Cross-Language Algorithm Performance Analyzer

A time based comparator application for different algorithms

Raj Pastagiya (22M2111)

Chaitanya Shravan Borkar (22M0810)

1 Background

We have seen many algorithms in our study in Computer Science and we have also theoretically compared them based on time complexity and space complexity. But as a Computer Science student we may also want to know how exactly those comparisons work on a real life computer system. While theoretical understanding may give approximate answers to our concerns in real life we may or may not use the same theoretical outcomes as there are many factors involved especially in calculating execution time of the algorithm on a certain data set.

2 The Cross-Language Algorithm Performance Analyzer

2.1 Problem Statement

Given practical implementation using programming language of the algorithm we want to compare the execution time on different type of data sets.

2.2 Tools and Technologies

- ***GitHub** for software management and version control*
- ***Eclipse IDE** as project development environment*
- ***Java-11** with **Spring Framework** for creating back-end APIs and supporting back-end logic*
- ***MySQL** for persistent data storage*
- ***HTML, CSS, and JavaScript** for supporting front-end logic of the application*
- ***Latex** to document the project*

2.3 Application Level Overview

Given a set of algorithms as selection options on front-end, user can choose 2 algorithms to compare with each other. The Application then will run these algorithms on a diverse type of data sets and note the time taken to execute certain process, sort the data set if the algorithm is sorting based. The required data then is given to front-end where front-end then will represent these data in a graphical form to show how the trend goes with different type of data sets. Below are different aspects of the application that defines the implementation of the application.

- ***Logical Algorithm Pools:***
  - *The predefined algorithms will come with the application which will be divided in some logical pools and the comparison can only be done between the algorithms in a specific pool.*
  - *Below are the predefined pools and algorithms that come with the application;*

    1. ***Sorting Algorithms:*** *Selection Sort, Merge Sort, Bubble Sort*
    2. ***Searching Algorithms:*** *Linear Search, Binary Search*
    3. ***Knap-sack Algorithms:*** *Brute Force Approach, Greedy Approach, Dynamic Approach*

- ***Data-Set Pools:***
  - *User will also be given choice of data-sets, which could be one of the following;*
    1. *Random unique numbers of various taste: Positive, Negative, Whole Numbers, and Decimal Numbers*
    2. *Random repeated numbers of various taste: Positive, Negative, Whole Numbers, and Decimal Numbers*
    3. *Sorted unique numbers of various taste: Positive, Negative, Whole Numbers, and Decimal Numbers*
    4. *Sorted repeated numbers of various taste: Positive, Negative, Whole Numbers, and Decimal Numbers*
    5. *Special Knap-Sack datasets*

- ***User's Perspective:***
  - ***User View:*** *User will be presented with different types of algorithms as options predefined under a logical comparison pool.*
  - ***Algorithm and Data set Selection:*** *User can select 2 algorithms to compare to within a single pool. User can also select data set and then submit it as a query.*
  - ***Graphical View:*** *Once submitted the query, the user will be presented with a graph showing live comparison of time over different data sets.*

- ***Backend Implementations:***
  - ***Storage:*** *Backend will store the implementation of algorithms specified in above algorithm pools and some predefined data-sets (excluding random data-sets) in different files in an organized manner. The implementation of these algorithms will be done using the reference of implementations done on well known open-source websites like GeeksforGeeks*
  - ***Request processing:*** *Given a specific query containing selected algorithms and data-set backend will use multi-threading approach to calculate execution time for each of the algorithms requested.*
  - ***Data collection to create graph:*** *For live graph creation frontend will poll to an open API provided by the backend at every second to get live data containing data-set size and time taken for execution for specific algorithm.*
  - ***Validations over requests:*** *While a query request is already in process the API will through validation error with proper error message and do not allow new request to come until older request is completed.*
  - ***Cleanup on successful comparison query completion:*** *Once all datasets have been completed processed or a timeout occurs (which is predefined in the application) the threads will stop running and all the threads created will be cleaned up. The timeout policy is used so that the application don't go to infinite wait for a large set of data to be processed and thus enhancing the user experience.*
  - ***Custom Algorithm Support:*** *A user can upload an algorithm written in languages like JAVA, and Python via "/upload" API and for future comparisons user can then select uploaded algorithm with already available algorithms to see the execution trend on data sets. This is purely a backend API with algorithm file as POST request.*

2.4 What did we achieve?

As part of the project timeline we were able to completely achieve all the aspects of the features mentioned in the above section. A brief summary of the features that are achieved to look out for are as follows;

- *The data mapping corresponding predefined algorithms provided is populated on the application startup. The data is populated using in-memory Map data structure.*

- *Once the user starts opens the HTML web page the frontend application collects the data from backend using API opened by the backend(ideally "/getAlgorithmDetails") which will then be displayed in certain manner on the web page*
- *The user is prompted to select algorithms to be compared to and the datasets on which the algorithms are going to be compared. The datasets are predefined and stored on the backend in organized file structure. The user can submit the query and backend will process on that query.*
- *Before process the query backend will do validation checks including if any other query is already running, according the validation checks if check fails corresponding failure is handled via Java Exception handling.*
- *Once query is validated, the backend now created one thread per algorithm to do processing on datasets parallely and corresponding execution times are stored in ConcurrentHashMap as part of CACHE to be accessed in future. For shared data access application also uses locks to avoid race conditions.*
- *On the frontend the code will now start fetching data from an API opened by the backend at every second and plot the graph corresponding to it.*
- *The whole application is maintained on GitHub updated time to time based on requirements and while adding new features.*
- *We have also written Python script to generate different kinds of datasets required for different kinds of use cases.*

2.5 Future Implementation(s)

- ***Caching:*** *Caching mechanisms by using unique request id for each requests so backend doesn't have to repeat same process for same type requests and user experience can be enhanced.*

2.6 Directory Structure

- ***Frontend:***
  - *Frontend directory contains "index.html" file which is the root HTML file to run for web page to be shown.*
  - *Frontend directory also contains 2 sub directory named as "css" and "JS" and as name suggests it contains correspondingly CSS and JavaScript files.*
- ***Backend:***
  - *Backend directory contains base directory "algo-comparator" from which the original project of Java is compiled using Spring Boot. The base directory is necessary for successful compilation as it contains useful configuration files too.*
  - *In the "algo-comparator" base directory apart from configuration files and directories there is a "datasets" directory which contains all the datasets required.*
  - *The "algo-comparator" base directory also contains "src" folder which is in general an organized directory structure containing all required source Java files.*

2.7 Libraries Used

- ***java.util.*:*** *Used for Lists and Maps.*
- ***java.io.*:*** *Used for different IO operations like file manipulation required to read data from ataset file.*
- ***org.springframework.*:*** *Used to import Spring Boot application framework based interfaces and classes to use Spring Boot annotation based implementations.*

- ***java.util.concurrent.*:*** *Used for loacks and concurrent hash maps*
- ***java.lang.*:*** *Used for Threading libraries and other default classes*
- ***Exceptions and Exception Handlers:*** *Used for any validation exceptions and handle those exceptions by returning appropriate HTTP response for such exceptions.*
- ***JavaScript and CSS libraries:*** *JavaScript for dynamic data manipulation for client side and CSS for designing.*
- ***chart.js from JavaScript Libraries:*** *Used for plotting charts on the HTML page.*

2.8 Compilation & Running Instructions

- *The frontend code do not need any kind of compilation, one can run the "index.html" file directly on the local browser.*
- *When using Eclipse IDE for the backend code, one can build the maven project by right clicking on the project base directory "algo-comparator" and then "Run As">"Maven Build" and as soon as the window pops up type "clean install" in "Goals" text box and then run.*
- *Once the Maven Build is successful, one can run the application by right clicking on base directory "algo-comparator" then "Run As">"Java Application".*