Samuel Cruz
CSCI 160
Final Report

Graph Visualizer:
Educational Tool for Generating and Visualizing Graphs

In the rapidly growing and evolving landscape of computer science, the visualization of complex data structures plays a pivotal role in understanding and analyzing intricate relationships within vast datasets. Every year, students spend numerous hours studying these data structures, attempting to imbed the structure visualization within their own heads.

Graph Visualizer was designed to empower students, developers, and all those interested in comprehending and interacting with graph-based information. As volume, complexity, and the importance of data continues to escalate, this tool serves as a helpful asset, offering an interface for graph interpretation. By combining visualization techniques with graph-related features, this Graph Visualizer provides a means to unlock valuable insights for all those learning about graph structures. This document delves into the key features, design principles, and the real-world applications of this graph visualizer, aiming to illuminate its potential for computer science prospects.

Project Statement:
This project is a visualizing tool for graph structures and several searches. This tool generates an arbitrary graph based on user-input and will visualize it. In addition to the graph generation, the tool also visualizes different search algorithms. The current algorithms are Breadth-First Search (BFS) and Depth-First Search (DFS).

Motivation:
As a computer science student who has and continues to learn about data structures, the challenges of learning different structures served as the primary motivation for the development of this visualizer. In recent years, there has been a remarkable surge in the enrollment of computer science students. However, with this increase, there has also emerged a significant

portion of individuals who struggle with understanding key topics within the field. As someone who intimately understands the struggles of learning data relationships, I wanted to develop a tool that could facilitate the learning process for others and bridge the gap between difficulty and comprehension. It is this personal project that propelled me to wanting to contribute, aspiring to provide access to effective tools that transform the daunting task of learning data structures into an easier and effective experience.

Project Milestones:

The tracking of my project milestones is complicated as I made a project transition after the completion of the midterm milestone. Prior to being a graph visualizer, the project was initially a Binary Tree Visualizer. Luckily, these two projects were not too different. In both the Initial Report and Midterm Report, I stated that I wanted the source code for both the tree generation and visualizations of the search algorithms to be complete. Translating these milestones to the Graph Visualizer, I was able to do both the graph generation and the visualizations for the algorithms. Ideally, in the future, a proper interactive GUI and several search traversals could be great additions for this project.

Challenges:

Throughout this project experience, the single most daunting challenge was simply evaluating my current knowledge and underestimating of how long each task would take. I had recognized this challenge in my previous reports as I understood the key algorithm and data structure concepts but had never implemented or visualized them. Since this project was a single-person team, the time consumption each task would take was crucial. If there was a moment I was stuck on a task, there was no one else but myself to aid in the debugging process. Due to this, the project suffered when I decided to learn how to implement a proper GUI for the initial project.

During the attempted implementation of the GUI, several changes were made to my project source code in attempt to accommodate for the designed GUI. After a long period of programming and attempted debugging, I dropped my idea for a GUI. However, by this point, the original source-code was poorly manipulated and highly illegible. Due to this, I was stuck with deciding between having to present a project without a demonstration or attempt to replicate my original code and take the chance of not being able to complete on-time. Out of desperation, I simply decided to scrap the project. After restarting, I decided to switch over to graph visualization as I felt that graphs are more relevant to the computer science student rather than a binary tree. I saw a larger appeal in developing a graph visualizer due to graphs' more complex and possibly confusing nature. I believe this project could have gone more smoothly if I had meticulously planned each step ahead rather than doing tasks on a needs-basis. I often found myself unprepared when programming and debugging, noticing faults within my own knowledge and understanding. During the in-person demonstration, I presented the visualizations for my project and received great constructive advice from my peers. However, there was a bit of confusion in the limitations of my code. It seems that many were confused with the node size limitations of the project because I had only demonstrated a graph with a size of 20 nodes. However, my project visualizes a graph based on the user input size. If the user desires, the file will visualize the graph based on the inputs, not just 20 nodes.

Project Overview/Architecture:

https://github.com/scruz567/GraphVisualizer

This project is a single Python-file that when run, will ask the user for 3 inputs:

- number of nodes (int)
- number of edges (int)
- search algorithm (string)
    - ("BFS" or "DFS")

After receiving the user input, the program will generate the graph and immediately begin to visualize the search process, highlighting each node at each step of the search traversal.

Libraries:

Matplotlib

- provides a high-level interface for drawing the graph.
- opens plot onto the screen, and acts as the figure GUI manager.

NetworkX

- provides functions to create graph structures.
- includes various algorithms for graph analysis.

Component Interaction:
Upon running the file:

3 values are read from the input (# of nodes, # of edges, type of search)

def generateGraph (nodesNum, edgesNum):
- Uses NetworkX's *gnm_random_graph* built-in function to create a random graph.
- Returns the graph created from *gnm_random_graph*.
    - Will only return the graph if it is a well-connected graph. If not, *gnm_random_graph* will continue to run until a connected graph is returned.

def visualizeSearch (search, graphName, graph, positions):
- This function visualizes the graph and the desired traversal.
- Uses NetworkX's *spring_layout* built-in function to determine the plot position for the nodes of the graph.
- Uses Matplotlib.pyplot functions to draw and display each node and edge of the graph.

def dfs (graph, start, visited = None):
- This function returns the Depth First Search order of the nodes.

def bfs (graph, start):
- This function returns the Breadth First Search order of the nodes.

*Further documentation is included in the GitHub repo seen above.

Further Resources:

MKS075. "Difference between BFS and DFS." *GeeksforGeeks*, GeeksforGeeks, 8 Aug. 2023, www.geeksforgeeks.org/difference-between-bfs-and-dfs/.
- This is a website that explains the major differences between breadth-first search and depth-first search. These are two of the searching algorithms that I implemented.