***GRAPH ADT DESCRIPTION***

There is an almost endless amount of choices for representing a graph with a data structure. You can use various combinations of arrays, types of lists, queues, maps etc. All have various advantages and disadvantages. This resulted in there being 2 guiding concepts on how to implement a graph in code, an adjacency matrix and an adjacency list. The matrix-style is normally used for a dense graph, where the number of edges approaches number of vertices squared. The simplest way this can be done is by using something along the likes of a 2D array. The matrix-style methods have super-fast indexing and ease of use, but the downside is that they have a ginormous memory cost, particularly for bigger graphs. An adjacency list is normally used for a sparse graph. These need more complicated index management and take more effort to maintain, however the space cost is massively reduced. As far as indexing speed goes, it depends on what implementation you go with.

The Boston metro system we are working with is a sparse graph. For this reason, we have gone with the adjacency list concept. For the implementation, we decided to use a map, specifically a Hashmap. It will have a key of a numerical station ID and a value of an array of edges connected to it. An edge will know what line they are on, what stations are connected at either end (both numerical ID and string name) and their own weight. A station will know its ID and what edges are connected to it. We will use that information to build our search algorithm for the graph, most likely using Dijkstra’s algorithm for its ability to traverse uninformed, weighted graphs. The search algorithm will return an array of edges that our frontend will use to display the route.

The are a few reasons we chose to use a Hashmap with a numerical station ID mapped to an array of edges. The Hashmap is an officially supported library implementation and is therefore likely to be very fast, efficient and reliable. This gives us more flexibility to implement our own search algorithm in a more efficient and reliable way. We will be using a standard array over an implementation like Arraylist, as we do not need one of its primary features, the ability to dynamically adjust its size. Therefore, whilst it has its benefits, using an Arraylist would mostly just add unnecessary overhead and complexity. Finally, we will be a numerical station ID as the key instead of its string name as it would be faster, easier to implement and increase maintainability. The final output shown to the user will pull the string name from the array of edges it is passed.