Nets 150 Final Project: Real Time Septa

Rajat Bhageria
Matt Chan
Colin Roberts

Overview

Here are the various steps and concepts we utilized in the creation of our project:

- We created a data scraper to turn the SEPTA map into actionable data, by scraping the station list from each of the SEPTA schedules.
- After that, we used a map reduce algorithm to take that data and create an adjacency list of the SEPTA map graph.
- Using the adjacency list of the SEPTA map, we created a shortest distance route calculator using breadth first search.
- We used the SEPTA API, Google Maps API, created KML XML files of SEPTA tracks, and created a real time map of the location of all the SEPTA trains.
- We used location data, the SEPTA API, and a JSONp scraper to tell the user when the next train would arrive at the station they were at.
- Taking the SEPTA map graph adjacency list, we also used the page rank algorithm to see which station(s) were the most influential in the system.

Project Relevancy

We went further than the project specifications, and used three elements from the list:

- Graph and graph algorithms – the SEPTA rail map is a very good example of both a directed and undirected graph, that has very relevant use cases. We used the concepts of an adjacency list, breadth first search, and PageRank in our application of the SEPTA rail map.
- Document Search – Integrating the SEPTA map into an app may seem easy at first, but unfortunately SEPTA's API isn't the best. We had to write a scraper and a map reduce algorithm to create the graph representation of the SEPTA rail map. Even trying to return the time until the next train required information retrieval techniques.
- Physical Networks – In the process of using SEPTA APIs, we had to learn how to use http GET requests, as well as how to get around the limitation of GET requests working only in the same domain (Same Origin Policy). This required a workaround using JSONp formatting that took a while to figure out, and helped us understand the physical network of the Internet better.

How to Install the App

In order to see a few of the features, including the breadth first search, the real time map, and the next train feature, one must run the app.

We've made things really easy by deploying on Heroku as a web app. Simply point your phone's browser towards https://realtimesepta.herokuapp.com. Allow the use of your location data, or the app features become pointless.

Data Scraper

In order to turn the above SEPTA map into actionable data, we wrote a scraper using JSoup to scrape data from individual SEPTA schedules to create a text list of stations for every single one of SEPTA's 13 rail lines. You can see the list of stations for each of the line in the excel file raillines.xlsx, and an integrated text version in raillinestext.txt. The code for the scraper that we used was simple, and we combined the schedules on the excel file.

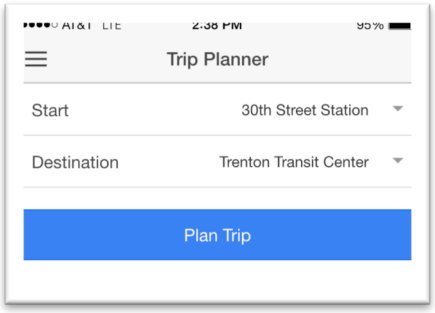
```
Document doc = Jsoup.connect("http://www.septa.org/schedules/rail/w/AIR_1.html").get();

// Get links on page
Elements links = doc.select("a[href]");

for (Element link : links) {
        // Detect to see if this is a station as a bold hyperlink
        if (link.parent().toString().contains("<b>")) {
                System.out.println(link.ownText());
        }
}
```

MapReduce

Following that, we fed the text file into a MapReduce algorithm that we wrote in Java to turn all the individual 13 lists of stations into an adjacency list representing a graph of the entire system. The map part of the algorithm put all the nodes (stations) into a key value store (hashmap), with the keys being the nodes and the values being the stations that node is connected to. This would also check for

duplicates between rail line lists in order to integrate the 13 lists into one cohesive graph. The reduce part of the algorithm takes the key value store and outputs a JSON formatted adjacency list. You can find this in the GraphTools.java file. The outputted JSON formatted adjacency list can be found in the file StationJSON.txt.

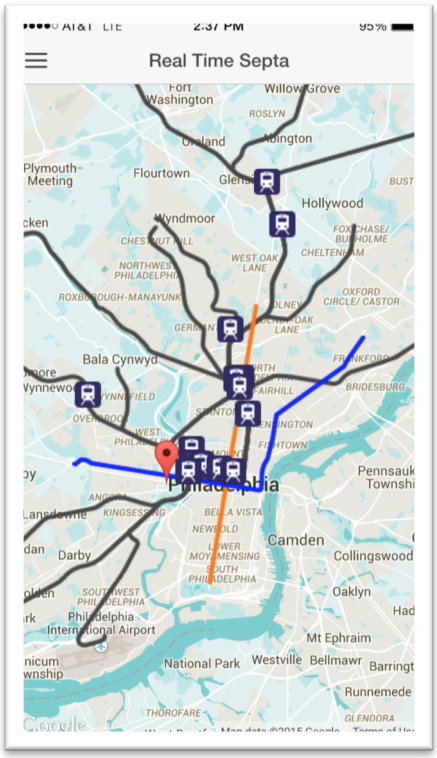
Breadth First Search (App Feature: Trip Planner)



In order to find the most efficient path between two nodes on the SEPTA graph, we implemented a Breadth First Search algorithm using the JSON formatted adjacency list. The algorithm uses a queue and looks through the graph on a step-by-step basis until it finds the station that one is trying to go to. After that, it lists the stations that the user should go through to get to the correct station. This is integrated into the app as the Trip Planner tab. The source code can be found in controllers.js under realtimesepta/www/js.

The use case for this is extensive. If someone wants to figure out the shortest way to get from one part of Philly to another, this feature will be extremely helpful in navigating the somewhat confusing SEPTA map diagram.

Real Time Map (App Feature: Home Screen)

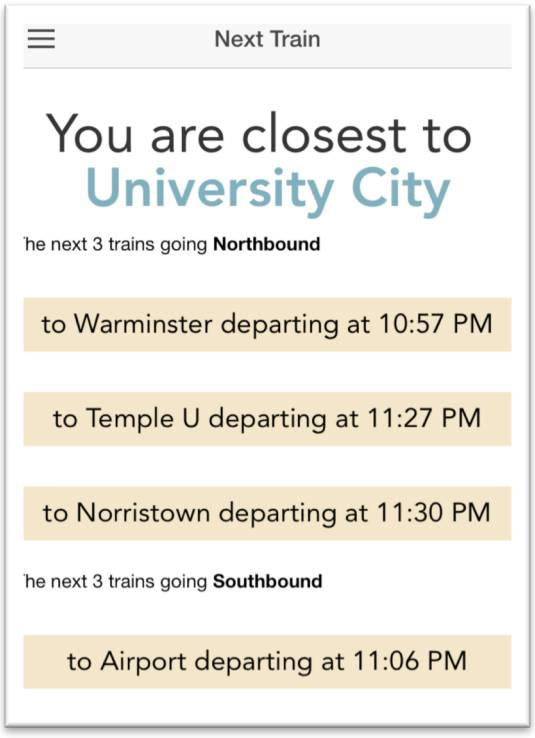


The Real Time Map is an amalgamation of various API tools in order to show a dynamic graph in action. First, we used the SEPTA API to get the GPS locations of all the trains in real time with a JSONp GET request. Secondly, we plotted the locations of all the trains onto a map using the Google Maps API. After that, we created and imported KML files to draw the lines on the screen representing the SEPTA tracks. KML files are in XML format, and are essentially lists of coordinates that connect together to draw a line. We also customized the look and feel of the map with a theme, KML track colors, and little train icons.

Finally, we used JavaScript to refresh the locations of the trains every 10 seconds, to keep the train locations up to date. This was accomplished via an array that deleted all the train icons and created new ones with new coordinates.

You can see this in action as the Real Time Map part of the app, which appears when you start the app. The source code can be found in controllers.js under realtimesepta/www/js.

Next Train (App Feature: Next Train)



The Next Train feature uses location services, the SEPTA API, and a JSON scraper to return to the user when the next trains will come based on what station they are at. The SEPTA API returns a list of trains passing through a specific station in JSON format, with a lot of irrelevant information. We first get the user's location and calculate the nearest station based on geographic distance, and then feed that station into the SEPTA API and parse through the useless information to get the relevant data. All the user sees is the next northbound and next southbound train, along with what time the train will be there. The use case for this app is simple: you're waiting at any one of SEPTA's numerous stations, and want to know how much longer you'll be waiting.

This feature is in the Next Train tab and the source code can be found in controllers.js under realtimesepta/www/js.

Page Rank Algorithm

We thought that a very interesting analysis we could do with the SEPTA rail graph would be to implement the Page Rank algorithm. This would return the most influential stations in the SEPTA map, which could be a good indicator of the stations that SEPTA should invest more money towards.

Thus we implemented the Page Rank algorithm in its entirety, complete with a damping factor of 0.85, a loop that could adjust to an n number of iterations, and a data structure to keep track of the calculated Page Rank values for each station between iterations. It was interesting to see that 30th Street Station was not the most influential station from a pure graph analysis standpoint, but rather Wayne Junction. Looking at the SEPTA map, however, it does seem that Wayne Junction is definitely more central than 30th street station, so it makes sense.

The most influential stations according to Page Rank:

1. Wayne Junction
2. 30th Street Station
3. University City

The code can be found in GraphTools.java.