Name: Kurra Bavanya Choudhry
Roll Number: 18075034
From: BTech CSE 6th Semester

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

I am giving March 31th update and adding all the previous updates as well for your reference:

(January 15 Update):

The keywords relevant to the topic are graph database and XML document.

1. **Graph database:** It is a type of NOSQL database. Information is stored in nodes and edges. The edges represent the relationship between the nodes. Neo4j is a graph database.

2. **XML document:** It 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.

The steps to be performed by the application are:

1. Data should be imported from the graph.

2. The following information must be extracted from the graph.
    a. Nodes
    b. Edges and Weights
    c. Any other information about the connection between nodes

3. XML encoding should be generated following the well defined set of rules of XML format.

4. The generated encoding should be saved to disk in a file with .xml extension in the requested directory.

After literature review, the approach to develop an application on this topic is:

1. The graph data can be taken either from **neo4j or networkx** to generate the XML file.
    a. **Neo4j:** It is a graph database which also provides native java API to access the database content using java.
    b. **networkx:** It is a Python library for storing and studying graphs and networks.

2. If **Neo4j** is taken for the graph data, the application can be written in **java**. The reasons for choosing java are as follows:
    a. Spring framework which can be used by any java application has support for neo4j for storing and extracting data. With this, the content of a database can be accessed to generate an XML file.
    b. The DOM parser in java supports creating an XML file effortlessly.

3. If **networkx** is taken for graph data, the application can be written in **python** as networkx is a library developed in python. Then the content in networkx data can be extracted and stored in **graphML** which is based on XML.

(January 31 Update):

Was able to integrate neo4j with java spring framework and create a sample graph using spring Data Neo4j and also with Spring Data Rest which takes the features of Spring HATEOAS and Spring Data Neo4j and automatically combines them together.

I have done the code explorations using spring framework and neo4j taking reference from the following links:

1) Getting Started | Accessing Neo4j Data with REST
2) Getting Started | Accessing Data with Neo4j

But started the project work in python because of the following reasons:

1) python has additional graph visualization packages like networkx and matplotlib
2) It is possible to integrate neo4j with python projects using the official drivers provided by neo4j.

Work done till now in python for the project:

1) Following object oriented programming for the project.

2) Created classes for node, edge and graph and other supportive classes Item and Attribute and a graphml_parser.py file which takes in a graph and generates and saves a graphML file at the specified directory.

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

4) Took the information of the graph to be created like:
    a) number of nodes
    b) names of nodes
    c) number of edges
    d) nodes with edges between them.

    From the command line and created the nodes using Node class in node.py and graph using Graph class in graph.py.

5) Breadth first search and Depth first search is also obtained with the code from graph.py and printed on the command line for the graph created.

6) The visualization of the graph is done using matplotlib and networkx. Networkx graph is created by adding the nodes and edges and nx.draw() and plt.show() functions are used.

7) The graphml_parser uses minidom from xml.dom to create a document and attributes, nodes and edges are added and written to the file provided in the command line.
    a) 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.
    b) If the check is failed, the generated document will be saved to default graphml.graphml file.
8) The work done till now in python is herewith attached. Also the generated graphml files are attached.

References for the python work:

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.

(February 15 Update):

1) Wrote 2 unit test files using unittest framework for python:
    a) test_graph.py:
        i) A small sample graph for testing is created in create_graph() function.
        ii) test_graph() function is written to assert the number of nodes and edges in the sample graph.
        iii) test_graph_search() function is written to assert that the DFS and BFS sequences are as expected for the sample graph.
    b) test_graph_parser.py:
        i) A small sample graph for testing is created in create_graph() function.
        ii) test_graph_io() function is written to assert that the graphml generated represents a graph with the expected number of nodes and edges.
2) Added type hints to the files for code documentation.

3) Added a function to add an edge between 2 nodes by their ids. Added exception handling in the function for when no node exists with a given ID.

(March 2 Update):

1) Since taking the information of the graph from the prompt is not feasible when the scale of data is huge, adding an additional functionality of taking the information from a file.
    a) The format of the data in the file will be:
        node1 node2
        node 1 node2
        node1 node2 so onnn...
        i) Where the number of lines in the file indicate the number of edges in the graph.
        ii) Each line gives the two nodes connected by an edge in the graph.
    b) A graph will be created by taking the nodes and edges information from the file using networkx library.
    c) This created graph will be visualized.
    d) An graphml file will be generated for this graph and saved to disk with specified name.
        i) If the specified file doesn't have a graphml extension, then default file name will be considered for creating the graphml file.

2) Wrote a unit test file(test_networkx,py) with two test functions using unittest framework for python:
    a) test_create_graph_from_file(self):
        i) A sample input.txt file is taken and graph is created from the content of the file.
        ii) The nodes and edges from the generated graph are obtained using nodes() and edges() functions from networkx library.
        iii) Using Counter from collections in python, it is asserted that the nodes and edges obtained from the generated graph are as expected.
    b) test_write_to_graphml_using_networkx(self):
        i) A sample preexisting networkx graph is called using the networkx library using nx.karate_club_graph()
        ii) 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.

(March 15 Update):

1) Created a flask application.
2) Created the following end points in the flask application:
    a) **'/' 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 will be generated with those number of nodes. A file can be uploaded with the customized graph information.
    b) **'/graph'** which does the following in:
        i) Backend:
            (1) A complete graph is created with the number of nodes as given in the url.
            (2) A png image of the created graph is saved to disk.
            (3) A graphml file is created using the networkx package for the created graph and saved to disk.
            (4) The created graph png image is passed to the frontend file for display.
        ii) Frontend:
            (1) The png image is displayed on the frontend.
    c) **'/graphml'** which does the following in:
        i) Backend:
            (1) The recently created graphml file will be opened and passed to the frontend template.
        ii) Frontend:
            (1) The graphml file will be displayed.
    d) '/file' which does the following in:
        i) Backend:
            (1) Receives the file uploaded
        ii) Frontend:
            (1) Returns a strings 'file uploaded successfully'

(March 31 Update):

1) Changed endpoint of **'/graph**' to '**/complete_graph**'.
2) Last time implemented only taking file from frontend and displaying a file received successfully message on frontend once the file is received by the

backend. This time, completed this functionality by adding the following in the backend function: (for the endpoint '**/file**')

a) A graph will be created based on the metadata given in the file.
b) Using matplotlib, graph.png will be created and saved to the disk.
c) Graphml file will be created for the graph and saved to disk.
d) The created networkx matplotlib graph visualization will be rendered on the frontend side.
e) The created graphml content will be visible on the frontend by going to the /graphml endpoint.