takes the fileAsString and construct a frequency array of each character.
- A min-Heap that constructs a tree like heap with maximum frequent character on top.
- REPLACE function that actually encodes the character according to their location into the min-Heap tree, traversing to the left adds zero , to the right adds one so: the most frequent character gets 1 bit size while low frequent chars get bigger number of bits and these bits are completely distinct 4- then a function DECODE that takes the minHeap root and traverse to return the original file in last.txt.

ENCODE:

Time Complexity: O(N) **Space Complexity:** O(N)

DECODE:

Time Complexity: O(N) **Space Complexity:** O(1)

4) JSON:

- Read file line by line.
- Extract tags from line.
- Check tag level and its siblings.
- Use brackets based on the children of the tags and its parent's level.

Time Complexity: O(N^2) **Space Complexity:** O(N)

5) Minify:

- Get each line of the file.
- Delete spaces and new lines.

Time Complexity: O(N) **Space Complexity:** O(1)

6) Prettify:

- Read file line by line.
- Extract tags from line.
- Check tag level its siblings
- Use tabs based on the children of the tags and its parent's level.

Time Complexity: O(N) **Space Complexity:** O(1)

Phase Two:

1) Most Influencer User:

Algorithm

- Loop on users vector.
- Return user with max number of followers.

Time Complexity: O(N) **Space Complexity:** O(N)

2) Most Active User:

Algorithm

- Iterate over all the users and save followers of each user in an frequency array.
- Iterate over the frequency array and get the max follower with following frequency.

Time Complexity: O(N) **Space Complexity:** O(N)

3) Mutual follower between two users:

Algorithm

- Pass IDs of two users and users vectors as parameters to the function.
- Extract followers of target users in separate vectors.
- Fill frequency array with the followers of first user.
- Iterate over the followers of the second user and compare them with the frequency array to get the mutual followers between two users then Iterate over mutual IDs to get the users.

Time Complexity: O(NlogN) **Space Complexity:** O(N)

4) Suggest Users:

Algorithm

- Take vector of users and target id to show suggestions.
- Get IDs of followers of target user.
- Get users of these IDs.
- Get followers of the followers of Target user.
- Skip duplicated IDs and the target user ID from the followers of followers.
- Get users of these IDs and return result as Complete users.

Time Complexity: O(N^2) **Space Complexity:** O(N)

5) Post Search:

Algorithm

- Take target keyword.
- Look for target keyword inside a vector of pairs: first is User ID, second is Post object.
- Look in each post object inside the post body and inside each topic for the keyword.
- Return users with posts or topics matching the target keyword.

Time Complexity: O(N^2) **Space Complexity:** O(N)