Team 20 Final Report - Plagiarism Detector for Java

Description of the system's functionality

Academic integrity is one of the most important university policies, expected to be followed and respected by all students. Universities all around the world have some kind of restrictions on work collaboration when it comes to submitting course work. When a student copies their work from other students or from the web without citation or permission, it is considered as Plagiarism. Plagiarism is against the policies of academy honesty and is not acceptable. To solve this problem, our project aims at developing a desktop-based code plagiarism detector. The product will help teaching staff to detect plagiarism in coding by highlighting lines of code that are determined to be copied.

Our product has chosen Java as the target language since it is one of the most common programming languages in the object-oriented paradigm and many universities include in as a part of their curriculum for teaching basic object oriented programming principles, or even for building advanced applications. Our project has been implemented using Java on the backend for running our detection algorithm and user interface was built using Javafx. A user can upload two files. Once uploaded, these files will be previewed. After preview there is option to check plagiarism. The output is displayed in the form of 3 layers. Similarity for each layer is displayed along with a message and some information about that layer.

We conducted some experiments on our application with some sample files that the TAs made available to us and some of our own files. In case of an entire project, one by one comparison would be needed as we do not support multiple files upload for now.

We can detect if the Java project is an empty project while upload. In addition, if users upload two files with the same name, the system will show a warning alerting the user about the same. The user can still check for plagiarism though. If user clicks on the close button of the application by accident, the system will popup a confirmation box to ensure if users really want to exit. Assuming users browse and upload Java source files correctly and click on “Check Plagiarism” button, the second scene will be shown. In this scene, results of three layers will be shown to users in form of 3 panels. In the last tab of the second scene, two tables highlight the lines of code which are detected to be similar. Once users check all results, the “Finish and Return” button can lead them back if they want to check another pair of Java source files.

High-level overview of your design

Comparison is done between 2 programs. We give an individual score for each of our 3 layers. Below are the 3 layers we use to check for similarity:

- Layer 0 shows the hash code comparison.
  If both the uploaded files are exactly same, the score will be 100.0, otherwise it will be 0.0.
- Layer 1 shows the function signature comparison.
  If the two uploaded files have functions with same return type and arguments, they are declared to have the same signature.
- Layer 2 shows the abstract syntax tree comparison.
  It constructs Abstract Syntax Trees of the two uploaded files and analyzes the similarity by Greedy String Tiling algorithm.

Changes from earlier design

We did not implement comment comparison as we had planned to before. Also, we decided to use JavaFX for UI instead of AngularJS. Over the course of designing the project, we realized that we did not really need a web application for the scope of our project. We decided to go with JavaFX because it gives us a simple packaged Java application that we can run on any system by just executing a jar file.

Discussion of the algorithm used for plagiarism detection

The algorithm follows a layered approach. It has three layers: Hash Code Comparison, Function Signature Comparison, and AST Comparison. Each layer has its own score that ranges from 0 to 100. Hash code comparison is intended to catch obvious cases of plagiarism where two files are exact copies of each other. Score of 100 is given if they are same, 0 if not.

Function Signature comparison identifies functions that have the same signature. Signature is composed of the function's return type and its arguments.

Score = number of matching functions / total number of functions * 100

Here the total number of functions corresponds to the program that has lesser number of functions amongst the two. For example, if program 1 has 5 functions, program 2 has 8 functions and 3 functions from program 1 have matching signatures in program 2, the score would be (3/5)*100.

AST Comparison does a more thorough comparison by matching the AST's of 2 programs. The eclipse JDT library is used to parse the program as an AST. Each node is represented by a two digit code along with its start and end line numbers. A string is generated using those two digit codes.

Similarity = total number of the matched nodes / total number of nodes in two programs. The algorithm used for getting substring matches is greedy string tiling.

Coding techniques and Testing

Design patterns:

We used visitor pattern for traversing the AST extracted by JDT.

Testing:

We used a few test cases given by TA's to test our application. Since we do not have support for multiple file, we tested some of the projects by individually selecting the files. We also used some of our own test cases that we have included as a part of our project.

1. We reached 100% statement test coverage for our algorithms.
2. Stress testing for greedy string tiling algorithm by using two long string having length as thousand tokens.
3. JUnit tests are used for white-box testing.

Refactoring:

We used issues raised on Github to help in refactoring our code. We used constants whenever for variables wherever needed. We separated all our test files into a single directory and created a class that had their file paths declared and stored as static strings.

Future Enhancements and limitations

1. Only supports one to one Java file comparison for now
2. Can be extended to compare two entire Java projects in the future easily
3. User login and register functionality can be added and a database can be added for each user to save the history of the plagiarism checks made
4. UI can be improved a dashboard report can show the visualization of the plagiarism