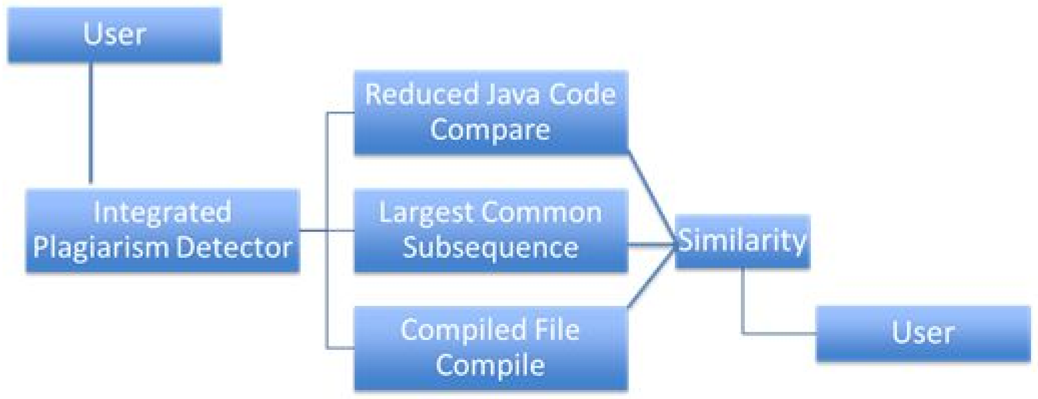
Plagiarism Detection

The objective of this code was to identify plagiarism in other programs. By viewing the source code file, there can be a number of ways to assess the plagiarism of one piece of code to another. A previous experiment run by another group showed that the Longest Common Subsequence algorithm could be employed on Java bytecode post-compilation in order to assess the functional plagiarism of the piece. However, this project also included several additional components that would be able to assess the validity of code plagiarism through various other methods. Much of the assessment and testing of the algorithms are experimental, and not every algorithm will be the catch-all for every case. Therefore, we decided to choose multiple algorithms and approaches as a form of exploratory research so the composite result of all the algorithms is better than the individual components.

The plagiarism detection system is broken into four different segments: the LCS Algorithm, the Code Reduction, the Compilation components, and the Edit Distance Algorithms. The LCS algorithm is contained with the Plagiarism Package, the Code Reduction, the Compilation segment is located in the compile and compiledFileCompare Packages, and the Edit Distance Algorithm is located in the binaryCodeCompare package. All packages are used in the integratedPlagiarismDetector package, which uses the various methods to deduce a similarity value. The user can declare this similarity value to be whatever they choose, providing a threshold for the maximum allowable similarity between code or originality in the piece. Shown below is a graphic of the algorithms being used and documentation of the classes and methods:


User
Integrated Plagiarism Detector
Reduced Java Code Compare
Largest Common Subsequence
Compiled File Compile
Similarity
User

Package javaPlagiarism

VariableEditor

Class that takes in an ArrayList of Strings, and compares each member of the string to another. It is used to assess the similarity of variable names from one ArrayList to another.

Private ArrayList<String> variableNames>

Public VariableEditor()

- Default Constructor. Does not populate any data

Public void setVariableNames(ArrayList<String> variableNames)

- Sets the private member variableNames to the new values passed in

Public ArrayList<String> getVariableNames()

- Returns the private member variableNames

Code Cruncher

Class for reducing the size of a file and populating a VariableEditor from that file. This class will read in a file and begin a loop that reads each line of a .java file in line by line to star the reduction process. Similarly, also handles the start of reduction after import statements. Block comments are stripped away also stripped away from the code, and the VariableEditor is filled with the names of all primitive data types used in the file.

Public CodeCruncher()

- Default Constructor. Does not populate any data

Public File crunchCode(File file1, Variable Editor variable1)

- Initiates the reading of a java file and reads line by line. If the line is not part of a comment, it is passed to LineParser.stripCode(String line), which further reduces it on a line by line basis. Each reduced line is then written to another file of the same name + “Edits”. If the file already exists, it is deleted upon running this method.
- Returns Reduced File of Java Code

LineParser

Class used to parse a line of code and remove any unnecessary syntax that Java requires in order to reduce the overall effect of the Java enforced syntax and code stylings in the LCS algorithm. By minimizing the amount of forced structure and design, then more user-content can be evaluated by the LCS algorithm instead.

Private ArrayList<String> variableNames

Public LineParser()

- Initializes an empty ArrayList of type String for its private member

Public ArrayList<String> getVariableNames()

- Returns the values of its private member

Public String stripCode(String line)

- Performs a series of evaluations against a line of Java code passed in as a parameter. Loops are reduced to the breaking member plus an 'l' marking that this was a loop. If-else statements are similarly marked for their conditions and reduced to 'z' plus those conditions. This builds a edited, reduced code segment from the parameter passed in. Then, a series of regex line.replaceAll(regex, replace) will remove any keywords left in the code, spaces, tabs, parenthesis, and semi-colons. Certain keywords are instead repalced

SyntaxScore

Head-level class of the javaPlagiarism package. It takes in a string of two file paths and passes them to CodeCruncher for evaluation. The returned file from CodeCruncher is then executed through the LCS algorithm and a weighted sum is computed from the VariableEditors of each file, which returns the

Public double getScore(String file1, String file2)

- Takes in two file paths as Strings, and then passes the respective files to CodeCruncher for analysis. The returned reduced file and the corresponding VariableEditors are weighted and summed, currently .9 times the LCS algorithm of the reduced file plus .1 times the VariableEditor similarity

Package pla

Ss

Class used to finding the longest subsequence common to all sequences in a set of sequences (often just two sequences). It differs from problems of finding common substrings because, unlike substrings, subsequences are not required to occupy consecutive positions within the original sequences.

Public List<string> getSameStr (String str1, String str2)

- Uses the longest common subsequence algorithm to get the length of the same String. Add the length to the list, return list.

Plagiarism

Class used to make a judgement that if the length of LCS is longer than 10. If it is longer than 10, it will be removed from the search strings and added to the sum of the same strings.

Public double pla (String path1, String path2)

- Deletes the same string we got. Continue to get LCS from the modified string. And add it to the sum of the same string.

Package Compile

This package contains program to compile source code in .java, .cpp, .py.

CompileJava Class

ComlileJava class implements from Compile interface. This class is designed for read a .java file, compile it and finally run it.

Private File file

Private String direction

Public CompileJava (File file, String direction)

- Initializes the class with the required components of compiling a java program with the file being the source code file and direction being the file path

Public void compileSourceCode ()

- Compiles a java program with the filename specified as the source code. Requires the private members to be qualified upon initialization

JavaStringObject

JavaString can convert a string into JavaFileObject. This class will finally be used by CompileJava.

Private String code

Public JavaStringObject (String name, String code)

- Constructor with required parameters to compile a Java File Object

Public CharSequence getCharContent()

- Getter method to return the private member String code

Reader

Class used for reading in a file and appending all code to a string for passing as a parameter

Public String getCode()

- Reads in a file and appends each line of code into a string separated by newline characters

CompilePython

CompilePython class can compile .py file into .pyc file.

Environment Requirement:
Install python3;

Set python in system environment.

Private File file

Public CompilePython(File file)

- Constructor for settings its private member

Public void compileSourceCode ()

- Compiles python using command line arguments on the file specified in its private member

CompileCpp Class

Compile .cpp file into .obj file.

Enviroment Requirement:
Install Visual Studio 2010.
Set vcvarsall.bat into system environment.

Private File file

Public CompileCpp(File file)

- Constructor for settings its private member

Public void compileSourceCode ()

- Compiles python using command line arguments on the file specified in its private member

Package binaryCodeCompare :

EditDistanceAlgorithm

This class performs the edit distance algorithm given two byte arrays.

Private int min(int i, int j, int k)

- Finds the minimum of the three parameters

Private int max(int i, int j, int k)

- Finds the maximum of the three parameters

Private class Matrix

  Private int rows

  Private int cols

  Private int[] data

  Public Matrix(int rows, int cols)

  - Constructor setting the matrix as a 1D array of length row*cols

  Public void set(int value, int i, int j)

  - Sets a member of the array to value at position i*cols + j

  Public void get(int i, int j)

  - Gets a member of the array to value at position i*cols + j

Public double EDAValue(byte[] a, byte[] b)

- Performs the Edit Distance Algorithm on two byte arrays passed as parameters

Package intergratedPlagiarismDetector

integratedPlagiarisimDetector
Group LCS algorithm, Edit algorithm and Reduced java compare together, and return all suspected files to user.

> **Private ArrayList<File>**
> **Private File file**
>
> **Public PlagiarismDector(ArrayList<File> fileList, String type)**
> - Constructor containing a list of files to look for plagiarism in, and the type of file (java, python, cpp)
>
> **Public ArrayList<File[]> getSuspectedFilePair()**
> - Grabs each file and compares it to every other file. Performs the LCS algorithm on the plaintext file, the LCS algorithm on the Reduced Code file, and the Edit Distance Algorithm. The sum of each algorithm is combined. If it crosses the threshold, it prints a message to console warning of plagiarism. Otherwise, it prints a message noting no plagiarism detected.

Overall, the project was successful. Due to the nature of the project, the group often worked separately on different components to experimentally try different methods to derive a solution. Since no solution we deemed would be ideal and cover every case, the group proposed three possible methods, and the composite of each can help increase the accuracy of the project as an ensemble method for plagiarism detection. In doing this, the project will be able to cover more variability in the samples, and the amount of plagiarism can be increased or decreased depending on the threshold level. This project enables the handling of multiple events and situations that may arise due to copying code.