Angelo Luna, Colin Myers, Justin Mikesell, Jacob Sanders

Mid-Project Deliverable for Cycle Detection:

Link to relevant data:
https://docs.google.com/spreadsheets/d/1IPwVmwoE5UuJCUhPkWT9QeUV7yPTEc6Xnr3THHmJEN4/edit?usp=sharing

I. Accomplishments

In the first phase of our project we were determined to find algorithms that met our requirements for cycle detection. The three algorithms we found and analyzed were Floyd (Tortoise and the Hare), Brent, and Depth First Search (DFS). Besides finding these, we've gained a deeper understanding of cycle detection which we then utilized in the next phase of this project; parallelizing one of the algorithms. Angelo was able to implement a parallel DFS-based algorithm. With one algorithm done, we should be able to find or create more parallel or distributed algorithms for cycle detection.

II. Discoveries and Analysis

For our study we tested 3 algorithms, Floyd, Brent, and Depth First Search. For each algorithm, we ran four tests, incrementing the number of vertices each time. For our vertices, we tested them with 10,000, 20,000, 40,000, and 80,000 vertices. Additionally, for each input size we ran 5 additional tests where we changed the location of the cycle we wanted to detect to give a more accurate representation of how the algorithm would perform on average.

The results we got for the 10,000 vertices were times ranging from 0.00005 to 0.00011 seconds for Floyd, 0.00007 to 0.00021 seconds for Brent, and 0.00052 to 0.0007 for DFS. The average times we had for each with 10,000 vertices were 0.00072 for Floyd, 0.00027 for Brent, and 0.00097 for DFS. Brent performed, on average, the best out of the three.

For an input size of 20,000, we had times ranging from 0.00011 to 0.00021 seconds for Floyd, 0.00020 to 0.00037 for Brent, and 0.00109 to 0.00127 for DFS. The average times we had for each algorithm with 20,000 vertices were 0 0.00016 seconds for Floyd, 0.00027 seconds for Brent, and 0.00120 seconds for DFS. Here, Floyd outperformed Brent while DFS remained the slowest.

The results we got for the 40,000 vertices were times ranging from 0.00024 to 0.00038 seconds for Floyd, 0.00075 to 0.00121 seconds for Brent, and 0.00408 to 0.00487 seconds for DFS. The average times we had for each with 40,000 vertices were 0.00031 seconds for Floyd, 0.00048 seconds for Brent, and 0.0023 seconds for DFS. For this input size, Floyd's average runtime was the fastest again while DFS remained to be the slowest.

Finally, we tested with 80,000 vertices and collected times ranging from 0.00036 to 0.00079 seconds for Floyd, 0.00038 to 0.00060 seconds for Brent, and 0.00213 to 0.00266 seconds for DFS. The average times we had for each with 80,000 vertices was 0.000608 seconds for Floyd, 0.001032 seconds for Brent, and 0.00438 seconds for DFS. In this last test we observed Floyd pulling considerably ahead of Brent while DFS fell way behind. It will be interesting to see how these algorithms perform with even larger input sizes after being parallelized.

III. Roadblocks

Our group faced a couple of roadblocks leading up to this point. First, our group has not met face-to-face once over the time period of our first phase; however, we do keep a group discussion open and talk as often as possible in that so we can meet deadlines and avoid the roadblock as much as possible. In the next phase we should meet more often in addition to our communication online in our group chat.

The second roadblock we faced was in our search for an already-implemented distributed algorithm for cycle detection. Unfortunately, after many hours of searching for a parallelized solution, we decided it was best to take our understanding of cycle detection algorithms and invest our time into creating one. Angelo took on the task and implemented a properly functioning parallel algorithm based on DFS. This was originally supposed to be part of our final deliverable, but seemed to be necessary to meet our mid-project deliverable. As such, our goals have shifted now more towards finding or making additional parallel or distributed algorithms for cycle detection and comparing them all at the end.

IV. Final Project Poster Elements

Our poster will include various aspects from this half and the second half our project. First it will include basic background information necessary to understanding cycle detection. Second, it will include the findings on the serial algorithms with brief descriptions and charts for the data. After the serial algorithms will come most of the poster; parallel or distributed algorithms for cycle detection. In this part we will discuss the parallel or distributed algorithms we have found or created during the course of the project. Each solution will be given a longer description compared to the serial and charts of data with run-times will also be on the poster. At the end will be a general section on what algorithms seem to work best and how they work best, as well as other parts of our discussions of the selected algorithms.

V. Progress and Deadlines

During the first half of this project, our group was able to meet the mid-project deliverable and go even slightly beyond. Originally, parallelizing an algorithm was an end-goal, but as mentioned before it was completed ahead of schedule. Our end-goal is now to find multiple parallel or distributed algorithms to be able to compare. In terms of progress, we are well-along and should be able to meet our final deadline with little difficulty and have a respectable end-product to present.