Broad DSP Engineering Interview Take-Home

Last updated: 2022-03-03

Described here is a coding exercise where you integrate with the Boston transportation system website https://mbta.com and use the data it provides to answer questions. We recommend you read through this document completely before starting. You may also want to register now to receive a key for your MBTA API client at https://api-v3.mbta.com/register You don't need an API key to experiment but having one means that the MBTA web server will be less likely to throttle your HTTP requests.

Please complete and email your solution to dsp-engineering-hiring@broadinstitute.org at least 24 hours before coming in for your interview. You should feel free to use any resources that you wish and deliver the solution in the language of your choice.

Despite being a toy problem, the purpose of this exercise is to gauge how you develop software as a whole, including design, documentation, and testing. We'll be paying attention to how you strike that balance. A clever one-line solution will make us smile but won't give us any insight into how you work on more complex projects, so we'd prefer it if you avoided them. The MBTA developer site recommends tools for generating client code from their documentation. We recommend you not use these tools or anything that wraps the MBTA API.

Instead, write your own code using HTTP client libraries (e.g. [requests (https://requests.readthedocs.io/en/master/)](https://requests.readthedocs.io/en/master/) for Python) to interact directly with the MBTA APIs. We also recommend that you pick a language that you are familiar with. It might be enticing to use this as an opportunity to learn a new language, but when we ask questions and code live during the interview you will want to have tools and code that you are comfortable talking about and using.

Feel free to use resources like Google and Stack Overflow to get answers to any questions you might have as you do your work.

How long should you spend on this? As much time as you want. We know you have other things going on in your life, so if you're not finished after a few hours, that's absolutely fine - submit as much as you have, and we can talk about that. We're interested in your approach to the problem, not how fast you code. In particular we'd rather see a clean solution that gets most of the way there and is well documented and tested versus a solution that correctly addresses every possible edge case. There are no trick questions involved here, we just want to see how you work.

If you have any questions along the way please do not hesitate to send them to dsp-engineering-hiring@broadinstitute.org.

Your deliverables should include:

- relevant source code,
- documentation on how to use your program to demonstrate all three of these functions,
- any relevant supplementary files such as tests or documentation.

When complete, share an accessible git repo or create a tarball or zip archive of your files and email it to dsp-engineering-hiring@broadinstitute.org.

Problem description

Boston's transportation system, the [MBTA (https://mbta.com/)](https://mbta.com/), has a website with APIs https://api-v3.mbta.com/docs/swagger/index.html.

You will not need an API key, but you might get rate-limited without one.

The MBTA's documentation https://api-v3.mbta.com/docs/swagger/index.html is written using OpenAPI/Swagger. If you haven't used Swagger before, this tutorial walks through the basics on an example project: https://idratherbewriting.com/learnapidoc/pubapis_swagger.html

The MBTA developer site recommends tools for generating code from their documentation. We advise you not to use these tools because they are heavyweight and unnecessary. Writing your own client code to interact directly with the APIs is most likely easier, and better demonstrates the skills we're interested in reviewing.

Question 1

Write a program that retrieves data representing all, what we'll call "subway" routes: "Light Rail" (type 0) and "Heavy Rail" (type 1). The program should list their "long names" on the console.

Partial example of long name output: Red Line, Blue Line, Orange Line...

There are two ways to filter results for subway-only routes. Think about the two options below and choose:

1. Download all results from https://api-v3.mbta.com/routes then filter locally
2. Rely on the server API (i.e., https://api-v3.mbta.com/routes?filter[type]=0,1) to filter before results are received

Please document your decision and your reasons for it.

Question 2

Extend your program so it displays the following additional information.

1. The name of the subway route with the most stops as well as a count of its stops.
2. The name of the subway route with the fewest stops as well as a count of its stops.
3. A list of the stops that connect two or more subway routes along with the relevant route names for each of those stops.

Question 3

Extend your program again such that the user can provide any two stops on the subway routes you listed for question 1.

List a rail route you could travel to get from one stop to the other. We will not evaluate your solution based upon the efficiency or cleverness of your route-finding solution. Pick a simple solution that answers the question. We will want you to understand and be able to explain how your algorithm performs.

Examples:

1. Davis to Kendall/MIT -> Red Line
2. Ashmont to Arlington -> Red Line, Green Line B

Hint: It might be tempting to hardcode things in your algorithm that are specific to the MBTA system, but we believe it will make things easier for you to generalize your solution so that it could work for different and/or larger subway systems.

How you handle input, represent train routes, and present output is your choice.