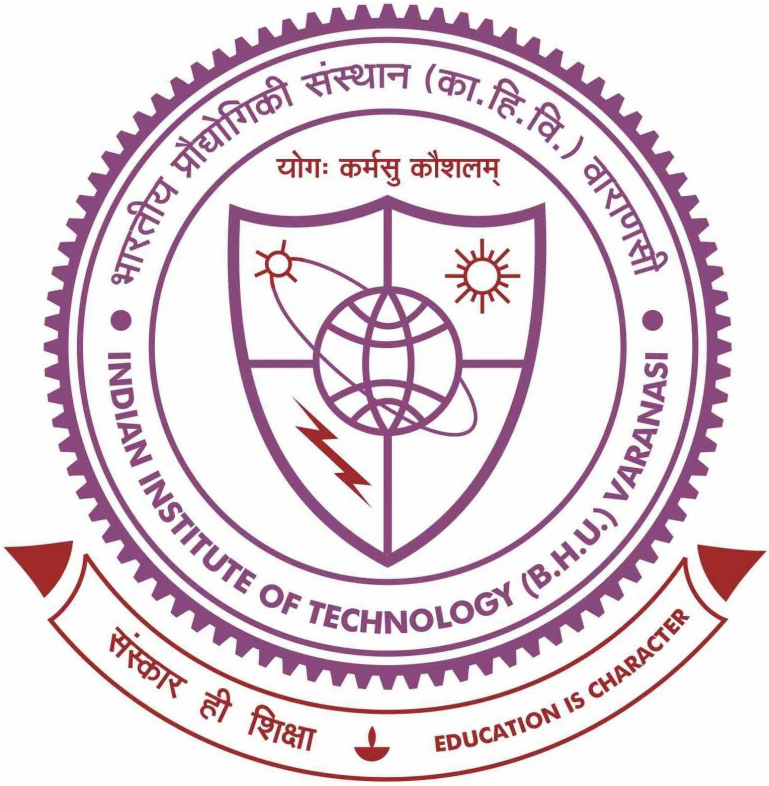
Implement a tool to generate XML document from a graph / business workflow



Name of Student: Kurra Bavanya CHoudhry
Roll No: 18075034
Department of computer Science & Engineering
BTech 3rd Year

Name of the Mentor: Dipty Tripathi
Department of computer Science & Engineering

Problem Statement

Develop an application which generates an XML document for a user provided graph. An XML document is a markup language which is both human and machine readable. Certain set of rules are to be followed for encoding documents in this language. Different file formats are developed based on XML like GRAPHML, GXL, XGMML etc. XGMML and GRAPHML are formats specifically developed for graphs based on XML. A Graph is a non-linear data structure which consists of a finite set of vertices(or nodes) and a set of Edges which connect a pair of nodes. The edges represent the relationship between the nodes. A directed graph has edges which point from one node to another node. A business work flow is a directed graph.

Objective

The objective is to generate an XML format because it is universally accepted by different applications working on areas related to graphs. The reason for this is that since XML language is based on a set of rules, it is easier for it to act as a common interface between different applications written in different languages and following different syntactical and representational rules for graphs. So this application can act as a common interaction point for different applications developed using different tech stack.

Abstract

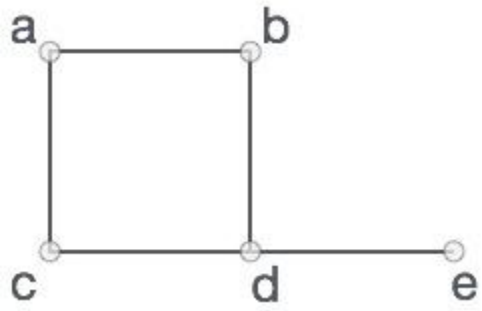
A web interface using flask framework is developed where users can provide a file with the graph information in a certain format required and the graphML generated will be displayed on screen and also the generated graphML file will automatically be saved to the disk. For graphical visualization a networkx graph will also be generated and displayed to the user on screen and the same png file is saved to the disk. The entire application followed a test driven and object oriented programming. Unit tests were written to all the core logic functions using the unittest framework in python.

CHAPTERS INDEX

Introduction

A **Graph** is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as vertices, and the links that connect the vertices are called edges.

Formally, a graph is a pair of sets (V, E), where V is the set of vertices and E is the set of edges, connecting the pairs of vertices. Take a look at the following graph −



In the above graph,

V = {a, b, c, d, e}

E = {ab, ac, bd, cd, de}

The details of a graph is taken by the application in a file format from the user's disk.

GraphML is a comprehensive and easy-to-use file format for graphs. It consists of a language core to describe the structural properties of a graph and a flexible extension mechanism to add application-specific data.

The format of graphML is XML based and describes the graph via XML elements such as <graph>, <node>, <edge>, and so on. Properties of the elements are described in <data> elements that use a certain key value to correlate them with a property.

A graphML file is generated by our application in python and saved to user's disk automatically. The generated graphML is also visible to users on the front end.

An example of graphML is:

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
     http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
  <graph id="G" edgedefault="undirected">
    <node id="n0"/>
    <node id="n1"/>
    <node id="n2"/>
    <node id="n3"/>
    <node id="n4"/>
    <node id="n5"/>
    <node id="n6"/>
    <node id="n7"/>
    <node id="n8"/>
    <node id="n9"/>
    <node id="n10"/>
    <edge source="n0" target="n2"/>
    <edge source="n1" target="n2"/>
    <edge source="n2" target="n3"/>
    <edge source="n3" target="n5"/>
    <edge source="n3" target="n4"/>
    <edge source="n4" target="n6"/>
    <edge source="n6" target="n5"/>
    <edge source="n5" target="n7"/>
    <edge source="n6" target="n8"/>
    <edge source="n8" target="n7"/>
    <edge source="n8" target="n9"/>
    <edge source="n8" target="n10"/>
  </graph>
</graphml>
```

Networkx is a python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. Networkx is used by the application to generate graph visualization and display to the user in the front end. The generated visualization is also saved to the user's disk automatically.

Flask which is a micro web framework written in Python is used for developing the web interface.

Minidom is a Python library called Minimal DOM. It's a minimal implementation of the Document Object Model interface. A GraphML document is created by the application using this library and attributes, nodes and edges are added.

Unittest a unit testing framework in python is used to write tests to the functions written to read the user provided data, create a graph in python, visualize the graph and generate graphML document for it.

Generating GraphML

Seperate classes are created for node, edge and graph and other supportive classes Item and Attribute.

Since taking the information of the graph from the prompt is not feasible when the scale of data is huge, the information of the graph is taken from a file.

The format of the data in the file is:

node1 node2
node 1 node2
node1 node2 and so on

Where the number of lines in the file indicate the number of edges in the graph.
Each line gives the two nodes connected by an edge in the graph.

A graph will be created by taking the nodes and edges information from the file using networkx library.
This created graph will be visualized.

A graphml file is generated for this graph and saved to disk with specified name.
If the specified file doesn't have a graphml extension, then default file name will be considered for creating the graphml file.

A graphml_parser.py file is written which takes in a graph and generates and saves a graphML file at the specified directory. The graphml_parser uses minidom from xml.dom to create a document and attributes, nodes and edges are added.

To save the created GraphML document, the precautions taken are:

1. Immediately after taking the file name input, it is checked if the file name is provided as an empty string or without a graphml extension.

2. If the check is failed, the generated document will be saved to default graphml.graphml file.

All the classes and functions from other files are called and executed in the main.py file.

Unit Testing

Wrote unit test ensuring the following using unittest framework for python:

test_create_graph_from_file(self):

A sample input.txt file is taken and a graph is created from the content of the file.
The nodes and edges from the generated graph are obtained using nodes() and edges() functions from networkx library.
Using Counter from collections in python, it is asserted that the nodes and edges obtained from the generated graph are as expected.

test_write_to_graphml_using_networkx(self):

A sample preexisting networkx graph is called using the networkx library using nx.karate_club_graph().
A sample file with graphml extension is created for the sample networkx graph and asserted that the created graphml file is saved to the disk or not using os.path.isfile() function.

test_graph.py:

A small sample graph for testing is created in create_graph() function.
test_graph() function is written to assert the number of nodes and edges in the sample graph.
test_graph_search() function is written to assert that the DFS and BFS sequences are as expected for the sample graph.

test_graph_parser.py:

A small sample graph for testing is created in create_graph() function.
test_graph_io() function is written to assert that the graphml generated represents a graph with the expected number of nodes and edges.

Web Interface

Created a flask application which includes the following end points in the flask application:

1. **'/' end point** home page which has a navbar with a link to view graphml and form to enter the number of nodes. Then a complete graph is generated with those number of nodes.

   Instead of a complete graph, to create a more customized graph, A file can be uploaded with the customized graph information.

2. **'/complete_graph'** which does the following in:
   a. **Backend:**
      i. A complete graph is created with the number of nodes as given in the url.
      ii. A png image of the created graph is saved to disk.
      iii. A graphml file is created for the created graph and saved to disk.
      iv. The created graph png image is passed to the frontend file for display.

   b. **Frontend:**
      i. The png image is displayed on the frontend.

3. **'/graphml'** which does the following in:
   a. **Backend:**
      i. The recently created graphml file will be opened and passed to the frontend template.

   b. **Frontend:**
      i. The graphml file will be displayed.

4. **'/file'** which does the following in:

    a. **Backend:**
        i. Receives the file uploaded
        ii. A graph will be created based on the metadata given in the file.
        iii. Using matplotlib, graph.png will be created and saved to the disk.
        iv. Graphml file will be created for the graph and saved to disk.

    b. **Frontend:**
        i. The created networkx matplotlib graph visualization will be rendered on the frontend side.
        ii. The created graphml content will be visible on the frontend by going to the /graphml endpoint.

Conclusion

The application has been developed using object oriented programming and it is made sure that proper unit tests are written for crucial and risky functions. To accommodate large scale graphs, we have added the functionality of taking the metadata of the graph in a file. The file is then read and the respective graph is generated. For the user to make sure that the graph generated is as per their specifications, we have developed a visual representation of the graph generated and it is displayed to the user post taking the metadata file input. Therefore, the proper software development recommendations are followed and a user friendly application is developed.

References

1) https://www.geeksforgeeks.org/create-xml-documents-using-python/ to understand creating xml documents using mindom in python.
2) https://networkx.org/documentation/stable/tutorial.html to understand how to create a graph in networkx and get the visualization.
3) https://www.w3schools.com/python/python_file_write.asp to understand how to write to file and save and open a file and read.

4) Took help from https://www.python.org/doc/essays/graphs/ in implementing graphs in python.
5) Learned from https://www.meccanismocomplesso.org/en/programming-graphs-in-python-part-1/ to code the graph, node and edge in classes following oops concepts.
6) Followed http://graphml.graphdrawing.org/primer/graphml-primer.html and http://graphml.graphdrawing.org/primer/graphml-primer.html to learn about GraphML format.
7) Took help from https://stackoverflow.com/questions/53161395/how-to-draw-networkx-graph-in-flask and https://stackoverflow.com/questions/50728328/python-how-to-show-matplotlib-in-flask to code the functionality of displaying the generated graph on front end.
8) Followed https://www.geeksforgeeks.org/unit-testing-python-unittest/ and https://docs.python.org/3/library/unittest.html to write unit tests and learn about unittest framework.
9) Learned how to develop flask application from https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3