Traveling Wendy User Manual

Xinhui Xu and Julia McDonald

What is Traveling Wendy?

Traveling Wendy was created to help Wellesley College students find the shortest path around the Wellesley College campus. It includes an about page, where you can find instructions for use of Traveling Wendy and a picture of its creators, and a map page, where you can find the shortest paths!

How do I use Traveling Wendy?

About tab:

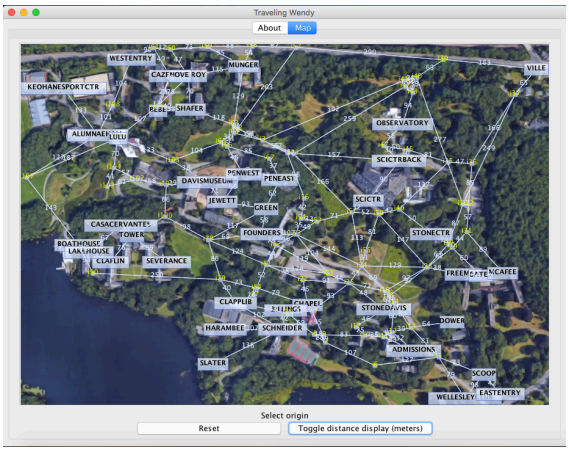
Upon running the program, you will arrive on the About page. Here you can read short instructions for use and see a picture of Xinhui and Julia. By clicking on the "Map" tab on top of the page, you can reach the map section of Traveling Wendy. You can return to "About" by again pressing the "About" tab.

Map tab:

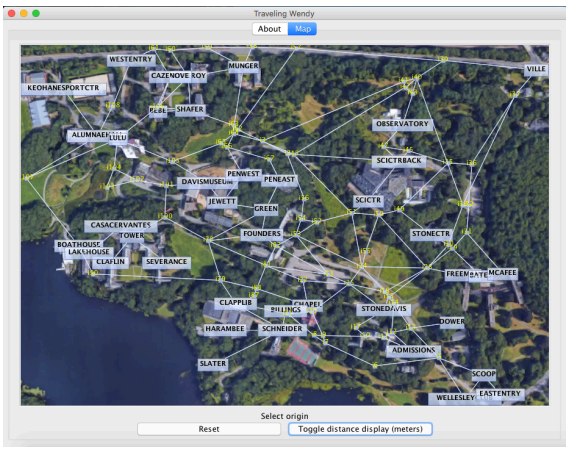
Our map has two buttons and numerous clickable nodes that represent buildings on Wellesley's campus. It has a graph of campus overlaid on an overhead image of the campus.

Toggle button:

The distances between buildings (marked in boxes) and intersections (marked in yellow) are shown by default in meters. To turn this feature off, click the "Toggle distance display (meters)" button.



(distances on)



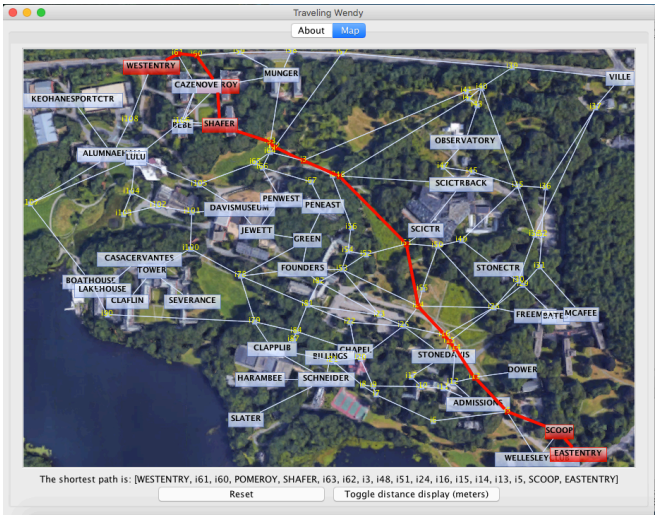
(distances off)

Finding a shortest path:

The shortest path between two buildings can be found by clicking on the name of the building that will start your path and then clicking on the name of the building that will end your path. Some building names overlap, but you can still click on them. Your shortest path will appear highlighted in red. It will also appear on the bottom of the screen as a series of intersections and buildings that you must pass through to get to your destination.

Reset button:

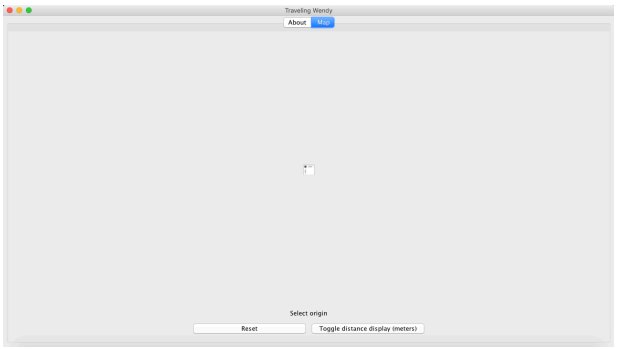
If you selected the wrong building or you found a path but would like to find another, use the reset button! It will reset the map to its original state so that you can find a new path.



(The shortest path from EASTENTRY to WESTENTRY)

What if I have a low-resolution screen?

Sometimes our map is too large for lower resolution screens. If this is the case for your screen, you may see this:



When you run the program, you can indicate the size of the map in pixels that you would like to use as a command line argument:

```
> run TravelingWendyGUI 830 550
```

The default measurements are 825 pixels X 570 pixels. Please do not increase the height beyond about 600 or the width beyond about 830, as it will cause the map to be misaligned with the image behind it (you will see a white border around the image). Most commonly, the height (the second argument) is what needs to be adjusted.

Traveling Wendy Technical Report

Xinhui Xu and Julia McDonald

Abstract Data Types:

i. Weighted Graph: Used to represent the map of Wellesley's campus. Nodes are buildings or intersections and Edges are paths or roads. Edges have their own class that constructs an Edge with a length and two Node endpoints. This was needed because in our implementation of Dijkstra's algorithm, we must know the distance between two points in order to calculate the shortest path. We could have stored all lengths between two points in another data structure within the WendyGraph class, but we felt that adding the length to an Edge made it easier to access and then
ii. Priority Queue: Used in Dijkstra's algorithm to store Nodes while the algorithm is finding paths between Nodes in the graph. The Node with the lowest distance must be dequeued first, which is why we used a Priority Queue instead of a regular Queue.
    a. NOTE: In my presentation, I said that we used a Hash Map and a Hash Set, but this was incorrect. We had tried it in a previous implementation but it did not work well, so we implemented Dijkstra's algorithm using a Priority Queue.
iii. LinkedList: Used in the adjacency list implementation of WendyGraph to store Edges for each Node. Java's doubly-linked list implementation was used. This data structure was of use because it is faster than an ArrayList (for example) when adding Edges even though it is slower when accessing Edges. This means that in a graph implementation, it is better to use a LinkedList because most often, we are adding Edges more often than we access them.

Class Descriptions (Main methods listed for each class):

i. `Node.java`
   Creates nodes for use in a graph that represent buildings and intersections on the Wellesley campus. Each holds a variable isBuilding that indicates whether a given node is a building. It also holds a name, latitude coordinate, longitude coordinate, and weight and previous node in shortest path (for use in Dijkstra's algorithm in WendyGraph.java).
    a. `compareTo(Node n)` – The Node class implements Comparable so that first Node in the Priority Queue (used in Dijkstra's algorithm in WendyGraph) to be dequeued will have the lowest weight.
ii. `Edge.java`
   Creates edges for use in a graph that represent roads and paths on campus between buildings or intersections (represented by two nodes). Each edge holds a length variable, which is calculated based on the latitude and longitude coordinates of the two nodes. A separate Edge class was needed because we are using a weighted graph, and therefore must store length in Edge.
    a. `getGreatCircleDistance(double lat1, double lon1, double lat2, double lon2)` – Returns the distance in meters between two latitude/longitude coordinate. For use in displaying distances on map AND in setting the length of an Edge, which is used in adjusting the weights of Nodes within Dijkstra's algorithm.
iii. `WendyGraph.java`
   Creates an adjacency list of Edges for each Node by reading from a text file in order to create a graph of Wellesley College. Runs Dijkstra's algorithm.
    a. `WendyGraph(String filename)` – Reads from a text file and constructs an adjacency list graph of all Nodes (Buildings/Intersections) and Edges (Roads/Paths) on the campus.

b. `getPixelCoordinates(double lat, double lon, int mapWidth, int mapHeight)` – Calculates the coordinates of the pixel where the Node is to be placed in the GUI.
c. `dijkstra(String sourceName)` and `dijkstra(final PriorityQueue<Node> q)` – These two methods are responsible for finding the shortest path from a source Node to all other Nodes in the graph. They use a Priority Queue to implement Dijkstra's algorithm.
d. `getPath(String endName)` – This method finds the path of Nodes from the source Node to a given end Node. This is useful in the TravelingWendyPanel for highlighting the path from the clicked start node to the clicked end node.

iv. `TravelingWendyPanel.java` – Represents a WendyGraph as a GUI. This class uses jgraphx (an outside library) to make modelling the graph significantly easier. Constructs a panel to be added to TabbedPanePanel and ultimately TravelingWendyGUI.
a. `TravelingWendyPanel(int graphViewportWidth, int graphViewportHeight)` - Creates graph and graph stylesheets, plots all vertices, draws all edges, adds the graph component to the panel, creates a click handler, displays labels and button, sets background, and creates button handler inner class

v. `TabbedPanePanel.java` – Creates two tabbed panes: one from an About panel, which is constructed in this class, and one from a TravelingWendyPanel, constructed in the TravelingWendyPanel class.

vi. `TravelingWendyGUI.java` – Main runner for the project. Adds an instance of TabbedPanePanel to a JFrame to deliver the final GUI.