# ECE 368 Project Report

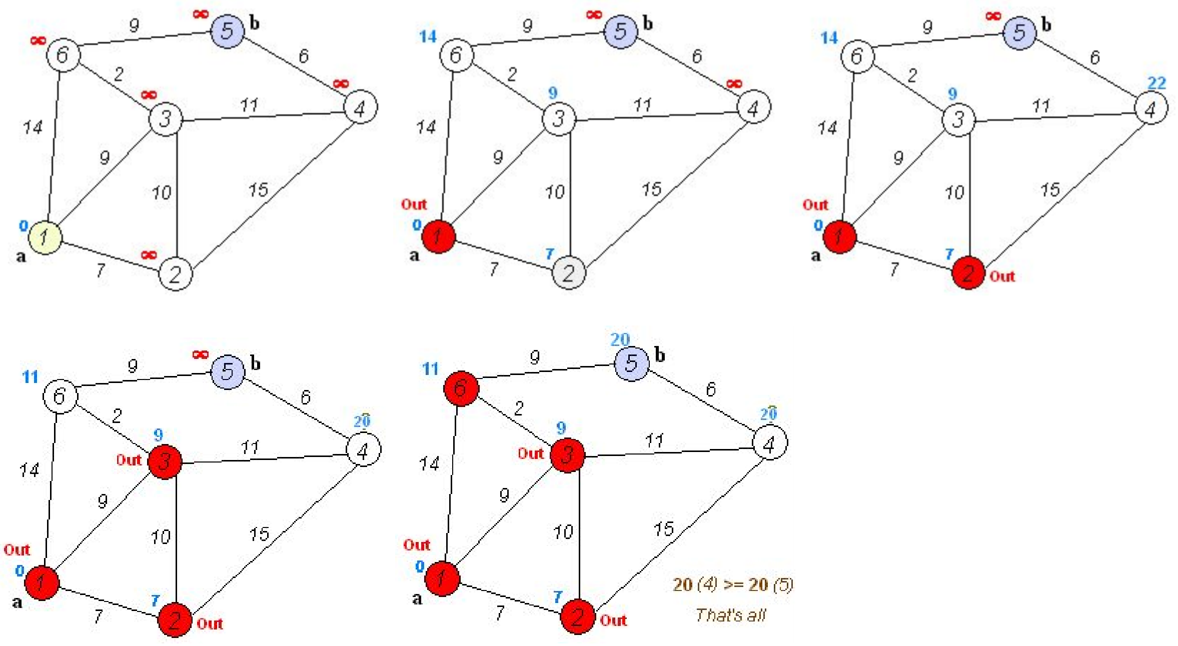
**Group 1:** Zane Johnson, Mark Brooks, Andrew Grossman, Yi Yang

## Executive Summary

The goal of this program is to find the shortest path between the given two nodes on the map input by the user.

## Data Structures and Algorithms

The algorithm applied to our project is Dijkstra's Algorithm. Dijkstra's algorithm picks the unvisited vertex with the lowest distance, calculates the distance through it to each unvisited neighbor, and updates the neighbor's distance if smaller. The algorithm marks a node visited when done with all of it’s neighbors. Below is a picture of how the algorithm works.



(Dijkstra’s Algorithm, Wikipedia, 3rd citation)

The main data structure used for the project is a directed graph. The graph is constructed using input from a data file provided by the user. Each vertex and its corresponding x and y coordinates is read from the file, and then added to the graph by the addVertex function. This function works by assigning the first vertex that is read as the head of a linked list. Each vertex thereafter is added to this linked list in the order it is read. The next section of the input file includes connections between the vertexes. These are assigned through the addEdge function. Each vertex that has at least one edge points to the head of a linked list of these edges. Each edge in this linked list points to some other vertex on the graph. The use of linked lists to implement the graph is beneficial as it is very efficient to add new vertexes and edges.

**Complexity Analysis and Optimization**

We store the directed graph as an adjacency matrix. The information about the edges is stored in a 2-D array. A priority queue of checking neighbor nodes is stored as an unordered list. Choosing the nearest neighbor node from the current node takes **O(n)** time. Updating distance for each neighbor takes **O(1)** time, with at most n neighbors. Checking for n nodes takes **O(n)** time and then the program deletes one vertex from the priority queue per loop. Therefore, the time complexity would be **O(n)*O(n)=O(n^2)**.
After a few runs of a huge input file, we realized that O(n^2) can be quite long. We attempted optimization by using allocations for all of the arrays involved with the algorithm. Originally, we used a global variable and made each array 50 blocks or 50 x 50 blocks. This took way too much time because most of the time, the algorithm was just accessing blank blocks. Changing the arrays to being malloc'd, helped runtime significantly. It is possible to optimize even more by using a heap implementation instead of an array implementation. This time complexity would be **O(n * ln(n)).**
We would try to accomplish that task if we were not constrained by time. Another optimization that we could have done that we tried to implement was for the 2 for loops in dijkstra function that search for the start and end index, we could have had some kind of code that exited the for loop when the indexes were found so rather than run through "n" times it would run through "1 to n" times. We attempted this but got too many errors so if we had more time to debug, that would have improved time complexity.

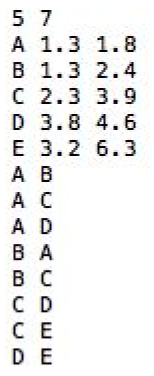
## Results and Testing Procedure

First testing was simple...

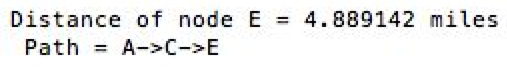
Input:

```
./final testnodes.txt AE.txt
```

Testnodes.txt: Output:



```
5 7
A 1.3 1.8
B 1.3 2.4
C 2.3 3.9
D 3.8 4.6
E 3.2 6.3
A B
A C
A D
B A
B C
C D
C E
D E
```



```
Distance of node E = 4.889142 miles
 Path = A->C->E
```

Then we used a file that generated random coordinates and edge connections with fixed vertices...

```
./buildfile
./final output.txt output2.txt
```

This was mainly to make sure there were no seg faults, valgrind errors, etc.

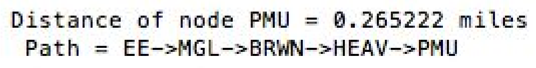
(Note: used stackoverflow for generating random numbers in buildfile.c, citation in paper and in code) (Generating Random numbers using SRAND, stackoverflow, 4th citation)

Then we made and tested Purdue's campus map as an input with buildings as nodes...

Input:

```
./final Purdue_Map.txt EE-PMU.txt
```

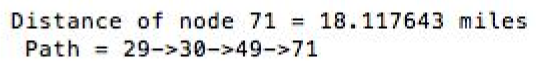
Output:



```
Distance of node PMU = 0.265222 miles
 Path = EE->MGL->BRWN->HEAV->PMU
```

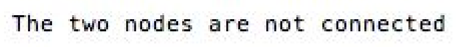
Then we were ready to test with usa.txt file supplied to us…

```
./final usa.txt 29-71.txt
```



```
Distance of node 71 = 18.117643 miles
 Path = 29->30->49->71
```

```
./final usa.txt 0-1234.txt
```



```
The two nodes are not connected
```

## Work Distribution

Yi Yang -

- Helped finding algorithm example
- Debugged code with Zane Johnson (mainly dijsktra and findEdgeDistances functions)
- Made Purdue_Map.txt
- Visualization of the test example
- Wrote project report

Zane Johnson -

- Created 2-D Array Implementation of Dijsktra's shortest path algorithm function

  *Note: Some code is based off of thecrazyprogrammer article noted in citations and noted in code header*
- Created two recursively freeing functions for graph and edges
- Helped create findEdgeDistances
- Debugged code with Yi Yang (mainly dijsktra and findEdgeDistances functions)

Mark Brooks - Came up with with a good way to implement VertexTag and EdgeTag structures (using linked lists for edges and next in VertexTag and linkedlist for connectsTo in EdgeTag)

- Debugged code errors like Segfaults in addEdge, AddVertex, and Djistrkas algorithm using valgrind and gdb.
- Made test file generator
- Worked on freeing memory and using recursion to help with time complexity

Andrew Grossman -

- Debugged coding errors in addEdge, AddVertex, and Djistrkas algorithm using valgrind and gdb.
- Assisted in creation of test file generator.
- Created functions to parse given graph input.
- Formatted code to a common standard.

## Citations

Dijkstra Algorithm for Finding Shortest Path of a Graph - The Crazy Programmer. (2014). Retrieved March 30, 2016, from http://www.thecrazyprogrammer.com/2014/03/dijkstra-algorithm-for-finding-shortest-path-of-a-graph.html
Much of second half of dijkstra's function was done using help from this article.

Dijkstra Shortest Path. (n.d.). Retrieved April 24, 2016, from https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html
Visualization of how the algorithm works. Mostly used for understanding before we actually started.

Dijkstra's Algorithm. (n.d.). Retrieved April 24, 2016, from
https://en.wikipedia.org/wiki/Dijkstra's_algorithm#cite_note-mehlhorn-3
Was used to find an easy to read explanation of the algorithm.

Generating random numbers using srand. (n.d.). Retrieved April 25, 2016, from
http://stackoverflow.com/questions/33182924/generating-random-numbers-using-srand