Data Analysis & Visualization - **Assignment 2**

Intelligent Graph Layout using d3.js

Due Date: 25th November 2023 (1159 pm) Marks: 100

Objective:

In this assignment, your objective is to create an Intelligent dashboard for graph layout from scratch using d3.js. The purpose of this assignment is to train you to be able to create custom layouts as per your own choice. The short description of the assignment is as follows: The dashboard should enable users to upload a file containing an adjacency matrix. Your primary tasks include identifying cycles, determining connectivity, classifying the graph as a General Graph, Directed Acyclic Graph (DAG) or a Tree, and laying out (visualizing it) appropriately it on the screen. The user will be provided with layout options according to the determined type of the graph. You must divide the code in different html and js files as described below. **Note: For laying out graphs, you CANNOT use any pre-built layout algorithm in d3 or any other source. You must lay out the graph from scratch using d3.js. Please read the evaluation strategy of the assignment before starting to implement it.** The detailed description of your assignment is as follows:

1. **UI Design & Development (index.html): [25 marks]**

   You have to create a user-friendly design of your Intelligent Graph Layout webpage. You may take inspiration from these links (https://www3.nd.edu/~cwang11/graphvisual/ & https://www3.nd.edu/~cwang11/treevisual/). The exact design of your web page UI is dependent on you. The UI must contain the following features i.e., Develop a user-friendly file upload feature, allowing users to upload a file that contains an adjacency matrix. Analyze the uploaded graph data to identify cycles. As a first step, determine the graph's connectivity. If the graph has disconnected components, ask the user to upload a connected graph. For a connected graph, determine if the graph is one of these special types i.e., a Directed Acyclic Graph (DAG) or a Tree or is it a general graph. Based on the type of the graph, the UI should intelligently offer the users appropriate layout options on the screen to select how they want to visualize the graph. The following options layout options should be shown as per the graph are as follows:

   - General Graph:
     1. Grid Layout
     2. Chord Diagram
   - Directed Acyclic Graph:
     1. Sugiyama Layout
     2. Radial Sugiyama Layout
   - Tree:
     1. Rheingold Tilford Layout
     2. Icicle Tree Layout

2. **Grid Layout (grid.js) [10 marks]**

   The general graph will be laid out according to the Grid Layout as discussed in class lectures only. Any other grid layout will not be acceptable. The nodes should be laid out as per your chosen criteria (e.g., alphabetically, degree based etc.). The structure of grid will be decided on number of nodes (may be edges too) in the graph.

3. **Chord Diagram (chord.js) [10 marks]**

   The general graph will be laid out according to the Chord Layout as discussed in class lectures only. Any other chord layout will not be acceptable. The edges/chords will be laid out using SVG curves as discussed in the lab lectures. The nodes should be laid out as per your chosen criteria (e.g., alphabetically, degree based etc.). The radius of Chord Layout will be decided according to number of nodes in the graph.

4. **Sugiyama Layout - Node Layers Calculation (Sugiyama-layers.js) [10 marks]**

   For either of Directed or Radial Sugiyama Layout, the first step is calculating the layers of all nodes of the graph. The heuristic to calculate the layers was discussed in class lectures. Any other Sugiyama Heuristic will not be acceptable.

5. **Directed Sugiyama Layout – Final Node Calculations (directed-sugiyama.js) [10 marks]**

   Use the node layers calculated in Node Layers Calculation step and layout the nodes in directed Sugiyama Layout form. For a simpler implementation you can ignore crossing minimization step.

6. **Radial Sugiyama Layout – Final Node Calculations (radial-sugiyama.js) [10 marks]**

   Use the node layers calculated in Node Layers Calculation step and layout the nodes in a radial Sugiyama Layout form. For a simpler implementation you can ignore crossing minimization step.

7. **Rheingold Tilford Layout (reingold-tilford-tree.js) [15 marks]**

   The tree will be laid out according to the Rheingold Tilford Layout as discussed in class lectures only. Any other implementation of Rheingold Tilford layout will not be acceptable. You may use bottom-up traversal using Depth First search strategy. For merging subtrees use the strategy discussed in class. As a hint, for bottom-up traversal strategies you may need to rearrange/redraw the trees after the layout is calculated. This last step will help make the root as the center of the tree.

8. **Icicle Layout (icicle-tree.js) [10 marks]**

   Use the layout of nodes calculated in the last step. But you must draw the tree in form of an Icicle Plot now. For sizing criteria, consider that all the leaf nodes are equal sized. Layout the Icicle Plot as discussed in class lectures only. Any other Icicle layout will not be acceptable.

Note:

- Ensure that the dashboard is user-friendly and intuitive.
- **This is an individual assignment. Plagiarism of any kind will not be tolerated.**

Submission:

You will need to submit your assignment in a zip file i.e., combine all your work in one folder. Rename the folder as per ROLL-NUMs (e.g., 20i-0002) and compress the folder as a zip file. (e.g. 20i-0002.zip).

Evaluation strategy:

- **IMPORTANT**: Submission of your assignment **on time** is mandatory for being considered for evaluation. I would highly recommend submitting your assignment at least one hour before the closing time, preferably a day before the deadline. If you have any work left in the last hour, you may resubmit the assignment again – I believe Google Classroom lets you turn-in the assignment multiple times. If you fail to submit the assignment on time (even forget to turn it in), the assignment will be closed at 1159 pm. And no late submissions will be allowed.
- There will be a **detailed demo** of your assignment where:
    - You must explain the working of any part of the code.
    - Any code from your assignment may be deleted and you will have to rewrite it.
    - NOTE: If you fail to explain any part of your code or unable to rewrite the code appropriately, your marks may be deducted substantially from your whole assignment. In short, an unsatisfactory demo may lead to **ZERO** marks in the whole assignment, even If you have submitted the assignment in full. The only way to avoid it is to do the complete assignment by your own self only.
- There will be **at least one question** in your **final exam** from this assignment.