ECE 368
FINAL PROJECT REPORT
TEAM BOILER RUSH

CHONGJIN CHUA
Kuo Tian
Zhengdong Qian
Junyan Shi
Yixuan Ding

BOILER RUSH OVERVIEW

Boiler Rush is a software that allows Purdue students to find the shortest path between two locations. Students often find themselves rushing from one place to another, and how they wish that they knew the shortest path to their destination. We want to provide a solution to this phenomena. Boiler Rush implements data structure theories to calculate shortest distance between two location, at the same time provides a friendly user interface.

This software could be categorized into three main parts: A Purdue map that is transformed into a 2-dimensional array consisting only 0's and 1's; A Python code that contains Dijkstra's algorithm; A Python graphical user interface that displays shortest path calculated.

For the 2-dimensional array, 0's indicates invalid location(i.e. A building or any non-road structure), and thus cannot be included as a node in the graph. 1's on the other hand indicates a valid location(i.e. Part of a road), and thus is included as a node in the graph. The python code will then read from this array, making only 1's as nodes and recording down the neighbors of each node(which are the 8 surrounding elements of any elements in the 2-dimensional array). After having these information, Dijkstra's algorithm is executed and the shorted path is obtained. The shortest path information is in the form of a 'list' containing tuples of coordinates, which is passed to the Python GUI to be plotted on the actual Purdue map.

IMPLEMENTATION

- DATA STRUCTURES/ALGORITHMS USED

The Dijkstra's algorithm implemented in this software utilizes adjacency list instead of adjacency matrix, because space consumption is one of our concerns when building this software. In order to implement Dijkstra's using adjacency list, programming concepts such as classes, methods and binary heap are fully utilized. Each 1's taken from the 2-dimensional array is stored as an object of Class Vertex, which contains the x,y-coordinates of the elements, the x,y-coordinates of neighbor elements(only 1's) of a particular element, and methods that assist in adding/retrieving information from the Vertex Class. These objects are then added to another object of Class Graph, which Dijkstra's algorithm executes upon. Since an adjacency list is used, a binary Minheap is also implemented in the code by importing the 'heapq' library.

- SETTING UP 2-DIMENSIONAL BINARY ARRAY

First, we use photoshop processing the BMP map to make the road in graph into white and building into black. Then we write a C program to read the BMP file and output a 2-Dimensional binay array with 1's and 0's. If the pixel is black the same place in the array will be 0 and if the pixel is white the number in the array will be 0. TheSince the thinest road has 6 pixels width, we reduce the map height and width to ⅙ of the original one. We get a digital map that has 1/36 number of pixels of the BMP map to ensure the Dijkstra’s algorithm has less vertexes to run through the whole map.

- GUI Design and User Interaction

The base UI layout and structure is designed under QtDesigner 4.0. To better adapt our needs to reach our goal, our project is constructed under the guidance of object oriented programming, hence class inheritance and overloading from basic Qt UI element are necessary. When user interacts with the UI, simply select source and destination on the drop down box and click run, then the result will be displayed on UI. A user can also saves the result as an image if one wants to. The interactions are done with the power of event handlers. The basic UI elements in Qt have most of the events such as click event, mouseover defined, so only function overloading is required. String, tuple dictionary is used for storing building location information and display geometry. The path drawing is done using QPainter and QGraphicsScene object in Qt Library.

COMPLEXITY ANALYSIS/OPTIMIZATION

The Dijkstra’s algorithm utilizes adjacency list, therefore the time complexity is $E * log(V)$, where E is number of edges, and V is number of nodes.

One of the toughest challenges in creating this software is mastering the Dijkstra's algorithm before Graph theory is being introduced in class. Because of the large amount of data (97*199 data points) needed to be processed by the algorithm, the downside of this software is that it takes around 20 seconds to calculate any shortest path. Given more time, instead of implementing Dijkstra’s algorithm, the A* algorithm would be our choice of implementation as the latter exhibits a better runtime.

RESULTS AND TESTING PROCEDURES

Tests are carried out by picking a source and destination that are both ‘1’ on the 2-D binary array, run the algorithm and obtain the result, and plot the shortest path result on the GUI. Below is a snapshot of the result:

Appendix

WORK DISTRIBUTION

ChongJin Chua	In charge of the backend of the program. The backend of the program consists of reading nodes from the 2-D binary array and adding them to a graph, the Dijkstra's algorithm, and storing/printing the shortest path result.
Kuo Tian	GUI construction, result displaying, integrating code from everyone and user interaction logics. Finalize and debug potential issues.
Zhengdong Qian	Using photoshop to process the map and coding the C program for transfering BMP file to 2-D binary array.
Junyan Shi	Found the pairs of the locations of the buildings on the maps and binary graph. Checked the binary graph to fix the unconnectness of the roads.
Yixuan Ding	Helping design the C program for transfering BMP file to 2-D binary array.

CITATION

https://en.wikipedia.org/wiki/Dijkstra's_algorithm
http://pyqt.sourceforge.net/Docs/PyQt4/classes.html