Plagiarism Detector

Objective

The objective of the project is to build a plagiarism checker that checks for similarity between two given input java source files or two directories containing java source files uploaded by the user in the given interface.
In this document we briefly discuss the algorithm for calculating the structural similarity of the given two java project directory/files

Functionality

The project aims at building a plagiarism checker:

- Compare 2 files.
- Compare 2 project directories.
- Calculate and displays similarity percentage.
- Detect similarity when structural changes like method rearranged within a file and split into files.
- Highlight/view plagiarized code.
- View all the plagiarized files within a directory side by side in the application.

Design

1. System

We have used the Strategy design pattern for our system.
"The strategy pattern is a behavioral software design pattern that enables selecting an algorithm at runtime. The strategy pattern-

- defines a family of algorithms,
- encapsulates each algorithm, and
- makes the algorithms interchangeable within that family.

Strategy lets the algorithm vary independently from clients that use it." – Wikipedia

This pattern is based on the "open-closed principle". In this pattern, the client is coupled to an abstract class and not the actual implementation- abstract coupling. This ensures that the client does not suffer if the implementation is changed.

2. AST Node

Visitor pattern:

- We use visitor pattern to traverse and populate the complete list on child nodes from given root node of the given java source file. This is essentially a top down approach.

- Then from this list we can use the bottom up approach to bubble up the hash value and descendant nodes up from the child nodes to root.

3. Concrete Classes

- Our project us the factory class pattern to instantiate the concrete classes.

Algorithm - Hashbased clone detection algorithm.

```
1.Clones = Nil
2. For source file and suspect file do:
        3. RootNode <- CreateAST(File) // Using eclipse JDT
        4. Create Node list <- using the visitor pattern in a top down // ASTVISITor
        5. While creating the node list add relevant attributes like hashcode
             and number of descendants in the bottom up approach
7. Sort node list srcNodes susNodes based on hashvalue and number of descendants
8. For each $s_i$ in srcNodes and $s_j$ suspect file: //iteratively check for equality between nodes.
                9.if $s_i$ ==sj
                        10.Clones.add(si)
11. For each Node $n_i$ in similarNodes
        12. Print $n_i$.toString() //(textColor = red if $n_i$ in Clones)
```

Experiments/Tests

We used the sample projects provided by the TAs as the input to test our system.The test suite includes tests specific to each class. The application handles the following:

Use Case	System implementation	Test Case
Empty folder is uploaded	Message for User to upload files	compareEmptyFolder
Folder/files that is not a java file is uploaded	Message for User to upload the right files	compareNonJava
Files which has 100% same code with the same variables	Computes 100% similarity and displays similar lines of code	comparExactFiles
Files in which the variable names have been changed	Detects as similar and and displays as similar lines of code	compareSet01
Files in which the loops have been refactored	Detects as similar and displays the similarity percentage and lines of code	compareSet05

Multiple java classes in given two directories	Detects the similarity of two files after creating a combined AST	compareSet02
Files where the methods are reordered	Detects the similarity and displays the copied content	compareSet01

Refactoring

- We have changed our architecture from client server to a standalone to focus more on the core of the project rather than client server communication
- Also removed login feature which were not the core requirement after feedback from the professor.
- We have tried to refactor few functions to follow single responsibility and also make the code more modular.

Bugs/Enhancements:

- The highlighting part needs more work. This was something we need bit more time.

Testing:

- Extensive and robust test cases are written – 72 cases
- Test coverage for the backend/algorithmic part – Unit tests – 98%
- Test coverage for the front end – UI test – 37%

Conclusion

We explored plagiarism detection with respect to structural difference between two source directories. This is very efficient in detecting plagiarism wrt :

- Variable renaming
- Changing the structure of the file
- Interchanging looping constructs.
- Interchanging conditional statements.

We were able to implement our intended algorithm and have achieved suitable accuracy. To do so we changed the default hash values to custom hash values which helped in detecting specific cases of plagiarism and increased the efficiency of the system.