GatorPages

Members:

Abhiram Lingamsetty

James Luberisse

Evan Zhang

Github:

https://github.com/GatorPager/GatorPager/tree/new

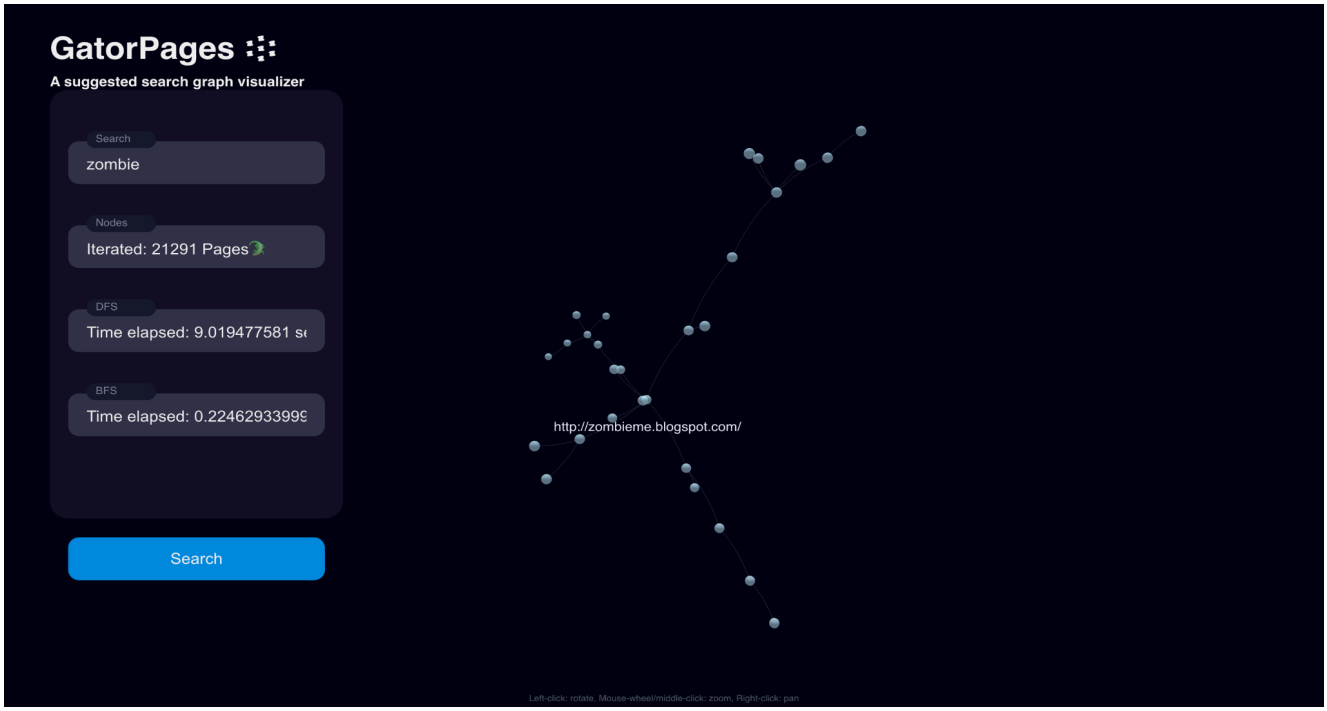
Video:

https://youtu.be/2-z07tp1g1M

Group Name: GatorPages

Abhiram Lingamsetty, James Luberisse, Evan Zhang

1. **Problem:** How do we graphically represent a set of websites returned from a bfs/dfs search?
2. **Motivation:** This problem is an attempt to grasp some perspective about the massively efficient search engines present today. Search engines today must search through exabytes of data and still return millions, even billions, of search results within seconds. This small project is an attempt to model said search engines and compare breadth first search and depth first search.
3. **Features**: This application will feature user-input of a keyword to search through a 100k tuple database of websites that fall under the category of art. We utilize a javascript library called Reactjs to visually represent a graph of web pages that contain the keyword that was searched. It will also display the time elapsed by our bfs/dfs algorithms.
4. **Data**: The database used contains a 100k set of websites that fall under the category of art. A node is assigned a URL. The edges are randomly assigned from one node to another, forming a graph of similar websites. Kaggle URL database (bit.ly/3rPgcc3)
5. **Tools/Languages/APIs/Libraries:** Python, C++, JavaScript, CSS, React, Replit (collaborative coding), srand.c, iostream.c, fstream.c, nodejs, axio (js), express (js), Microsoft Excel
6. **Visuals**:



An image of the GUI searching for a keyword "zombie"

7. **Graph Algorithms Implemented:**
    a. Breadth First Search

    b. Depth First Search
8. **Additional Data Structures/Algorithm:**
    a. Stack/Queue
    b. Ordered Map
9. **Distribution of Responsibility and Roles:**
    a. James: Creation of the graphical representations and backend program management, hosted ReactJS webapp with NodeJS..
    b. Evan and Abhiram: Conversion of parsed database information into .json files, .csv file management
    c. Everyone: Implementation of Graph algorithms, creation of the Project 3 document, and video.

Analysis

The first two proposals were completely changed due to our lack of understanding in what the project was asking for. Our initial proposal was a minecraft seed rater using an n-ary tree and a graph, but because we did not understand that this was a project meant for comparing graph algorithms, we had to change our entire design. Our second proposal was very rudimentary as we were still spotty on what we should do. We did not scope our second proposal on a real life problem, but rather was just a benchmark between an adjacency list and an incidence matrix using breadth first search.

After consulting with a few TAs, we narrowed our proposal down to website searching, as we had also coincidentally ran across a website database on kaggle that had over 2.4 million tuples of data. We saw it appropriate to only focus on one of the categories found in the database. With that, for the third time, we reformatted our proposal to explore the issue of search engines and focus on modelling their searches with something similar.

Complexity Analysis

Functions	Big O Time Complexity	Description
DFS() - graph.py	O(n) - where n is size of dataset	This function utilizes iterative depth first searching to implement DFS. Because it searches through the entire graph to find URLs with similar keywords, it **must** search through the entire dataset. Therefore it is always O(n).
BFS() - graph.py	O(n) - where n is size of dataset	This function utilizes iterative breadth first searching methods to implement BFS. Like the DFS() function above, it **must** search through the entire dataset to find

		similar URLs. It is also always O(n).
nodeJSON() - main.cpp	O(p) - where p is size of parsed data that matches keyword in graph.py	This function iterates through every similar URL that is returned through the BFS/DFS functions and creates the node half of the JSON file used to graphically represent similar URLs on Reactjs. See sample of nodes.json below.
linkJSON() - main.cpp	O(p) - where p is size of parsed data that matches keyword in graph.py	This function iterates through every similar URL that is returned through the BFS/DFS functions and creates the link half of the JSON file used to graphically represent similar URLs on Reactjs. See sample of nodes.json below.


```
# {
# "nodes": [
        # { "id": "http://www.new.com/", "group": 1 },
        # { "id": "http://www.google.com/", "group": 1 },
        # { "id": "http://www.great.com/", "group": 1 },
        # { "id": "http://www.youtube.com/", "group": 1 },
# ],
# "links": [
        # { "source": "http://ww.google.com/", "target": "http://www.google.com/%22%7D,
        # { "source": "http://www.google.com/", "target":
"http://www.youtube.com/%22%7D,
        # { "source": "http://www.google.com/", "target": "http://www.new.com/%22%7D,
        # { "source": "http://www.youtube.com/", "target":
"http://www.great.com/%22%7D,
        # { "source": "http://www.youtube.com/", "target": "http://www.new.com/%7D
# ]
# }
```


Sample of nodes.json.

Reflection

The experience shared by all group members on this project can be described as excellent. There was an extraordinary amount of clear and concise communication between all group members. Everyone had matched each other's effort and commitment to completing and making this project work in a reasonable time frame.

We all faced challenges in terms of implementing multiple features using different programming languages, one of which was an incompatibility with c++ compilers. A .cpp file that almost got implemented compiled on a MSVC (Microsoft Visual Studio Compiler) compiler, but when run on the mingw compiler on Visual Studio Code IDE, it completely failed. It was then when we decided to switch to a python implementation of our graph. Aside from that, it was challenging to visualize our abstract data whilst implementing our adjacency list.

If we were to start the project over, we definitely should have started the project earlier. As mentioned earlier, we went through multiple iterations of our proposal because of our lack of understanding on the project parameters/purpose. Because of that, we held off on starting our project due to the fact that we needed approval of our proposal, which only got feedback two weeks after the release of the project.

The group individually learned different things as a result of this project. For James, he gained a better understanding of how to implement a solution using multiple languages/APIs. For Abhiram, he learned about different compiling issues across different platforms, and different methods of debugging. For Evan, he learned how to apply knowledge gained from Data Structures and Algorithms to databases in a practical setting.

References

[1] https://www.kaggle.com/shawon10/url-classification-dataset-dmoz?select=URL+Classification.csv

[2] https://docs.python.org/3/

[3] https://reactjs.org/

[4] https://www.educative.io/edpresso/how-to-implement-depth-first-search-in-python

[5] https://www.educative.io/edpresso/how-to-implement-a-breadth-first-search-in-python

[6] https://www.cplusplus.com/