Name: Daniel Ems
Date: 02/20/2017
Current Module: Data Structures and Algorithms Capstone
Project Name: Zergmap

Project Goals:

This projects goal is to accept packet captures, and parse out the relevant data; gps packets, and status packets. Once the relevant information has been parsed, the developer must then create data structures and algorithms to complete the following objectives.
Objectives:
1. Determine which zergs are within the appropriate range.
2. Remove nodes as long as by doing so you do not remove more than half of the zerg.
3. Ensure the graph is a disjointed path, and modify appropriately if not, while abiding by objective 2.
3. Accept health parameters for which you print the zerg with health below that level.

Considerations:

- Figuring out how Dijkstra's works inside of code.
- Deciding which data structure to build a graph.
- Determining what properties to incorporate into the data structure that lends flexibility to being modified, or duplicated.
- Determine what will be need to solve the problems in addition to Dijkstra's
- Error handling user and file input
- Printing numerically.
- Determine how to modify and navigate the graph without losing a handle on the original data structure.
-

Initial Design:

The design for this project revolved solely around solving the Dijkstra's problem. Being that Dijkstra's was only a fraction of the problem, extensive hours of research went into to understanding how to program this from scratch. While the materials to replicate Dijkstra's were readily available, the lack of knowledge on how to implement the algorithm created an unnecessarily large time complexity to achieving a design. The result of which was first identifying how the problem could be achieved, and then attempt to build out a design form there. This was the only way myself, and other members of the class could appropriately approach the true test of this capstone. However, considering the lack of instruction on implementing graphs as well as Dijkstra's made the initial design a piece of constantly developing work for as we answered one question, we were left with many more.

Data Flow:

1. Read in files and re-implement the decoder code.
2. Check and error handle to ensure good packets are being passed.
3. Build out a graph as gps packets are passed.
4. Build out a BST as status packets are passed.
5. Merge the final graph with the BST so that all zerg are present in the health prints.
6. Check to remove all node's with less than 2 connections.

7. Check the removals to see if there are more removals than half of the node, if so, handle accordingly.
8. Check the remaining graph to see if it has the same number of connections throughout.
9. print removals
10. If no removals are necessary, then print accordingly.

Potential Pitfalls

There are several pitfalls present, however, the most notable of which revolves around the graph. While most edge cases have been handled, I. e. nodes with less than two connections, this is only a sliver of the problem. The true problem requires a hybrid algorithm that uses Dijkstra's the first time through, and then a modified version the second.

Due to our emphasis on theoretical knowledge of the concepts, the major pitfall is in essence a result of the lack of knowledge of practical application. The theoretical understanding of the concepts throughout this project has improved drastically, however, that does not translate in being able to efficiently program the solution in the time alloted.

That being stated, the largest pitfall was creating a new algorithm that solved the problem. There are no algorithms present, or provided that helps to solve this particular question. Furthermore, our lack of implementation with Dijkstra's hindered progress in the arena of developing such algorithm. As a result of which, you have received an incomplete assignment.

Test Plan

User Test:
1. Reading in ipv6 and ipv4 files
2. Read in gps and status
3. Error check appropriately, and handle appropriately.
4. Insert status packets in BST and gps packets into graph.
5. Repeat steps 1-4 until entire file is inserted.
6. update BST with graph nodes.
7. Print the insertion of nodes into both the graph and the BST.
8. Identify test case on the white board and possible solutions
9. Implement potential solutions and identify potential pitfalls.
10. Repeat steps 8 and 9 until the particular problem is solved.
11. Continue on to the next problem and repeat steps 8 and 9.
12. Check to ensure the new solution to the new problem does not conflict with prior solutions.
13. if step 12 creates an issue in prior solutions, go back and attempt to rework the original problem.
14. Continue until as many non conflicting solutions have been solved.

Test Cases:
1. Use the test files that were given by Liam for this project.
2. Use the test files provided by Liam for the codec project.
3. Attempt to identify potential pitfalls in your current application and attempt to address appropriately.
4. Repeat steps 1-3 for user input.

Conclusion:

The most effective aspect of this project was collaborating to solve the theoretical issues the problem at hand arose. This would primarily take place in the form of white board discussions and continuing analysis of a particular problem set. This led to a much better theoretical understanding of

graphs and the considerations necessary to solve the problem at hand. We were able to identify what was going to work, and what was not going to work, theoretically. As a result, some new concepts were derived that would have helped in the implementation of a complete algorithm.

The most detrimental aspect of this project was the lack of practical experience with both graphs and Dijkstra's. As was admitted by SSG Jaramillo, the corresponding exercises need to be reviewed. This admission confirms that we were left with no more than a theoretical understanding of the concepts were being asked to demonstrate. Furthermore, on a time line designed for students who received instruction on implementing the concepts.

An extension was granted, totaling 77 real hours, and 36 programming hours(12 hour days). This time, while appreciated, was still not enough. In a typical class room setting, we were told we were given a week dedicated to graphs and Dijkstra's. Over the course of a week, that would result in 40 hours of instruction from a trained professional to instill both theoretical and applicable understanding of the concepts. Without practical understanding, we were left with two weeks to teach our selves how to implement these concepts. Even if we were to say we could teach ourselves how to implement these concepts in the same time frame as a licensed professional, a 36 hour extension would still not suffice in giving us that time to do so.

Most importantly, as a result of the previous, I believe this capstone was unable to truly test us on what we understand, and what we can implement. The only metric for theoretical understanding on the grading rubric is the correctness of our code. This requires a fundamental understanding of how to implement the theoretical concepts taught. Without such knowledge, we are unable to truly answer the problem given.