

Project (XML to JSON converter)



Data Structure and Algorithms
(CSE331s)

Team Members:

1_ Zeyad Khaled Abdelkhalek

ID: 1804794

2_ Mamdouh Ibrahim Youssef

ID: 1806716

3_ Nada Abd El Latif Salah

ID: 1809465

4_ Mohamed Ali Mohamed Ali

ID: 1805980

5_ Mazen gasser Mohamed erfane

ID: 1808434

Contents

Building GUI	2
• Background:	2
Detecting different error in xml file (consistency)	3
• Background :	3
• Implementation Details :	3
• Complexity of operation :	5
Implementing tree class & Parsing data in tree Structure	5
• Background :	5
• Implementation Details :	6
• Complexity of operation :	6
Formatting (Prettifying) XML	7
• Background :	7
• Implementation Details :	7
• Complexity of operation :	7
XML to Json	8
• Background :	8
• Implementation Details :	8
• Complexity of operation :	9
Minifying xml file :	10
• Background :	10
• Implementation Details :	10
• Complexity of operation :	10
Prettify xml file :	11
• Background :	11
• Implementation Details :	11
• Complexity of operation :	12
Compressing the file :	12
• Background :	12
• Implementation Details :	13
• Complexity of operation :	14
Representing XML file using graph	14
• Background :	14
• Implementation Details :	14
• Complexity of operation :	15
References :	15

Building GUI

- Background:

XML files can be viewed with place for input and output to get a user experience, edited and saved in xml, txt or JSON extensions

Clear: clear the input and output places to use a new one

Browse: choose the file directory and show it in the input screen

Save as: save the edited file in the output screen in xml , txt or JSON extensions

Convert XML to JSON: convert the xml input file to json format

Correct Errors: add any missing closing tags and correct any invalid closing tags

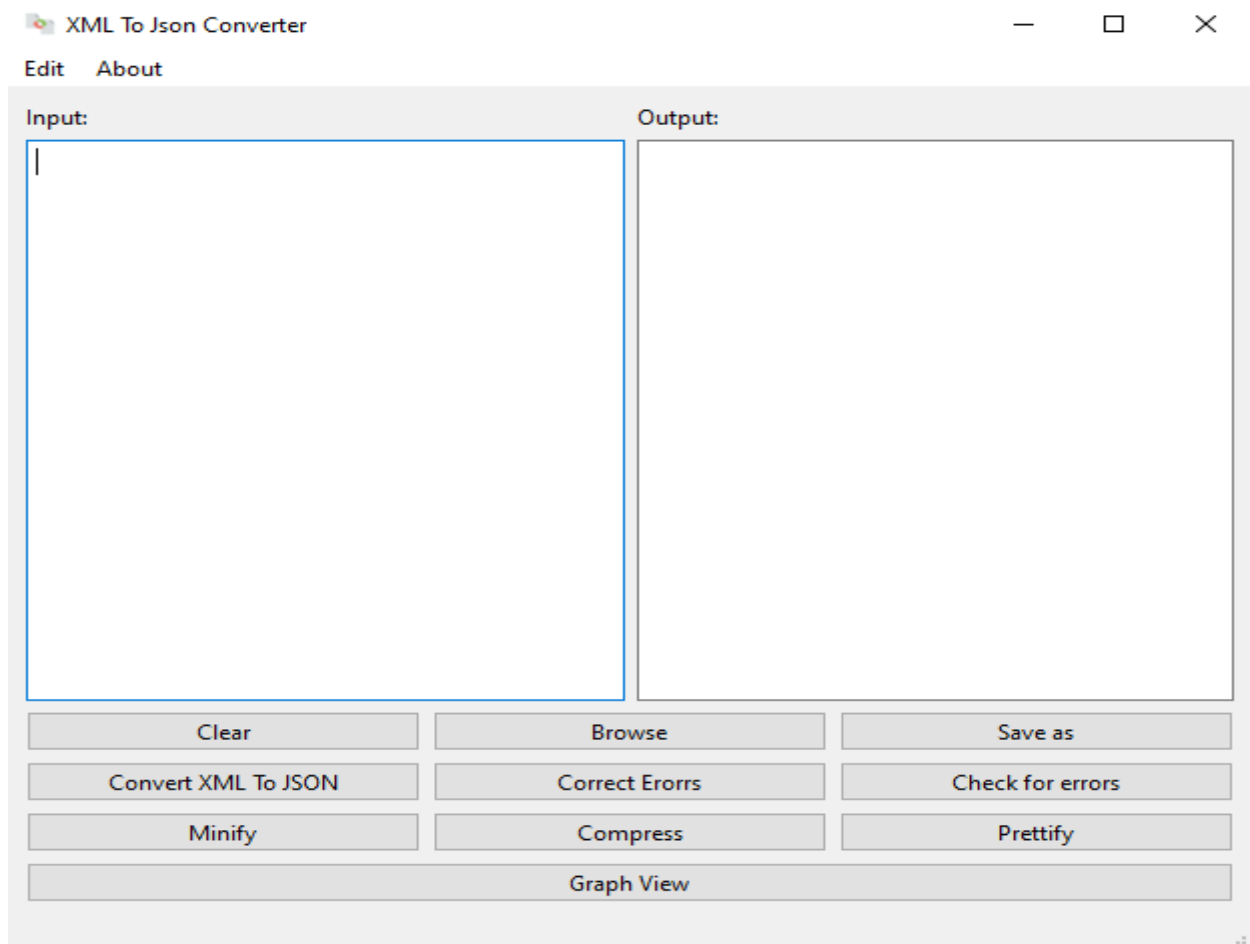
Check for Errors: add (-> error here) in positions of missing or incorrect tags and list with all invalid tags

Minify: remove any spaces to reduce file size

Compress: reduce file size and saved in the same directory of the exe file

Prettify: format the XML file with levels to get best experience to show the output

Graph View: View the XML file in Graph view to show the social network between users



Detecting different error in xml file (consistency)

- Background :

We simply divide checking the consistency of XML file into two main functions one to check errors of any missing closing tag or incorrect one and locate the error position with marker at the line with the error (-> error here) and another function to correct these errors with the right tags

- Implementation Details :

We used stack data structure to check and correct errors as we know it's working by pushing the data at the top of the stack and pop also from the top of it so it's called (LIFO) data structure , we divide the XML file into two categories outer tags which is the tags have an opening tag after it and inner tags which is the tags have no opening tags after it only string

⇒ **check XML Errors , check XML consistency :**

two functions we used 3 stacks the first one to store inner opening tags, second one to store outer opening tags, third outer temp opening tags if we want to check a tag not in the top and use another stack to store the tag names of the errors(only for check_XML_Errors) so the two functions are consisted of

- 1- functions take String data type and return also string data type value
- 2- A constrain in this method that: If it reads "<?\".....\"?>\" , it should erase it by erase() & substr() methods.
- 3- For loop that which iterate on the whole XML text if it finds '<' so it will go to execute the rest of the function else it will skip this loop
- 4- If it finds '<' so it will check the next if it wasn't '/' so it will be an opening tag and check first if the stack of inner tags isn't empty so the closing tag is missing so we need first to insert the closing tag of it in function check_XML_consistency or insert (-> error here) and push it in error stack in function check_XML_Errors and pop the top before get an extra open tag if the coming tag was neither outer or inner
- 5- check second if the stack of outer tags isn't empty so we need to search in the stack if the coming open tag is found in the outer opening stack which means the closing tags is missing so we need first to insert the closing tag of it in function check_XML_consistency or insert (-> error here) and push it in error stack in function check_XML_Errors and pop all the closing tags before it and pop it also
- 6- then we need to check the coming open tag if it has an open tag after it, it will be pushed in the (outer) stack and if it has a string after it will be pushed in the (inner) stack

- 7- If it finds '<' so it will check the next if it was '/' so it will be a closing tag and check first if the stack of inner tags isn't empty so it will check if the top of the stack and coming one are equal so it will pop it safely , else it will first insert the closing tag of it in function check_XML_consistency or insert(-> error here) and push it in error stack in function check_XML_Errors then it will need to check the coming tag in the outer stack if it's found so it will insert all the closing tags before it in function check_XML_consistency or insert (-> error here) and push it in error stack in function check_XML_Errors and pop them all from the stack else (closing tag is not found in inner or outer stacks) so we need to erase the closing tag in function check_XML_consistency or insert (-> error here) and push it in error stack in check_XML_Errors and pop it
- 8- If the inner stack is empty so it will check the outer stack if the top of it is equal to the coming tag, it will pop it safely else it will check all the stack if it's found it, it will insert all the closing tags before it in function check_XML_consistency or insert (-> error here) and push it in error stack in function check_XML_Errors and pop them all from the stack else (closing tag is not found in outer stack) so we need to erase the closing tag in check_XML_consistency or insert (-> error here) and push it in error stack in function check_XML_Errors and pop it
- 9- When it ends the whole string, we need to check inner stack if it wasn't empty insert all tags and pop them all, and also if the outer stack wasn't empty insert all tags and pop them all finally in function check_XML_Errors it will append the tags with errors at the end of the output string

⇒ **Input & output sample** : if tag is missing or incorrect

- **check_XML_Errors method :**

Input:	Output:
<pre> <users> <user> <id>1 <name>Ahmed Ali</id> <posts> <post> <body> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor </pre>	<pre> <users> <user> <id> 1 --> error here <name> Ahmed Ali --> error here</id> <posts> </pre>

- **check_XML_consistency method :**

Input:	Output:
<pre> <users> <user> <id>1 <name>Ahmed Ali</id> <posts> <post> <body> Lorem ipsum dolor sit amet, </pre>	<pre> <users> <user> <id> 1 </id> <name> Ahmed Ali </name> </pre>

- Complexity of operation :

Time complexity : It's implemented using stack so any push , pop or top methods it takes $O(1)$ so in check_XML_Errors and in check_XML_consistency [the worst case when all the tags are outer tags except last one and the closing tags of them are all missing so it will take $O(n+m+x)$ which n is the string length , m is the size of outer stack , x is the size of inner stack else it will be approximately $\approx O(n)$ which n is the string length]

space complexity : It's implemented using stack so in check_XML_Errors it takes $O(n+m+x+y)$ which n is the size of outer stack , m is the size of inner stack , x is the size of outer temp and y is the size of error stack and in check_XML_consistency it takes $O(n+m+x)$ because we didn't use error stack

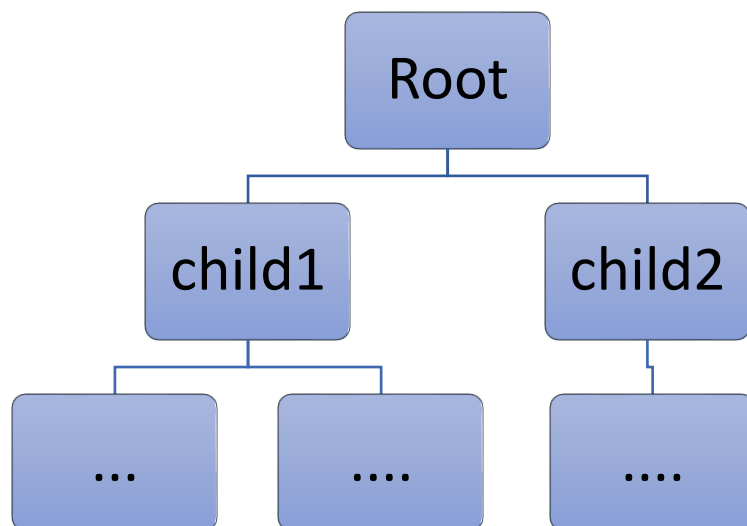
Implementing tree class & Parsing data in tree Structure

- Background :

After checking our consistency, here as we want to put our data in tree structure as shown the following diagram

level 0

level 1



So we could implement different functions and operations in easy way & to get use of Data structures which simplifies our Code.

- Implementation Details :

So first we should implement:

- 1- Tree & Node Class
- 2- parse Method to parse our data in the tree
- 3- Format Method.
- 4- XML to Json Method.

- **Node Class:**

A class which has 3 data fields (key, data, vector of pointers to children of the node) This class is friend class Tree;

- **Tree Class**

This class has a setter method & add_ Node method to enable us add new in parsing our file and represent our xml file in tree structure.

- **Parse method**

- 1- This method takes String data type and return tree data type value
- 2- First constrain in this method that: If it reads <? "....."?> , The method should erase it and parse the left data in tree by erase() & substr() methods.
- 3- It sets variables for opening & closing tag "<" ">" and start to parse the function by reading it char by char.
- 4- As long as closing tag "</" ">" isn't found → it creates new node
`Node*newnode = new Node;`
- 5- As long as the node has key only without having data that defiantly means this node has children so we push it in the stack `s.push(parent);`
- 6- Else: when the node has data value → in this case it must be a leaf node which has no children after it so it put the data value
- 7- By repeating this algorithm on the xml file we could parse it in tree.

- Complexity of operation :

Time Complexity $O(n)$, (n: number of characters)

Space Complexity $O(m)$, such we store data at vector of strings. (m: # of lines)

Formatting (Prettifying) XML

- Background :

In this stage we have our file XML parsed in tree, we need to format the tags in levels to show it in better and organized way

- Implementation Details :

- 1- This method takes the level which the pointer will point and a pointer to Node as parameters (int level = 0, Node *parent = nullptr)
- 2- By passing no argument to this function we will point to our root node, (as we are now in level zero).
- 3- Then we loop on children of this node If (children.size()>0) → so we will create pointer for each child Node *node = parent->children[i];

- For each child :

We will display its opening tag, then checking if the children of this node = 0, so it will print its data value and the closing tag → **leaf node**

- Else if the children of this node > zero so recursively call format method:
Output += format (level + 1, node); until it returns the final output

- 4- Handling indentation case by incrementing level by one so at level 0 spacing = "", in level 1 we have " "(increase one tap \t at each level of the tree). As shown in the following output

- Complexity of operation :

Time Complexity: O (m), looping line by line. (m number of lines).

Space Complexity: O (2n+m) ~O (n+m), such n is space of vector and m space of stack.

XML to Json

- Background :

Here we will repeat similar algorithm of formatting as our data are parsed in tree, then we can convert the xml format to any format like JSON

- Implementation Details :

- 1- The method takes the level which the pointer will point and a pointer to Node as parameters (int level = 0, Node *parent = nullptr)
- 2- By passing no argument to this function we will point to our root node, (as we are now in level zero).
- 3- Then we loop on children of this node and follow the previous algorithm but:
 - Instead of displaying opening tag we will display “key value”:
 - Adding more constraints if this node is not the last node we will print “, /n” if it is the last node in its level so will not print “, “only new line
- 4- Finally reaching to the last children, the last level we will display the closing curly bracket
- 5- We can handle the [array or object {by checking condition if the children of one node is repeated, it will be an object {...}, else it will be an array bracket[...]
- 6- The output of the previous algorithm will be as shown in the figure

```
"class": [{
  "student": {
    "id": "101",
    "firstname": "Naman",
    "lastname": "Kumar",
    "subject": "Math",
    "marks": "83"
  },
  "student": {
    "id": "102",
    "firstname": "Kapil",
    "lastname": "Kumar",
    "subject": "Chemistry",
    "marks": "60"
  },
  "student": {
    "id": "103",
    "firstname": "Harsh",
    "lastname": "Singh",
    "subject": "English",
    "marks": "70"
  },
  "student": {
    "id": "104",
    "firstname": "Jitesh",
    "lastname": "Singh",
    "subject": "Physics",
    "marks": "76"
  }
}]
}
```

- Complexity of operation :

Time Complexity: $O(n*t)$, n: number of tags, t number of char in tag name

Space Complexity: $O(h + c)$, where h height of the XML Tree, c number of characters in XML

Minifying xml file :

- Background :

We simply remove all spaces or '\t' and insert '\n' so every tag will be in separate lines so the file size will be decreased or view the XML to have a useful user experience

- Implementation Details :

We used string built in function to check any spaces between tags and remove it so the tags will move and string size will still the same so at the end of the function we need to remove the spaces from the end and resize it before return it

- minify XML file :

- 1- function takes String data type and return also string data type value
- 2- For loop that which iterate on the whole XML text if it finds '<' so it will go to execute the rest of the function else it will skip this loop
- 3- If it finds '>' so it will go until it reaches '<' and remove all spaces between them and insert a new line '\n'
- 4- Else it means it find a string between 2 tags so it will remove all spaces between first tag and string then insert a new line '\n' and then move until it finds '<' and check if the string so it will break else it will increase iterator by 1 until it finds '<' and remove all spaces before, until it reaches the string
- 5- After it finishes all string it will call a function to remove all spaces from the end and resize the string with the new size

Input & output sample :

Input:	Output:
<pre><users> <user> <id>1</id> <name>Ahmed Ali</name> <posts> <post> <body> Lorem ipsum dolor sit amet, ... </body> </post> </posts> </user> </users></pre>	<pre><users> <user> <id> 1 </id> <name> Ahmed Ali </name></pre>

- Complexity of operation :

Time complexity : worst case is $O(n*m)$ which n is the string length , m is the number of spaces and best case is approximately $O(n)$ when the string is already minified

space complexity : it takes $O(n)$ which n is the size of string

Prettify xml file :

- Background :

We simply add tabs before every new tag and increment number of tabs if a new outer tag comes and let the new tags with the same number of tabs if it was an inner tag and decrement the number of tabs if it was a closing outer tag and let the new tags with the same number of tabs if it was a closing inner tag

- Implementation Details :

We used stack data structure as we know it's working by pushing the data at the top of the stack and pop also from the top of it so it's called (LIFO) data structure, we divide the XML file into two categories outer tags which is the tags have an opening tag after it and inner tags which is the tags have no opening tags after it only string

- 1- functions take String data type and return also string data type value
- 2- For loop that which iterate on the whole XML text if it finds '<' so it will go to execute the rest of the function else it will skip this loop
- 3- If it finds '<' so it will check the next if it wasn't '/' so it will be an opening tag and check first if it has an open tag after it, it will be pushed in the (outer) stack, insert a tab after it insert a new line if it doesn't have a new line then increase number, if it doesn't have a tag after it, it will do the same as before but it doesn't increase the number of tabs then it will be pushed in the (inner) stack
- 4- If it finds '<' so it will check the next if it was '/' so it will be a closing tag and check first if the stack of inner tags isn't empty so it will add the number of tabs and pop the top of the stack but if it's empty then it will be a closing outer tag and so it will add the number of tabs and pop the top of the stack and decrement 1 from number of tabs

Input & output sample :

Input:	Output:
<pre><?xml version="1.0"?> <class> <student> <id> 101 </id> <firstname> Naman </firstname> <lastname>Kumar</lastname> <subject>Math</subject> <marks>83</marks> </student> <student> <id> 102</id> <firstname>Kapil</firstname> <lastname>Kumar</lastname> <subject>Chemistry</subject> <marks>60</marks> </student> <student> <id> 103</id> <firstname>Harsh</firstname> <lastname>Singh</lastname></pre>	<pre><?xml version="1.0"?> <class> <student> <id> 101 </id> <firstname> Naman </firstname> <lastname> Kumar </lastname> <subject> Math </subject> <marks> 83 </marks> </student> <student> <id> 102</pre>

- Complexity of operation :

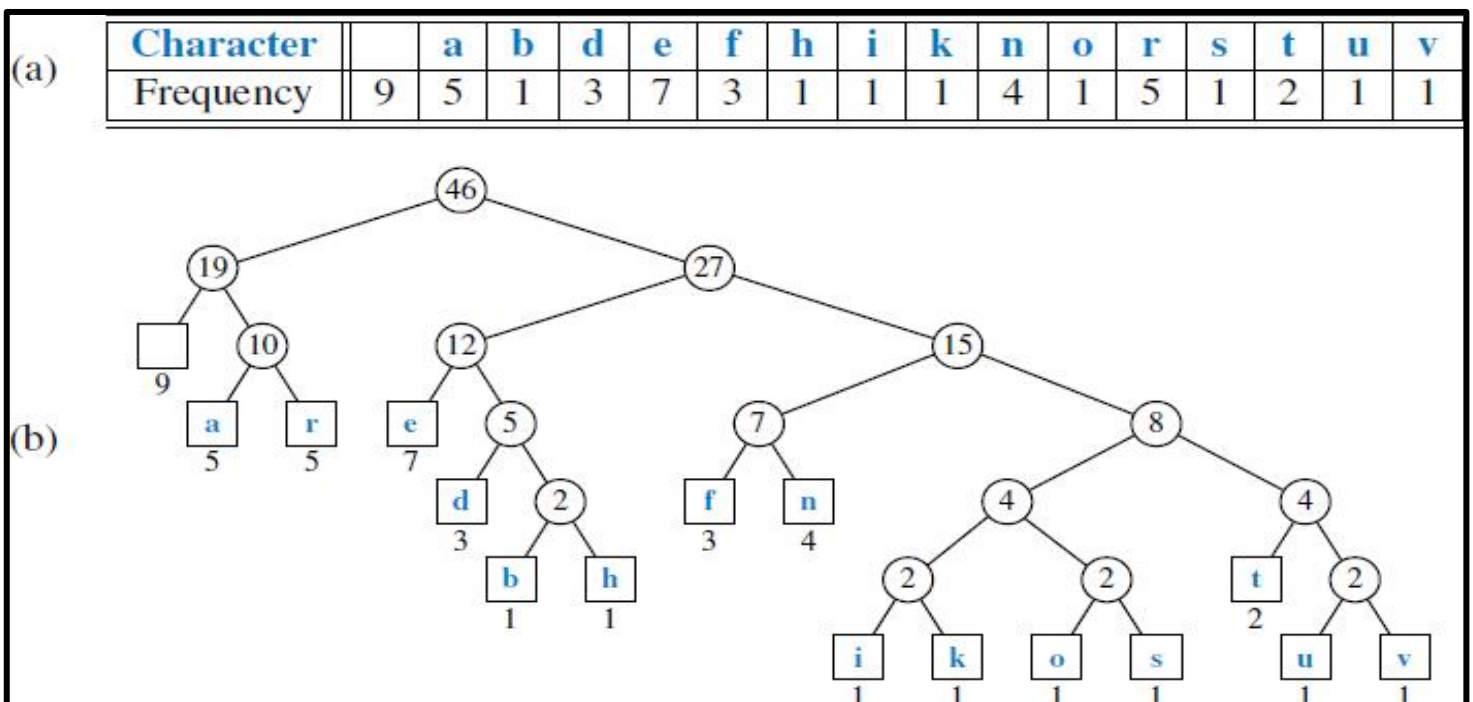
Time complexity : worst case is $O(n*m)$ which n is the string length , m is the # of tabs

space complexity : it takes $O(n+m)$ which n is the size of outer stack and m is the size of inner stack

Compressing the file :

- Background :

- I needed to find a lossless compression technique to allow the original data to perfectly be constructed after it had been compressed.
- Huffman algorithm is a lossless compression technique that is suitable for our project.
- The main idea of Huffman algorithm is:
 - 1) Get the frequency of each character in the file.
 - 2) Build Huffman tree and assign binary code to each character
 - 3) Replace each character with associative binary code (Encode or compression)
 - 4) The most frequent character gets the smallest binary code and the least frequent character gets the largest binary code.
 - 5) Replace each binary code with associative character (decode or decompression)



The more the character has been repeated (Higher frequency), the closer it will be to the root of Huffman Tree.

- Implementation Details :

- 1- Implement function to create node of a tree either left or right.
- 2- Compare between nodes with respect to frequency of repetition as the more the node has been repeated (Higher frequency), the closer it will be to the root of Huffman Tree.
- 3- Traverse the Huffman Tree and store Huffman Codes in a map>>encoding
- 4- Put "0" if we go to left a "1" if we go to right.
- 5- At the opposite direction traverse the Huffman Tree and decode the encoded string
- 6- Building Huffman Tree with given input text
- 7- Create a priority queue to store live nodes of Huffman tree
- 8- Create a leaf node for each character and add it to the priority queue
- 9- Remove the two nodes of highest priority from the queue
- 10- Create a new internal node with these two nodes with frequency equal to the sum of the two nodes' frequencies and add the new node to the priority queue.

Input & output sample : (I have chosen an arbitrary string for test)

⇒ Input : (I have chosen an arbitrary string for test)

```
// Huffman coding algorithm
int main()
{
    string text = "Data structure project <Compression part> using Huffman Algorithm in C++.";
    string encoded = Huffman_encoding(text);

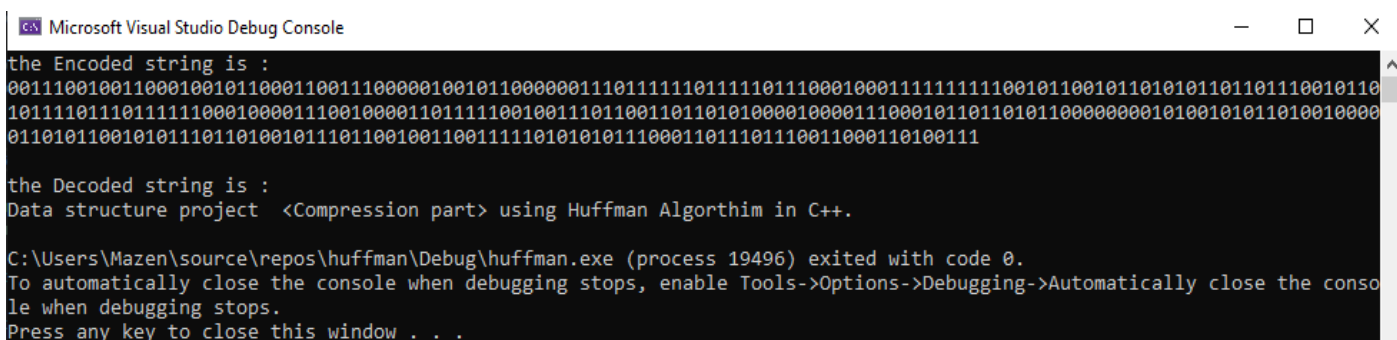
    cout << "the Encoded string is :\n" << encoded << endl<< endl;

    string decoded = Huffman_decoding(encoded);

    cout << "the Decoded string is :\n" << decoded << endl;

    return 0;
}
```

⇒ output : (after encoding then decoding)



Microsoft Visual Studio Debug Console

```
the Encoded string is :
0011100100110001001011000110011100000100101100000011101111101111011100010001111111111001011001011010101101110010110
101111011101111110001000011100100001101111100100110110011011010100001000011100010110101100000000101001010110010000
01101011001011101101001011101100100110011111010101110001101110111001100110100111

the Decoded string is :
Data structure project <Compression part> using Huffman Algorithm in C++.

C:\Users\Mazen\source\repos\huffman\Debug\huffman.exe (process 19496) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

- Complexity of operation :

Time complexity : $O(\log(n))$

space complexity : $O(n)$

Representing XML file using graph

- Background :

We need to show XML file in graph, the XML file will represent the user's data in a social network (their posts, followers, ...etc). The user data is his id (unique across the network), name, list of his posts and followers, So we will represent the relation between the followers and users using the graph data structure as it will be very helpful for the network analysis

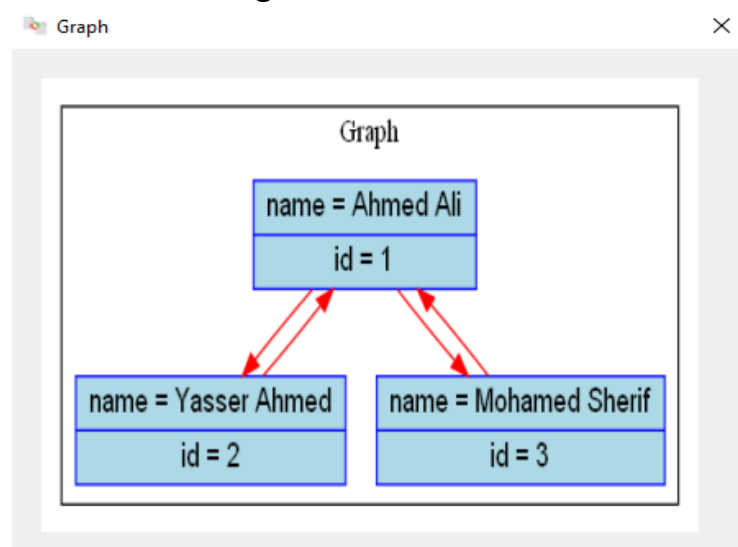
- Implementation Details :

- 1- Implement algorithm to create dot file in the format of graph which can be executed to PNG file showing social network between users.
- 2- We divide the algorithm into 2 functions first one to get user ID or name between opening and closing tags .
- 3- Second one which takes xml file as a string and searching about `<id>`, `<name>`, `</user>` and `<follower>` to extract name, id of the user and id of each follower.
- 4- Append the extracted ids and names in a string variable in the same format of dot language.
- 5- Execute the file and then show the PNG in GUI

Input & output sample :

⇒ Choose an arbitrary XML file with `<id>` or `<id>` and `<name>` tags

```
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>
      </post>
    </posts>
    <topics>
      <topic>
        economy
      </topic>
      <topic>
        finance
      </topic>
    </topics>
  </user>
</users>
```

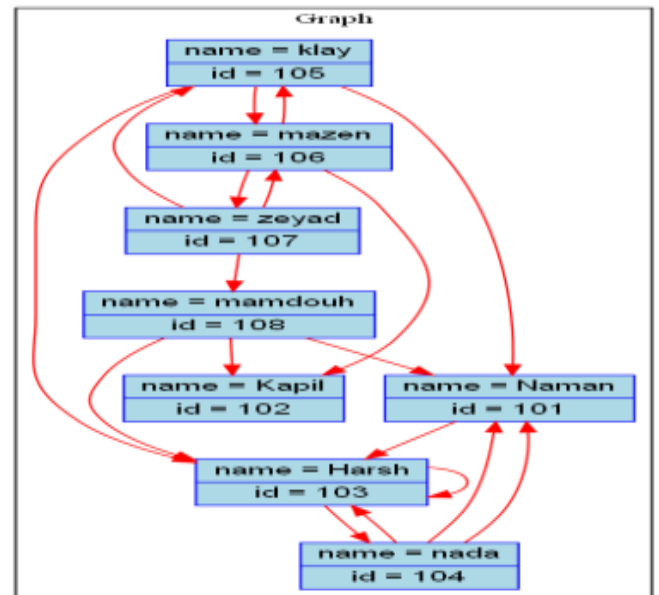


Input:

```
<?xml version="1.0"?>
<class>
  <user>
    <id> 101 </id>
    <name> Naman </name>
    <followers>
      <follower>
        <id>103</id>
      </follower>
    </followers>
    <subject>Math</subject>
    <marks>83</marks>
  </user>

  <user>
    <id>102</id>
    <name>Kapil</name>
    <subject>Chemistry</subject>
    <marks>60</marks>
  </user>
```

Graph



- Complexity of operation :

Time complexity : $O(n)$, which n is the number of open tags like `<id>`, `<name>`, `<follower>` and `</user>`

space complexity : $O(n)$, which n is the size of string which holds open tags like `ids` and `names` of users

References :

- Geeks for Geeks : <https://www.geeksforgeeks.org/>
- GitHub : <https://github.com/zeyadkhaled1/XML-Project>
- Geeks for Geeks : <https://www.geeksforgeeks.org/huffman-coding-using-priority-queue/?ref=gcse>
- Qt_Forum : <https://forum.qt.io/user/zeyad-khaled>
- Stack Overflow : <https://stackoverflow.com/questions/70393068/string-into-binary>
- GraphViz : <https://renenyffenegger.ch/notes/tools/Graphviz/examples/index>