Grade Your Code: An Online Platform for Conducting Programming Quizzes

Shoaib Ahmed Muhammad Fawad Jawaid Malik Zohaib Masood Syed Muhammad Zohair Naqvi Faraz Idris Khan Muhammad Rafi

Computer Science Department

National University Of Computer and Emerging Sciences, Karachi

FAST-NUCES

{k112016, k112116, k112114, k112181, faraz.idris, muhammad.rafi}@nu.edu.pk

**Abstract**

Grade Your Code is an online platform where teachers can make programming questions or pick questions from a library of programming questions and post a quiz comprising of these questions. They also decide the grading criteria for every question according to test cases. Plagiarism rate is mentioned by the instructor.

On the student's side, there is a compiler, which lets student to try their code again and again and show the result of test cases implemented on his code. When the student gets satisfied by his result he can submit the answer and he will get the grades according to grading and plagiarism criteria. There is also a timer and a counter to note the time taken to solve the questions and the number of retries by the student, which will be used to determine the difficulty level of question as well as skill level of student.

**1.** **Introduction**

Grade is a web portal containing an online compiler of C, C++ and Java and a plagiarism detection engine. The purpose of this project is to provide a platform to the teachers who are teaching programming courses to conduct online and offline quizzes related to programming only. The teacher is able to make programming questions and is also facilitated with a list of questions shared by other teachers. He has the facility to create a quiz, add questions and test cases, add starting and ending time, assign points to each question and select the students who can attempt the quiz. They will be asked to mention the grading criteria for every question according to the test cases. Plagiarism rate is also be defined by the teacher.

On the student's side, there is a compiler. The students can login through their student IDs and then they can start coding in one of the above mentioned languages or the language mentioned by the teacher if any. Using the online compiler, the students can compile their code, run the test cases given by the teacher, remove the bugs and view the output of their code. The students can run their code again and again and can

view the result of the test cases implemented on their code. The students are also able to check the plagiarism percentage of their codes. When the student gets satisfied by the output of his/her code then he/she can submit his/her code. The students will get the grades according to the grading and plagiarism criteria. They are allowed to submit their code till the deadline mentioned by the teacher. Only the teacher has the authority to change the deadline. The students will not be able to submit their quizzes after the deadline. There is also a timer and a counter to note the time taken to solve the question and number of retries by student, which will be used to determine the difficulty level of question as well as skill level of student. A student can view his profile, the courses in which he/she are registered and the quizzes given by the teacher. The system will grade the codes written by the students on the basis of the number of test cases running successfully and the plagiarism percentage.

**2. Literature Review**

We have done extensive researched to achieve the goals of this project. We have reviewed some of the available Grade Your Code like projects two of them are hacker rank and remoteinterview.io. HackerRank is a site for hackers from all over the world to solve programming problems in different CS domains like algorithms, machine learning and artificial intelligence, and to excel in different programming paradigms like functional programming. Whereas, Remote Interview is a combination of two solutions:

Screening tests, helps you create tests which shall be sent to your candidates. After those candidates solve your given test, the scores are automatically calculated and detailed reports are presented to you so to help you pick the best ones quickly. The interviewing tool is for conducting interviews with programmers in the browser, like CollabEdit, or Stypi, with some major features which include: Code Compiler, Note Taking, Advanced Reporting etc.

For the plagiarism part we have done some researched on MOSS (Measure of Software Similarity). Moss (Measure of Software Similarity) is a system for finding the resemblance among the program codes. The objective of the application ‘Moss’ is to identify plagiarism in programming classes. This application has proved to be very effective till now. It was developed in 1994. The algorithm working at the backend of the application is a unique one as that of other plagiarism detection engines.

**3. Online Compiler**

It has always been difficult for programmers to organize and work on projects across a variety of locations, computers, and devices – especially when working in teams. When a code is stored on one computer at one location, the developer is bound to that one computer, finding himself running back to that machine. Worse, while working in a team, each member of the team has to send his/her code to every other member, each step of the way, creating organizational distractions and unnecessary delays. This is the problem that online compilers have solved, and the programming world will never be the same.

With an online compiler, the developers can store their code online, allowing them the freedom to program from any computer (or device) with a connection to the internet. With an online compiler, a team can work on the exact same set of files, regardless of where the team members are, allowing programmers to work better as a team and keep their projects organized.

* 1. Independent of Tools

Online IDEs have cut the number of tools you need tenfold – now, to access and edit your code, all you need is an internet connection. This allows you to access and edit your code from just about any computer worldwide, freeing you from the need to have constant access to a single computer where your files are stored.

* 1. Team Work

When working in a team, it’s essential to keep all team members on the same page, keep team members updated to the latest changes in code, and keep projects organized without a confusing jumble of different project versions stored on different computers. An online compiler allows a team to work on the exact same set of code, browsing files just as they would if they were stored on their local machine, marking the end of organizational hassles.

* 1. *Online IDE*

Shipped with just about language/library/etc. you’ll ever need, online IDEs come ready to program out of the box. With an online IDE, you can get your projects up and running faster than ever by skipping over tedious installations, and getting right down to the programming of your project itself.

* 1. *Program with Any Device*

Can you log into your online compiler with your Smartphone or tablet, and edit your code? Absolutely. It might seem a bit awkward to consider writing code on your iPhone or BlackBerry, but when you have a client demanding coding changes fast, you will be glad you can. With an online IDE, you can log in with your Smartphone, edit a project’s code, and have it sent off to your client in a matter of minutes.

**4.** **Plagiarism detection engine**

The process of discovering plagiarism among certain documents or files is called Plagiarism Detection. In this age of computers and internet, copying others work has now become a common practice. Majority of such cases are found in academia where assignments and home tasks are of textual form.

Plagiarism detection engine is software that finds the similarities between documents or other files by comparing these files with the help of available sources and determines whether the document submitted by a student is plagiarized or not.

* 1. *What is Moss?*

Moss (Measure of Software Similarity) is a system for finding the resemblance among the program codes. The objective of the application ‘Moss’ is to identify plagiarism in programming classes. This application has proved to be very effective till now. It was developed in 1994. The algorithm working at the backend of the application is a unique one as that of other plagiarism detection engines.

* 1. An Internet Service

Moss is an internet service. It is very easy to use. A list of files are provided to Moss to find the plagiarism if any. The Moss server generates HTML pages listing pairs of programs with similar code. It also detects individual methods in programs that appear the same. It can remove similarities that are obvious e.g. libraries or code provided by the teacher. So in this way it can differentiate between the codes, copied or not.

* 1. *Languages Supported*

Moss can detect plagiarism among the codes written in these languages:

* C
* C++
* Java
* Python
* Visual Basic
* JavaScript
* FORTRAN
* ML
* Haskell
* Lisp
* Scheme
* Pascal
* Modula2
* Ada
* Perl
* TCL
* VHDL
* MIPS assembly
* a8086 assembly
  1. *Moss Code*

Certain users of Moss have contributed versions of the submission script:

1. A Java version (from Bjoern Zielke)
2. A PHP version (from Phillip Rehs)
   1. *How Does It Work?*

We have researched on the working perspective of Moss Plagiarism Detection Engine and found a paper named “Winnowing: Local Algorithms for Document Fingerprinting” by Saul Schleimer, Daniel S. Wilkerson and Alex Aiken.

According to this paper, they have introduced the class of local document fingerprinting algorithms, which seems to capture an essential property of any fingerprinting technique guaranteed to detect copies. They have proved a novel lower bound on the performance of any local algorithm. And also developed winnowing, an efficient local fingerprinting algorithm, and show that winnowing’s performance are within 33% of the lower bound. Finally, they have also given experimental results on Web data, and report experience with MOSS, a widely-used plagiarism detection service.

* 1. *Winnowing*

Winnowing is an algorithm for selecting fingerprints from hashes of k-gram. An upper bound on the performance of winnowing, expressed as a trade-off between the number of ﬁngerprints that must be selected and the shortest match that we are guaranteed to detect. Given a set of documents, we want the find substring matches between them that satisfy two properties:

1. If there is a substring match at least as long as the guarantee threshold, t, then this match is detected.
2. We do not detect any matches shorter than the noise threshold, k.

A do run run run, a do run run

1. Some text from

adorunrunrunadorunrun

1. The text with irrelevant features removed.

adoru dorun orunr runru unrun nrunr runru

unrun nruna runad unado nador adoru dorun orunr runru unrun

1. The sequence of 5-grams derived from the text.

77 72 42 17 98 50 17 98 8 88 67 39 77 72 42 17 98

1. A hypothetical sequence of hashes of the 5-grams.

72 8 88 72

1. The sequence of hashes selected using 0 mod 4

The constants t and k ≤ t are chosen by the user. We avoid matching strings below the noise threshold by considering only hashes of k-grams. The larger k is, the more confident we can be that matches between documents are not coincidental. On the other hand, larger values of k also limit the sensitivity to reordering of document contents, as we cannot detect the relocation of any substring of length less than k. Thus, it is important to choose k to be the minimum value that eliminates coincidental matches.

Figures 2(a)-(d) are reproduced from Figure 1 for convenience and show a sequence of hashes of 5-grams derived from some sample text.

**Figure 1:** Fingerprinting some sample text.

Given a sequence of hashes h1 . . . hn, if n > t − k, then at least one of the hi must be chosen to guarantee detection of all matches of length at least t. This suggests the following simple approach. Let the window size be w = t − k + 1. Consider the sequence of hashes h1h2 . . . hn that represents a document. Each position 1 ≤ i ≤ n − w + 1 in this sequence defines a window of hashes hi . . . hi+w−1. To maintain the guarantee it is necessary to select one hash value from every window to be a fingerprint of the document. We have found the following strategy works well in practice.

*DEFINITION (WINNOWING):* In each window select the minimum hash value. If there is more than one hash with the minimum value, select the rightmost occurrence. Now save all selected hashes as the fingerprints of the document.

Figure 2(e) gives the windows of length four for the sequence of hashes in Figure 2(d). Each hash that is selected is shown in boldface (but only once, in the window that first selects that hash).

A do run run run, a do run run

1. Some text.

adorunrunrunadorunrun

1. The text with irrelevant features removed.

adoru dorun orunr runru unrun nrunr runru

unrun nruna runad unado nador adoru dorun

orunr runru unrun

1. The sequence of 5-grams derived from the text.

77 74 42 17 98 50 17 98 8 88 67 39 77 74 42 17 98

1. A hypothetical sequence of hashes of the 5-grams.

(77, 74, 42, 17) (74, 42, 17, 98)

(42, 17, 98, 50) (17, 98, 50, 17)

(98, 50, 17, 98) (50, 17, 98, 8)

(17, 98, 8, 88) (98, 8, 88, 67)

(8, 88, 67, 39) (88, 67, 39, 77)

(67, 39, 77, 74) (39, 77, 74, 42)

(77, 74, 42, 17) (74, 42, 17, 98)

1. Windows of hashes of length 4.

17 17 8 39 17

1. Fingerprints selected by winnowing.

[17, 3] [17, 6] [8, 8] [39, 11] [17, 15]

1. Fingerprints paired with 0-base positional information.

**Figure 2:** Winnowing sample text.

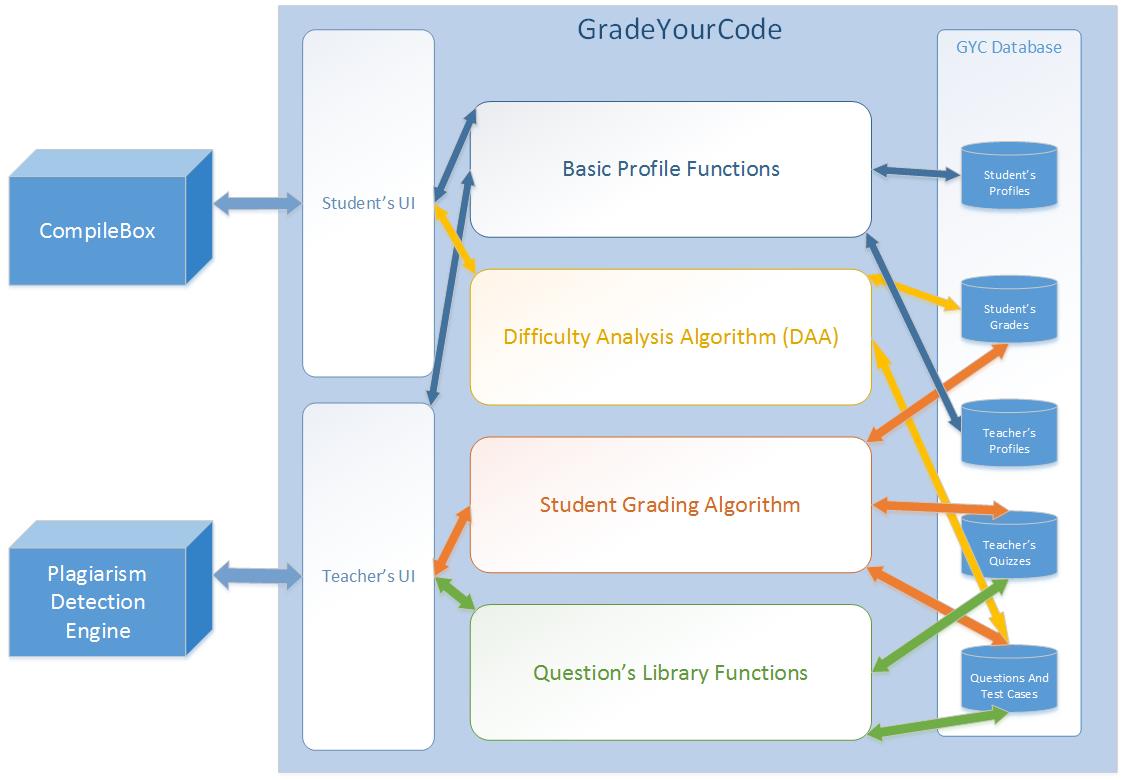
The intuition behind choosing the minimum hash is that the minimum hash in one window is very likely to remain the minimum hash in adjacent windows, since the odds are that the minimum of w random numbers is smaller than one additional random number. Thus, many overlapping windows select the same hash, and the number of fingerprints selected is far smaller than the number of windows while still maintaining the guarantee. Figure 2(f) shows the set of fingerprints selected by winnowing in the example. In many applications it is useful to record not only the fingerprints of a document, but also the position of the fingerprints in the document. For example, we need positional information to show the matching substrings in a user interface. An efficient implementation of winnowing also needs to retain the position of the most recently selected fingerprint. Figure 2(f) shows the set of [fingerprint, position] pairs for this example (the first position is numbered 0). To avoid the notational complexity of indexing all hashes with their position in the global sequence of hashes of k-grams of a document, we suppress most explicit references to the position of k-grams in documents in our presentation.

* 1. *Plagiarism Detection*

One of the authors has run MOSS, a widely-used plagiarism detection service, over the Internet since 1997. MOSS is primarily used for detecting plagiarism in programming assignments in computer science and other engineering courses, though several text formats are supported as well. The service currently uses robust winnowing, which is quite efficient and scalable (in the sense that it selects fewer fingerprints for the same quality of results).

**5. System Architecture**

There will be a web portal which contains an online compiler of C, C++, Java and Python and a plagiarism detector. The teacher will set the questions and a plagiarism rate for the quiz and will also provide certain test cases for each question. The students will login through their student IDs and then they will start coding in one of the above mentioned languages or the language mentioned by the teacher if any. Using the online compiler, the students will compile their code, run the test cases given by the teacher, remove the bugs and view the output of their code. The students will also be able to check the plagiarism percentage of their codes. If the plagiarism percentage of a code submitted by a student is greater than the plagiarism rate set by the teacher then the student will get a zero in that question. They will be allowed to submit their code till the deadline mentioned by the teacher. Only the teacher will have the authority to change the deadline. The students will not be able to submit their quizzes after the deadline. The system will grade the codes written by the students on the basis of the number of test cases running successfully and the plagiarism percentage.



**Figure 3:** SystemArchitecture of GradeYourCode.

**6. Evaluation**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **HackerRank** | **MOE** | **GradeYourCode** |
| **Open for public** | **Yes** | **Yes** | **No (Restricted to institutions)** |
| **Automatic Grading** | **Yes** | **Yes** | **Yes** |
| **Plagiarism detection** | **No** | **No** | **Yes** |
| **Paid Service** | **Yes** | **No** | **No** |
|  |  |  |  |

In comparison with other platforms providing similar services as Grade Your Code, GYC is not an open service. It has limited users. These users include students registered in programming related courses and the teachers teaching such courses in an institution. Other than these users don’t have access rights to the services of GYC.

In order to facilitate the teachers and assist them to grade the programming quizzes submitted by the students, GYC provides the facility of automatic grading of these quizzes. The grading of quizzes is done on basis of the test cases satisfied by the code and the plagiarism percentage. In addition to this, GYC has a service of plagiarism detection engine which helps teachers to discover the plagiarism cases. Using this service, teachers get an opportunity to check plagiarism automatically rather than doing it manually. Moreover, GYC provides its services with free of charges.

**7. Conclusions**

In this paper, we have proposed a project “Grade Your Code” in which we are providing a platform to the teachers teaching programming courses to conduct online and offline quizzes related to programming only. There is a web portal which contains an online compiler of C, C++, Java and Python and a plagiarism detector. The teacher can set the questions and a plagiarism rate for the quiz and will also provide certain test cases for each question. The students can login through their student IDs and then they can start coding in one of the above mentioned languages or the language mentioned by the teacher if any. Using the online compiler, the students can compile their code, run the test cases given by the teacher, remove the bugs and view the output of their code. The students are also able to check the plagiarism percentage of their codes. If the plagiarism percentage of a code submitted by a student is greater than the plagiarism rate set by the teacher then the student will get a zero in that question. They are allowed to submit their code till the deadline mentioned by the teacher. Only the teacher has the authority to change the deadline. The students will not be able to submit their quizzes after the deadline. The system will grade the codes written by the students on the basis of the number of test cases running successfully and the plagiarism percentage.

For plagiarism detection, we have used Measure of Software Similarity (MOSS). Moss can detect plagiarism among the codes written in these languages: C, C++, Java, Python, Visual Basic, JavaScript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, VHDL, MIPS assembly, a8086 assembly. Certain users of Moss have contributed versions of the submission script: A Java version and A PHP version.

**Acknowledgments**

First of all we thank to **ALLAH (SWT)** for everything HE granted us and then we thank to our supervisor **Sir Faraz Idris** for his assistance and generosity in his area of expertise, and for his valuable input. We are also thankful to **Sir Muhammad Rafi** for guiding us and helping us to complete the project within the deadline. This project could not be completed without his help that he assisted us in all possible ways. We are also grateful to **Sir Bilal Hayat** who suggested us to take this project as our final year project and gave us confidence and motivation to face the challenge.

**References**

1. Saul Schleimer, Daniel S. Wilkerson, Alex Aiken. Winnowing: Local Algorithms for Document Fingerprinting.
2. “The Source Code Plagiarism Detection Using AST” by Xiao Li; Sch. of Computer. Sci. & Technol., Southwest Univ. for Nat., Chengdu, China; Xiao Jing Zhong. October 2010 <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5663286>
3. “Automatic Source Code Plagiarism Detection” by Kustanto, C.; Sch. of Electr. Eng. & Inf., Inst. Teknol. Bandung, Bandung, Indonesia; Liem, I. May 2009 <http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?tp=&arnumber=5286623&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5286623>
4. “An Approach to Source-Code Plagiarism Detection and Investigation Using Latent Semantic Analysis” by Cosma, G.; P.A. Coll., Larnaca, Cyprus; Joy, M. March 2012 <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6086533&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6086533>
5. “Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises” by Aleksi Ahtiainen Helsinki University of Technology, Finland Sami Surakka Helsinki University of Technology, Finland Mikko Rahikainen Helsinki University of Technology, Finland, 2006 <http://dl.acm.org/citation.cfm?id=1315831>
6. “Sphere Engine” <http://sphere-research.com/sphere-engine.html>