Angel Routes.

Author: Ashot Ordukhanyan (orduka@gmail.com)
Published Date: 09/29/2024

Abstract

Angel Routes is a toy application created for submission to the Alphathon 2024 competition, organized by the Society of Quantitative Analysts (SQA).

More details about Alphathon 2024 can be found here.

Angel Routes is a proof-of-concept application designed to illustrate potential uses of an open source high-performance reactive stream processing library called csp.

This document, as well as all of the source code associated with the application is made publically available under the terms of Apache 2.0 license in this github repository.

Introduction

In recent years, bike-sharing apps have surged in popularity across the globe, offering convenient, eco-friendly transportation alternatives in urban areas. As of 2024, there are over 2,000 bike-sharing programs worldwide, with millions of shared bicycles in operation across major cities. In the United States, New York City boasts one of the largest bike-sharing programs—CitiBike, which is administered by Lyft. CitiBike provides access to over 27,000 bicycles across the city, making it an integral part of New York's public transportation system. With the growing demand for shared mobility solutions, optimizing bike availability has become critical to ensuring efficient, reliable service for all users.

One of the main challenges in bike-sharing systems is maintaining a balanced distribution of bikes across docking stations. Programs like CitiBike's "Bike Angels" incentivize users to help redistribute bikes, moving them from full stations to emptier ones. Bike Angels encourages riders to take short trips that improve bike availability. Users earn points[1] for completing rides between targeted stations, which can be exchanged for rewards. This approach reduces the need for manual redistribution, which is typically costly and resource-intensive.

In a city where everything is a competition, CitiBike's Bike Angels program has quickly turned into a fierce battleground. Forget helping the community; New Yorkers are all about climbing the leaderboard and racking up points like it's the Tour de France. Reddit forums buzz with strategy tips on the best stations, routes, and times to maximize points, while riders form rivalries and alliances in their quest to dominate the rankings. It is no longer just about getting from A to B, it is about winning the city's most competitive (and eco-friendly) game. The competition and hustle of bike angels has even been a subject of several recent newspaper articles[2].

This project aims to use real-time information provided by CitiBike backend services to identify "Angel Routes" – short bike rides that a user can take that will move a bike from a (nearly) full to a (nearly) empty station, thereby improving the distribution of bikes across stations, helping out fellow commuters (and yes, potentially generating some Angel Points for participants).

Methodology

At the core of this project there is a CSP graph that processes real-time bike share information. Implementation of the graph can be seen angel.py.

To feed data into the graph, two real time adapters have been created:

- `GBFSStationInfoAdapter` – pulls real time (albeit rarely ticking) information about Citibike stations – their location, total capacity etc.

[1] The exact algorithm that CitiBike program uses to determine the number of points that would be awarded is not made public (probably for fears of users abusing the system). It's not the intention of this project to reverse-engineer the algorithm – rather I am simply trying to highlight potential short CitiBike trips that will improve the distribution of bikes across stations, in line with the spirit and intention of the Bike Angels program.

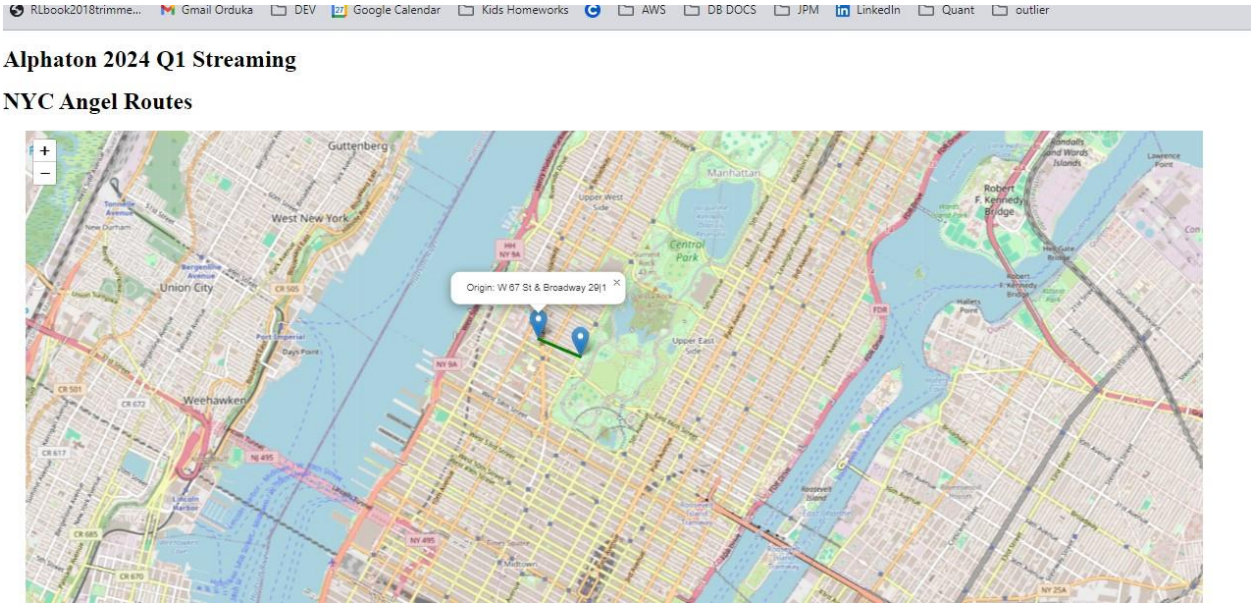
[2] See NY Times Article 1 and NY Times Article 2

- `GBFSStationStatusAdapter` – pulls real time status information about Citibike stations – number of available bikes, available docks etc.

The code for the adapters can be seen in gbfsadapter.py

Main graph processes events, calculates haversine distances between stations calculates real-time occupancy rates in every station (defined as number of available bikes/number of total docks) and highlights shortest routes from every (nearly) full station (defined as occupancy rate > 95%) to (nearly) empty (occupancy rate < 5%) station.

The routes are then displayed in a simple flask web app in a graphical form (superimposed an interactive map).



20 shortest rides that fit the criteria are displayed on a map at any given point in time, map updates on a 20 second cycle.

To facilitate efficient processing of potentially voluminous real-time data in our CSP graph, the following CSP concepts were utilized:

- Historical adapters. Pull REST data from CitiBike servers in an intelligent fashion (.e.g. keep track of timestamps to not generate unnecessary duplicate ticks in the graph).
- Dynamic baskets: NYC CitiBike has over 2000 stations, on any given update cycle only some of them will change their real-time status (i.e. number of available bikes/docks). Representing this information as a dynamic basket indexed by station id made it possible for nodes in the graph to include only stations that "ticked" but also have access to the most up to date state of all stations.

Acknowledgements:

Author would like to especially acknowledge the following open-source libraries that were used in this project.

- CSP. High performance reactive stream processing library

- Haversine. Library to calculate distance between two points on Earth's surface.
- gbfs-client. Library for discovering and capturing like bike-share data feeds from hundreds of global bike-share providers.
- folium. Library for generation and display of annotated maps.

Further Work.

This project could be extended further to analyze patterns that arise distribution of bikes across stations as a function of time of day, day of the week and other exogenous factors. Historical bike share ride data is made available by CitiBike via compressed data files (e.g. here). Using CSP with input adapters written to download and parse this data would ensure that the logic used in historical analysis is portable into a real-time context and that historical analysis does not suffer from look-ahead bias.