Zergmap Design Document

Program Flow

1. Check for valid amount of args
2. Check for valid flags and flag values
3. Verify files can open
4. Create an empty graph
5. Read through every file and make nodes if the packet is valid
    a. Check for repeating node Ids
    b. Auto add edges if they are in range with each other
    c. Make connections from 3+ edged nodes to other 3+ edged nodes to be extra heavy
6. Remove any bad nodes (Ones with no GPS data)
7. Analyze the graph for invalid nodes
    a. Run Dijkstra twice
8. Print out all the bad nodes
9. Print out all the nodes with health that is below the threshold
10. Free the graph

Notable Data Structures

- Typedef struct _graph *graph
- Struct _graph
- Struct _data
- Struct _node
- Struct _edge
- Struct _stack

Notable Functions

- Graph graphCreate(void)
    - Creating and returning an empty graph
- Int graphAddNode(graph g, union zergH, struct gpsH *gps)
    - Adding a node to the graph
- Int graphAddStatus(graph g, union zergH zHead, struct statusH status)
    - Adding a status to a node or making a node with just a status
- Void graphAnalyzeGraph(graph g)
    - Analyzing the graph for bad nodes
- Void graphPrint(graph g)
    - Printing bad nodes
- Void graphRemoveBadNodes(graph g)
    - Removing any incomplete nodes (nodes without GPS data)

- Void graphDestroy(graph g)
    - Freeing the graph

Anticipated Challenges

- Keeping track of invalid nodes
- Marking nodes properly for the second Dijkstra sweep

Targeted Features

1. Man Page
2. Support Big-Endian
3. Ethernet 802.1Q
4. 6in4

Architecture

I will be using my decode code as the base for this project so I can read in the PCAP data. I will be using my graph library from my pervious projects. My graph utilizes linked lists for all the nodes and the edges. I will be using my Dijkstra that I made in the maze project. I will read in all the PCAPs passed to the program via command line arguments and will add each valid item into the graph. When a node is added into the graph, it will have edges auto added based on its position. I will make any edges from a node with 3+ edges that is connected to another node with 3+ edges weight extra. I will analyze the graph for bad nodes by running Dijkstra twice to see if it is a fully connected node. The first time I run Dijkstra, I will mark that path as a visited path, so the second time Dijkstra is ran, it avoids the previous path and tries to find a new one. The program will store any invalid nodes it finds in a stack. I will run this analysis on every node in the graph against every other node (Trying every possibility). I will only store the stack with the lowest number of invalid nodes. When the analysis is over, I will print out the invalid nodes and any nodes that are low on HP.