Plagarism detection in code is the method and process used for locating instances of cloned code within a set of documents. The tool we have chosen to demonstrate and explore plagiarism detection is JPlag.

JPlag is a system that finds similarities, software plagiarisms, among multiple source code files. It is robust against many attempts to disguise the copied code because it is aware of programming language syntax and program structure. Thus, it is very hard to deceive: 90% of plagiarisms are detected and the rest raise suspicion. [1] Moreover, JPlag is able to process a hundred programs with several hundred lines of code in seconds, making it a very robust and scalable tool.

It has been successfully used for detecting plagiarism among students’ Java programs but support for other languages such as C and C++ are also available.

JPlag’s comparison algorithm is based on the “Greedy String Tiling” [2], which takes in a set of program source code as input and outputs a similarity score between pairs of programs and their corresponding similarity regions. To do so, it must initially convert the program source code in to token strings, this is the only language dependent process involve din JPlag’s plagiarism detection. Tokens should be chosen in a manner, which captures the essence of the program’s structure rather than the surface aspects, as a program’s structure is harder for plagiarists to modify. JPlag does this phase whilst ignoring whitespaces, comments and names of identifiers. Figure1 shows an example Java code and its corresponding tokens generated by JPlag. A list of possible JPlag tokens can be found in this technical report [3].

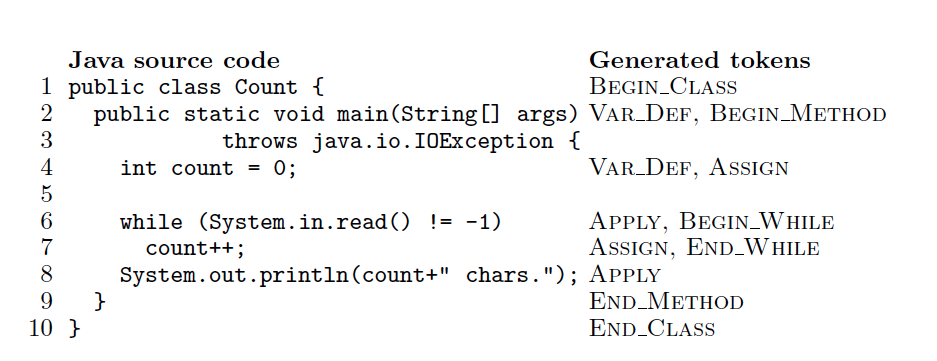


Figure A sample Java Source code with the generated token strings

Once the token strings have been generated, JPlag beings the comparison phase by comparing the two generated token strings. When comparing two strings A and B, JPlag’s objective is to discover a maximal set of contiguous substrings. Each substring must occur in both A and B and must be as long as possible. These substrings (matches) must be unique in that each substring should not cover tokens already covered by other substrings. To avoid false matches, a minimum match length M is enforced.

The Greedy String Tiling algorithm itself has two steps Figure2. Firstly, all substrings common to both token strings are found with lengths equal to or greater than the minimum M. Secondly, all the tokens found in the substrings are marked so that they are no longer picked up by subsequent iterations of the first step. By marking its tokens, a match becomes a “tile”. Thus, the similarity score between is calculated based on the fraction of tokens that were found as matches:

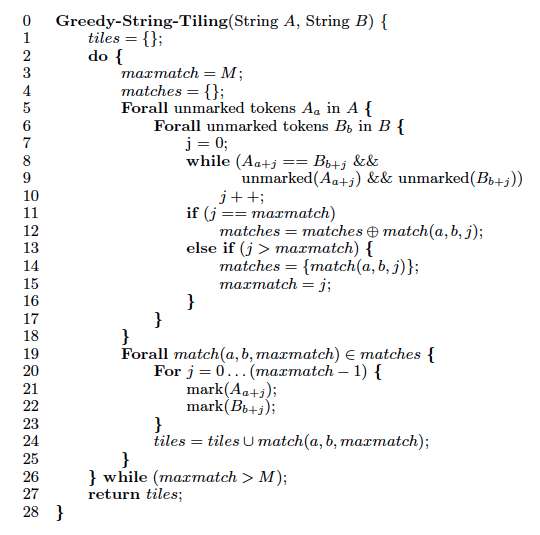


Figure Greedy String Tiling algorithm

[1] Prechelt, Lutz, Guido Malpohl, and Michael Philippsen. "Finding plagiarisms among a set of programs with JPlag." J. UCS 8.11 (2002): 1016.

[2] Michael J. Wise. String similarity via greedy string tiling and running Karp-Rabin matching. ftp://ftp.cs.su.oz.au/michaelw/doc/RKR GST.ps, Dept. of CS, University of Sydney, December 1993.

[3] Prechelt, Lutz, Guido Malpohl, and Michael Philippsen. “JPlag: Finding plagiarisms among a set of programs.” Technical Report 2000-1, March 2000