Dijkstra’s Algorithm in the Real World (SL Building)

Max Comer & David Nguyen

Project Summary

- What did we do?
    - Created a web application for indoor navigation of the SL Building
- Why did we do it?
    - To expand on class concepts by using a covered programming language in implementation
    - To create a practical application that has real world use
- How did we do it?
    - Written in JavaScript
    - Applying graph theory

Learning More About JavaScript

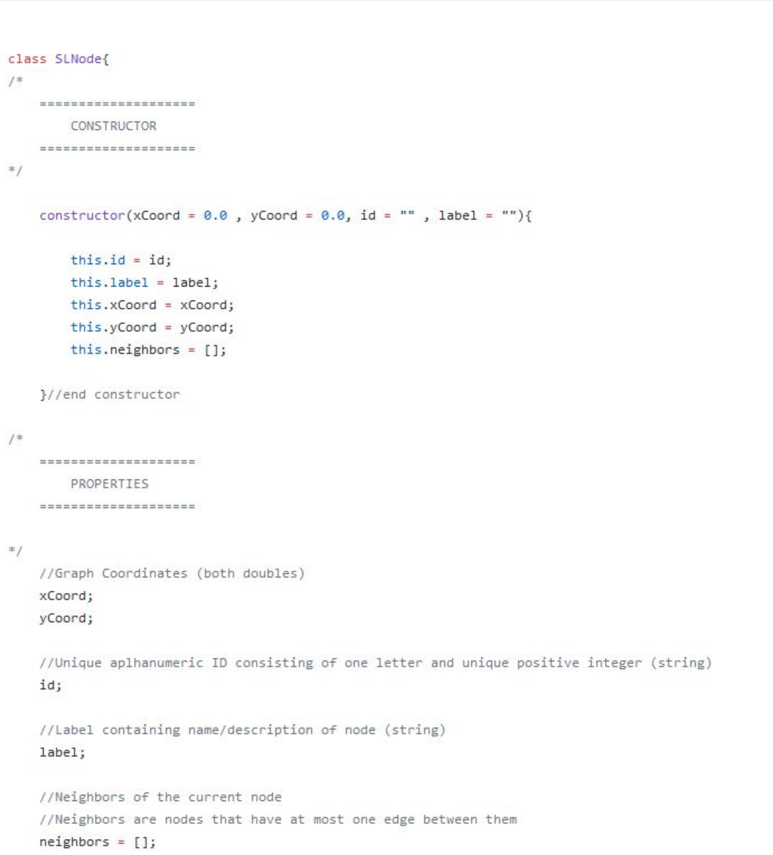
JavaScript Classes

Not what they look like...

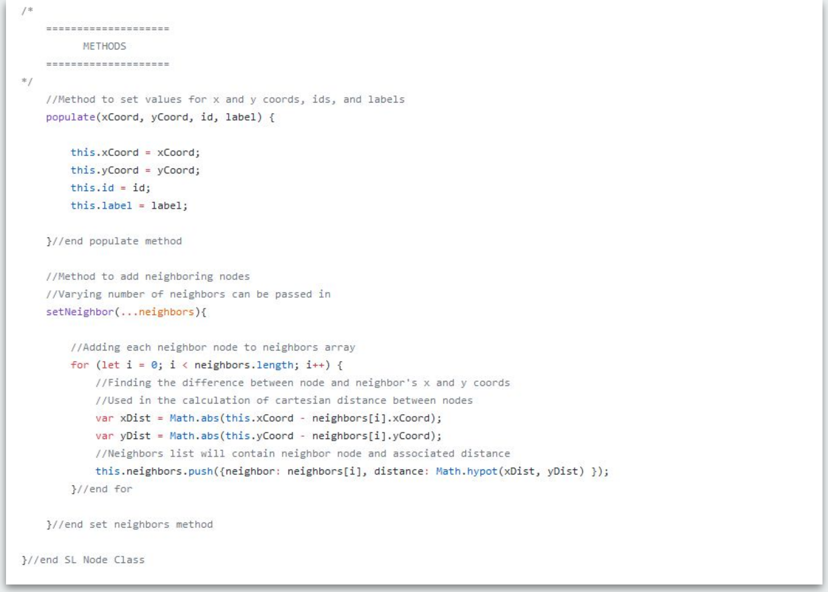
Syntactic Sugar

Class Syntax/Structure

Object Composition



```
class SLNode{
/*
    ====================
        CONSTRUCTOR
    ====================
*/

    constructor(xCoord = 0.0 , yCoord = 0.0, id = "" , label = ""){

        this.id = id;
        this.label = label;
        this.xCoord = xCoord;
        this.yCoord = yCoord;
        this.neighbors = [];

    }//end constructor

/*
    ====================
        PROPERTIES
    ====================

*/
    //Graph Coordinates (both doubles)
    xCoord;
    yCoord;

    //Unique aplhanumeric ID consisting of one letter and unique positive integer (string)
    id;

    //Label containing name/description of node (string)
    label;

    //Neighbors of the current node
    //Neighbors are nodes that have at most one edge between them
    neighbors = [];
```



```
/*
    ====================
          METHODS
    ====================
*/
    //Method to set values for x and y coords, ids, and labels
    populate(xCoord, yCoord, id, label) {

        this.xCoord = xCoord;
        this.yCoord = yCoord;
        this.id = id;
        this.label = label;

    }//end populate method

    //Method to add neighboring nodes
    //Varying number of neighbors can be passed in
    setNeighbor(...neighbors){

        //Adding each neighbor node to neighbors array
        for (let i = 0; i < neighbors.length; i++) {
            //Finding the difference between node and neighbor's x and y coords
            //Used in the calculation of cartesian distance between nodes
            var xDist = Math.abs(this.xCoord - neighbors[i].xCoord);
            var yDist = Math.abs(this.yCoord - neighbors[i].yCoord);
            //Neighbors list will contain neighbor node and associated distance
            this.neighbors.push({neighbor: neighbors[i], distance: Math.hypot(xDist, yDist) });
        }//end for

    }//end set neighbors method

}//end SL Node Class
```

SLNode Class

JavaScript Outside of the Web

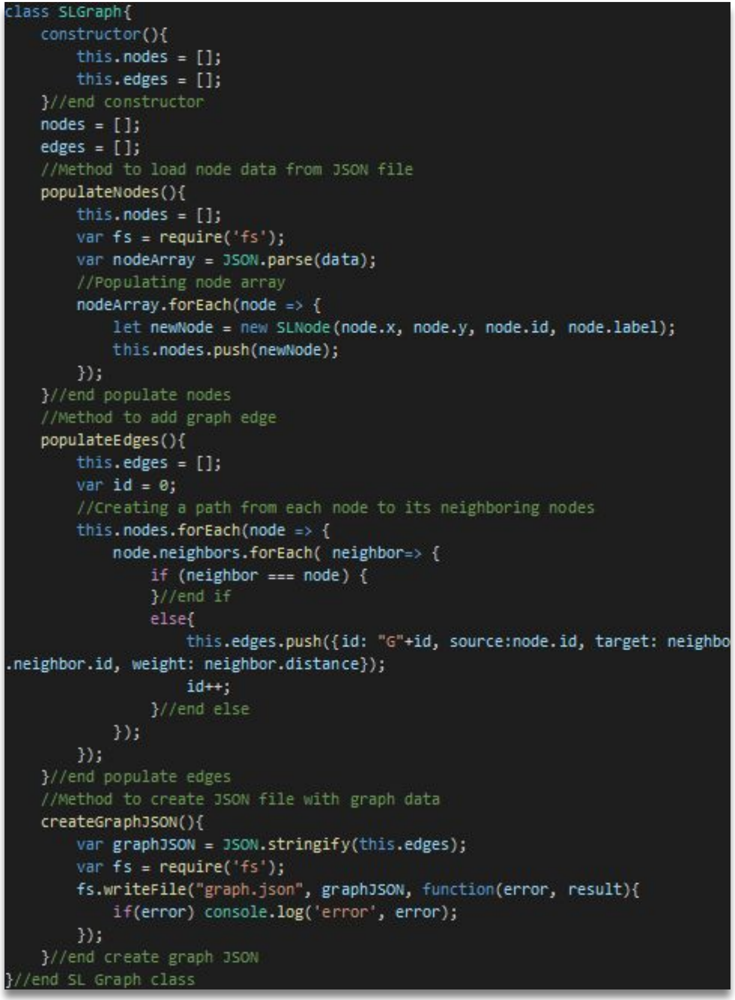
Developing with Node.js

“Vanilla” JavaScript

Node.js Environment

Local File I/O

Implementing Data Structures in JavaScript

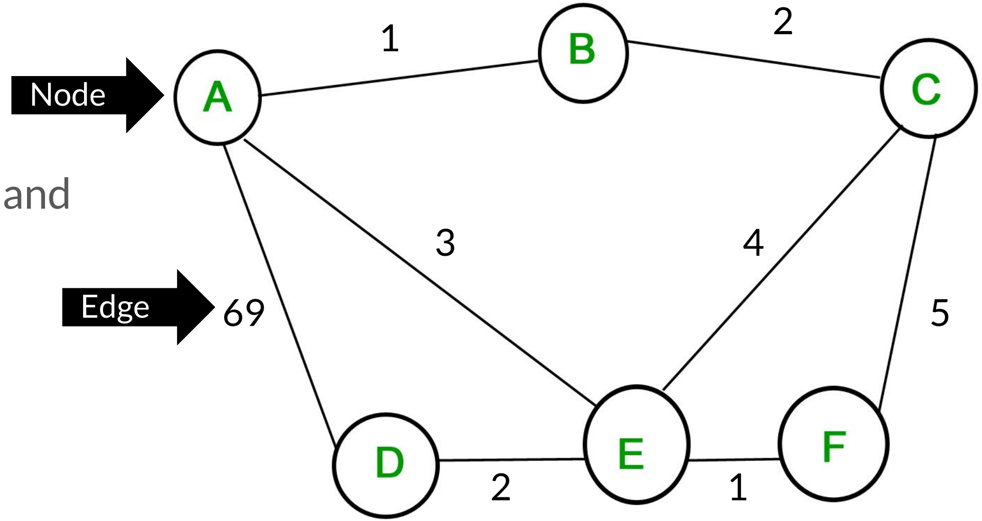


```
class SLGraph{
    constructor(){
        this.nodes = [];
        this.edges = [];
    }//end constructor
    nodes = [];
    edges = [];
    //Method to load node data from JSON file
    populateNodes(){
        this.nodes = [];
        var fs = require('fs');
        var nodeArray = JSON.parse(data);
        //Populating node array
        nodeArray.forEach(node => {
            let newNode = new SLNode(node.x, node.y, node.id, node.label);
            this.nodes.push(newNode);
        });
    }//end populate nodes
    //Method to add graph edge
    populateEdges(){
        this.edges = [];
        var id = 0;
        //Creating a path from each node to its neighboring nodes
        this.nodes.forEach(node => {
            node.neighbors.forEach( neighbor=> {
                if (neighbor === node) {
                }//end if
                else{
                    this.edges.push({id: "G"+id, source:node.id, target: neighbo
.neighbor.id, weight: neighbor.distance});
                    id++;
                }//end else
            });
        });
    }//end populate edges
    //Method to create JSON file with graph data
    createGraphJSON(){
        var graphJSON = JSON.stringify(this.edges);
        var fs = require('fs');
        fs.writeFile("graph.json", graphJSON, function(error, result){
            if(error) console.log('error', error);
        });
    }//end create graph JSON
}//end SL Graph class
```

Dijkstra’s Explanation for the Uninitiated

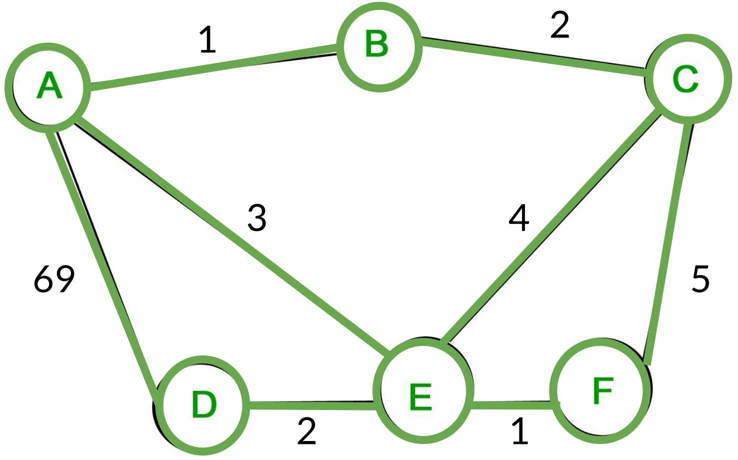
What’s a graph?

- A graph is a collection of **nodes** and **edges**.
  - Essentially a “map”.
- **Nodes**: destinations on a map
- **Edges/Vertices**: paths to said destination
- **Weight**: length of a given path



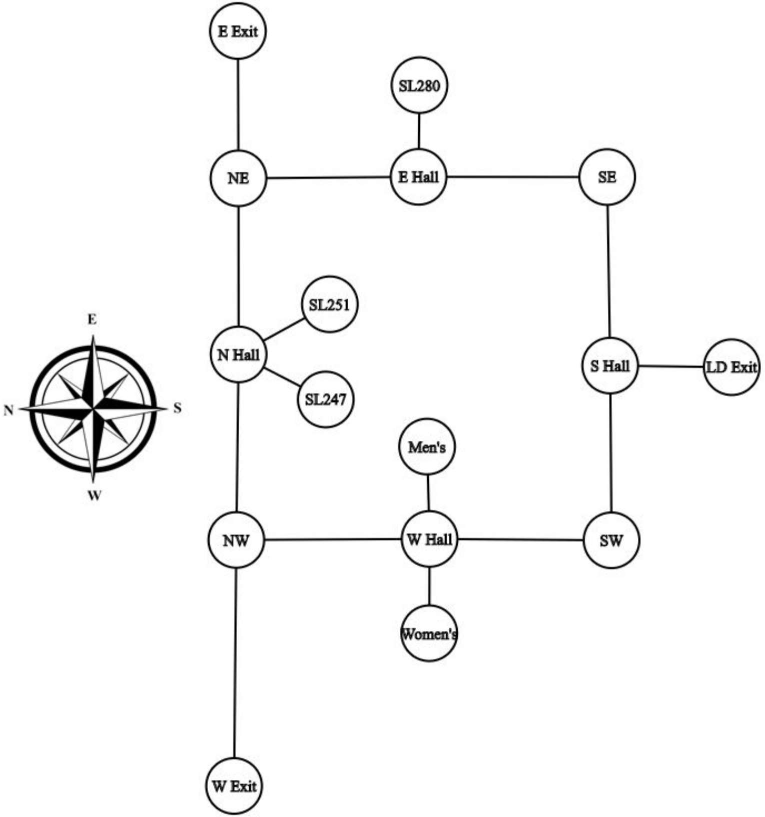
How Dijkstra's algorithm works:

Node	Shortest Distance	Path
A	0	-
B	1	A → B
C	3	A → B → C
D	5	A → E → D
E	3	A → E
F	4	A → E → F



Visited: A B C E F D

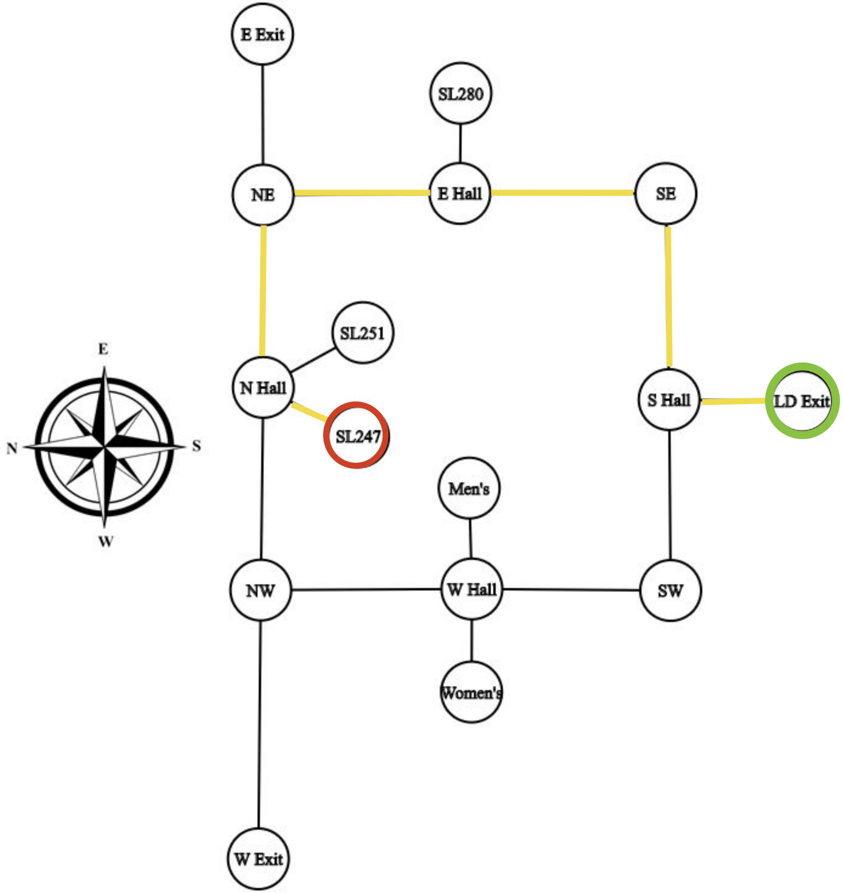
Done!



Simplified recreation of the SL Building 2nd Floor layout in a graph format.

Applying Dijkstra's to SL

Given user-selected starting and end nodes, **find** the **shortest path** between said nodes and **produce visual output** on graph.



Conclusion

Challenges and Difficulties

- Gathering data
- Making the map
- Keeping track of shortest paths
- Creating the map interface