Yash Gupta
John Kim
Mike Lee
Adish Patil

Final Report

Our final project used a dataset of subreddits that were downloaded from the Stanford database (SNAP) as a tsv (tab-separated values) file. The file was converted to a csv file using python to load into a pandas data frame. Only the information that included the source, destination, and link sentiment was used, and the file was again filtered to create one fully connected graph. The majority of the graph ADT was taken from lab_ml, which allowed easier implementation of the constructor. The project includes 4 implementations of graph algorithms, which are breadth-first search (BFS), depth-first search (DFS), iterative-deepening search (IDS), and strongly connected components (SCCs).

The majority of the testing was performed on 4 manually created csv files, which had different sizes and graph structures. Some of the edge cases were tested by passing in an empty csv file. For each test file, the answers to different functions were manually solved. In tests.cpp, the expected solutions were added as a vector of Vertex and after storing the answer from the function calls, each test case required the actual solution to exactly match the expected solution. Specifically for strongly connected components, the manually solved answer was inputted in sorted order (ex. {A, B, D, F, H}), and the actual solution was also sorted before being compared.

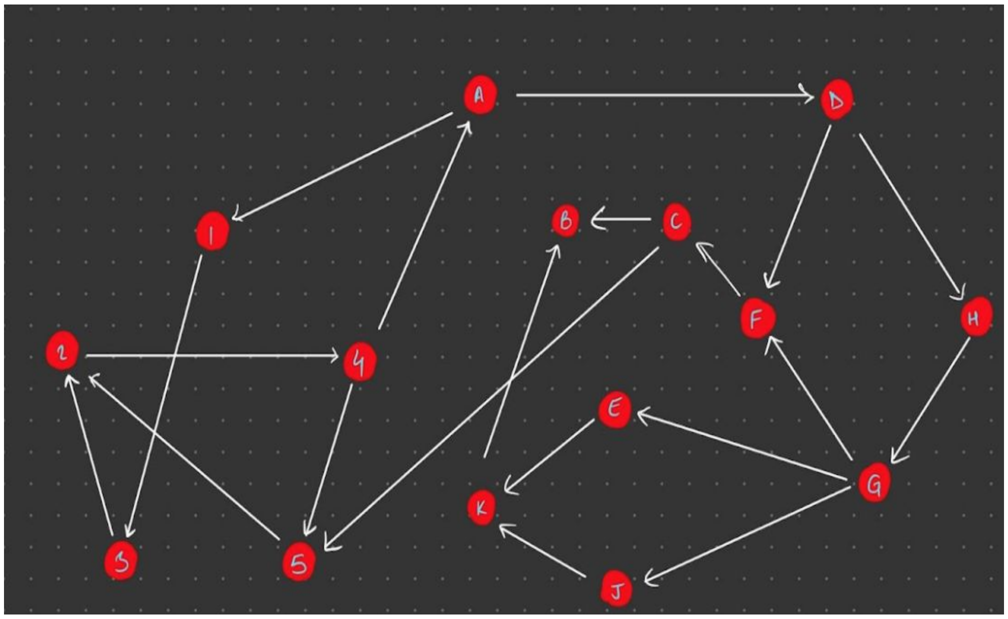
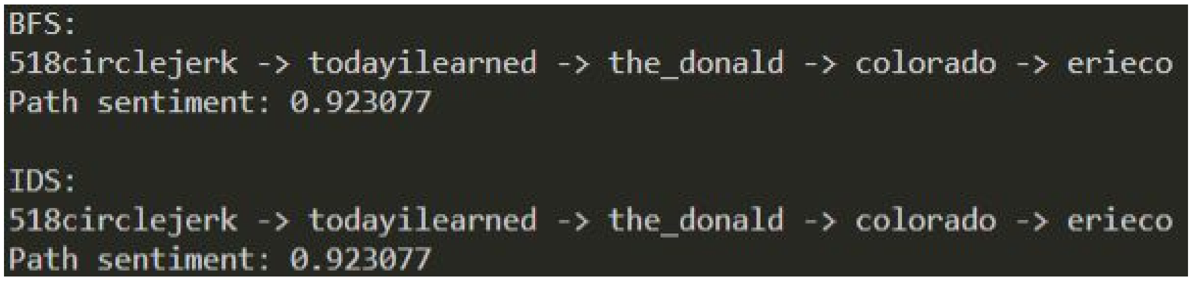


Figure 1. Sample Test Case Used

Through our testing, we were able to find unexpected outcomes in strongly connected components, runtimes in Kosaraju's Algorithm and Tarjan's Algorithm, and memory usage in IDS and BFS. When testing our strongly connected components on the real-world data, we found that 14,389 vertices were strongly connected out of 52,468. There were also 37,837 vertices with no strongly connected components. We were not expecting to see this kind of data where a huge number of subreddits were strongly connected. It was also surprising that there were only 11 different strongly connected components that had 3-4 vertices. We were expecting to see different categories of subreddits like movies, tv shows, politics, or sports, but instead, we are finding that most of the subreddits have some level of connection or have no connections with other subreddits. We also noticed that the runtime for Kosaraju's algorithm was faster than Tarjan's algorithm. Our Kosaraju's implementation was running in around 547 milliseconds and Tarjan's running in around 645 milliseconds. We discovered that this unexpected efficiency in Kosaraju's algorithm was due to our implementation of creating the transposed graph in the constructor so it did not have to perform an extra DFS traversal. You can see that if our Kosaraju's implementation had to do another traversal to transpose the graph, it would have taken 261 ms more. As a consequence of the transposed graph in memory, our Kosaraju's implementation allocates a lot more memory with 82 MB compared to Tarjan allocating 33 MB. Since we decided to implement both BFS and IDS to get the shortest path between two subreddits, we were curious to see how they compared in terms of their time and memory use. Theoretically, BFS is faster but uses more memory due to its use of a queue, whereas IDS is slower but uses less memory due to its recursive nature.



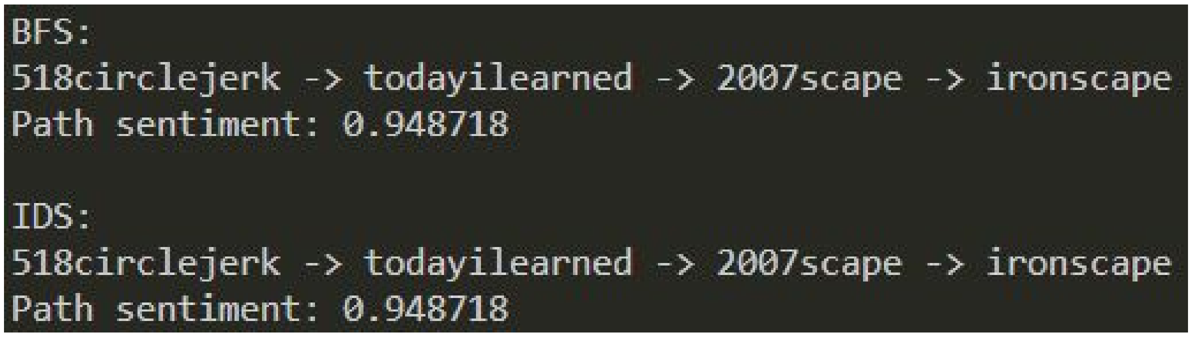
```
BFS:
518circlejerk -> todayilearned -> the_donald -> colorado -> erieco
Path sentiment: 0.923077

IDS:
518circlejerk -> todayilearned -> the_donald -> colorado -> erieco
Path sentiment: 0.923077
```

Figure 2. Final Run of BFS and IDS on "518circlejerk" and "erieco"

To test our implementations we searched between two subreddits "rarepuppers" and "erieco" which have the shortest path of 5 subreddits inclusive using both IDS and BFS. As expected, BFS was about 2.3x faster than IDS, however when we checked the memory allocated for each function call using Valgrind we were surprised to see that IDS allocated about 2.4x more memory than BFS did. After some brainstorming, we determined that this wasn't unexpected as the IDS implementation can visit vertices that are close to the source vertex multiple times and as a result will add and remove them from the map tracking visited nodes multiple times. So while the total memory allocated during the function call for IDS was greater, its maximum simultaneous memory allocation would be less than that of the BFS function call.

The final run of the project included printing the path traveled for BFS and IDS functions on the terminal. For the SCCs function, it was not feasible to accurately test on such a large dataset with more than 240,000 edges, so the result was compared to an output obtained using a python script that used the networkx library's strongly connected components function.



```
BFS:
518circlejerk -> todayilearned -> 2007scape -> ironscape
Path sentiment: 0.948718

IDS:
518circlejerk -> todayilearned -> 2007scape -> ironscape
Path sentiment: 0.948718
```

Figure 3. Final Run of BFS and IDS on "518circlejerk" and "ironscape"