Optimisations

Before I decided on a HashMap for my *ADS2List* data, I had the idea to use a linked-list approach. This was for a number of reasons, not least because insertion (*addNode*) was of O(1) complexity. This led in most cases to broadly similar times between my Linked List and my HashMap, although I soon noticed the HashMap did run faster on some occasions, as seen below (1 is the time taken for HashMap’s *addData* 2 is the linked list’s *addNode*). However, on the terms of the *add* operation alone, there was little between *addNode* and *addData*, particularly when all the other operation times were taken into account.

A picture containing text

Description automatically generated

*A picture containing timeline

Description automatically generated*

Graphical user interface, application

Description automatically generated

However, when considering this optimisation it was not only the *add* methods that needed to be considered. A crucial part of the project is taking an ID and searching for a name in the *ADS2List* data, and it was here that the linked list approach began to suffer significant performance problems. This is because accessing items in a specific position is less efficient with a linked list, taking O(n) time compared to HashMap (O(1) when we know the key, which we do because it’s what we’re using to search with). As such, for this project’s more specific requirements it made sense to switch my approach to deliver better execution time, and this meant choosing to implement a HashMap instead. Of course, this also meant implementing new methods.

These new methods, the code for which can be seen below, delivered a change in approach towards their respective operations to reflect the new data structure being used. While most of the performance gains came from switching the data structure, I also implemented helper functions like OpenAddressing to further improve performance and avoid collisions. A further performance optimisation would’ve been to implement a binary search: however, this was unfortunately not possible in the time. Nevertheless, the eventual *SearchByID* method delivered good execution speed, as evidenced by profiling, matching its expected time complexity of O(n).

A picture containing diagram

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated