Olympiads in Informatics, 2024, Vol. 18, 51–62
© 2024 *IOI, Vilnius University*
DOI: 10.15388/ioi.2024.04

Analysis and Evaluation of the Contestant's Progress in Real-time Coding Contests

Mirvari MAMMADLI, Nihad MAMMADLI, Jamaladdin HASANOV
ADA University, School of IT and Engineering
Ahmadbey Aghaoglu str. 61, 1008 Baku, Azerbaijan
e-mail: mmammadli13254@ada.edu.az; nmammadli14004@ada.edu.az; jhasanov@ada.edu.az

Abstract. This paper presents a model for analyzing contestant progress in real-time coding contests, emphasizing the critical need for effective measures in assessing code similarity and plagiarism cases. Current coding contest platforms often need more robust procedures to identify and address these issues, compromising the integrity of the evaluation process. To tackle these challenges, we propose a novel system that leverages advanced techniques to analyze code and collective behavior, providing a holistic evaluation of submissions. The system enhances performance assessment accuracy and maintains fairness and credibility in real-time coding contests. The findings and insights from this study shed light on the importance of integrating sophisticated mechanisms to ensure the authenticity of code submissions and uphold the competitive nature of coding competitions.

Keywords: plagiarism, contest, CMS, real-time, C++.

1. Introduction

The rise of online platforms has shifted competitive programming and coding contests into the digital world, bringing forth a pressing issue – plagiarism. As the number of online coding platforms increases and resources for cheating become more available, some contestants' temptation to engage in unethical practices has grown exponentially. Ensuring fairness in coding contests is crucial and a central concern for researchers and instructors. However, most coding contest platforms lack solid systems to identify plagiarism in real time, jeopardizing the validity of the evaluation process.

Moreover, the analysis of contest data is pivotal regarding the quality and fairness of coding competitions. As mentioned in (Hasanov *et al.*, 2021), based on the study of the International Olympiad in Informatics (IOI), statistical analysis of the competition data can provide valuable insights into the strategies used, competition dynamics, and contestant performance. Contest data analysis in competitions enables the organizers, coaches, and participants to identify the shortcomings, strengths, and areas for improvement. Therefore, we ensure real-time data visualization to facilitate further analysis in the proposed system. Using data-driven approaches, coding contests can be optimized to promote fair competition.

This review explores plagiarism detection in programming, specifically in real-time coding contests, emphasizing the need for customized solutions to address their unique challenges.

1.1. *What is Code Similarity, and How Does it Work*

Code similarity involves assessing the resemblance between different code pieces to identify potential plagiarism instances. Understanding the concept of code similarity in the context of programming and software development is essential. A comparison of syntactic and semantic structures of code pieces usually takes place when assessing code similarity. One of the most common approaches includes token-based comparison (Yetthapu, 2023). The source code is divided into essential components known as tokens, which could consist of parts such as literals, identifiers, and keywords. Subsequently, the overlap or similarity of these tokens across various code segments is examined. This particular method does not necessarily consider structure or semantics. Thus, this method captures syntactic similarities between code segments (Prechelt *et al.*, 2000). To maintain the integrity of coding contests, our project introduces checker.js, which includes three methods: comparator, cppChecker, and remover. Traditional methods face challenges adapting to source code nuances, as noted by Prechelt *et al.* (2002). Also, there has been widespread use of string matching algorithms like the Longest Common Subsequence (LCS) algorithm (Myers, 1986). However, our approach presents a remover method that goes beyond conventional string matching, using regular expressions to improve accuracy in code similarity detection.

1.2. *Plagiarism in Contests*

One of the main reasons for jeopardizing the evaluation process's integrity in programming contests is plagiarism. As mentioned in (Wu *et al.*, 2022), cheating has become challenging. Real-time detection of plagiarism, crucial for contest integrity, is a key focus. The comparer method from our project, which is aligned with timely interventions during competitions, uses async for concurrent processing. This strategic approach ensures real-time plagiarism evaluations, emphasized by Jeske *et al.* (2018). The cppChecker method, managing multilingual code comparison, is vital given the array of programming languages used in coding contests. As participants engage in diverse coding languages, the capability to manage this variability becomes crucial in maintaining fair and unbiased contest conditions. Academic institutions and contest organizers prioritize solid measures to restrain plagiarism, fostering an atmosphere where skills and imagination can be seen. Checking and comparing each piece of code could be too time-consuming for the instructors. Therefore, before introducing our system, an analysis of previous systems and their advantages and disadvantages is considered in Section 2. The approach used in the project is mentioned in Section 4.

2. Previous Work

Some of the prominent and extensively cited state-of-the-art plagiarism detection systems, according to Burrows *et al.* (2007), are JPlag (Prechelt *et al.*, 2002) and MOSS (Schleimer *et al.*, 2003). Additionally, there exist several other noteworthy systems, such as Sim (Gitchell & Tran, 1999), Plague (Clough, 2000), YAP (Wise, 1996), Plaggie (Hage *et al.*, 2011), and FPDS (Mozgovoy *et al.*, 2005). Each system employs diverse methodologies and algorithms to address the intricate challenges of detecting plagiarism in programming code (Đurić & Gasevic, 2013).

In 1996, during his student project at Karlsruhe University, Guido Malpohl created a plagiarism detection tool, JPlag**,** which evolved into the first online system. Subsequently, with the effort of Emeric Kwemou and Moritz Kroll, the online system transformed into a web service (Prechelt *et al.*, 2000). This tool specializes in plagiarism detection across C, C++, Java, and Scheme programming languages. The underlying architecture of JPlag relies on the Greedy String Tiling comparison algorithm (Prechelt *et al.*, 2002). Compared to other tools, JPlag demonstrates superior plagiarism detection performance (Yetthapu, 2023). As stated in (Yetthapu, 2023), JPlag processes submitted source code assignments and then generates HTML pages as output, with code similarity values ranging from 0% to 5% indicative of no plagiarism and 100% representing blatant plagiarism. Intermediate values, such as 40%, warrant further manual investigation for a conclusive judgment (Prechelt *et al.*, 2000).

Code comparison tools like **MOSS** (Measure of Software Similarity) are frequently used in software development projects to find instances of code plagiarism. MOSS is a free online service created in 1994 by Aiken *et al.* at Stanford University. It functions as a web service. It offers an easy-to-use web interface with Windows and UNIX operating systems. MOSS splits code into continuous substrings called K-grams using the Winnowing algorithm and hashes each K-gram (Luke *et al.*, 2014). MOSS uses several algorithms to increase efficiency, such as control flow analysis, code structure analysis, and string matching. About twenty-five programming languages, including Java, C, C++, Python, JavaScript, Matlab, VHDL, and Verilog, are supported by it (Hage *et al.*, 2010). Results from MOSS are presented in HTML through a graphical interface that highlights suspicious code fragments, the percentage of similarity, tokens, and matched lines. Each matched pair is clickable, facilitating manual inspection (Yetthapu, 2023).

As mentioned in (Schleimer *et al.*, 2003), which connects to our topic, the authors investigate the concept of local document fingerprinting algorithms to identify copying across large datasets accurately. They describe the efficient winnowing algorithm and demonstrate its performance within 33% of the lower bound. The paper discusses the difficulties in detecting partial copies and suggests using the k-grams mentioned above and hashes as fingerprints. The article emphasizes the importance of selecting the appropriate "k" value to eliminate coincidental matches.

In the context of document fingerprinting, the article explains the mechanics of the winnowing algorithm, as well as querying and optimal hash selection. It examines the correctness of local algorithms for finding matching substrings and provides a lower bound on their density. The experiments with web data demonstrate the algorithm's performance.

The paper emphasizes the importance of handling low-entropy strings in fingerprinting, as demonstrated in experiments with non-uniformly random data. It introduces robust winnowing, which reduces density, emphasizing its usefulness in specific applications.

The study examines the copying structure of 20,000 web pages, identifying a non-uniform data distribution and discussing the power law relationship between k-gram frequency and rank. MOSS, a plagiarism detection service that uses robust winnowing, efficiently detects document similarities (Schleimer *et al.*, 2003).

The paper introduces and evaluates the winnowing algorithm for document fingerprinting, addressing challenges while emphasizing practical applications such as plagiarism detection (Schleimer *et al.*, 2003).

(Sharma *et al.*, 2021) also mentioned "code clone detection". In the article, various types of machine-learning techniques for source code analysis were researched. A section about code clone detection was similar to our project. However, in the article, the emphasized approach is ML (machine learning), which differs from ours. Apart from investigating the methods for automatically detecting plagiarism, the study mentioned approaches to validate the accuracy of clones reported by existing tools. The methodology in the article entails creating a dataset of source code samples classified as non-clones or clones. Following that, feature extraction techniques are used to identify relevant features. Then, they are then input into machine-learning models for training and evaluation. The models can detect clones within sample pairs (Sharma *et al.*, 2021, p.19). It is worth noting that while our project focuses on code clone detection, our approach differs from the ML-centric methodology discussed in this study.

As mentioned (Đurić & Gasevic, 2013), contestants can modify the source code in several ways, including lexical and structural forms. Most common examples of lexical modifications could include modification of source code formatting, addition, modification, or deletion of comments, language translations, reformatting or modification of program output, etc. Lexical modifications often do not require advanced programming skills, unlike structural modification, which involves changing the order of variables in statements, changing the order of statements within code blocks, reordering code blocks, adding redundant statements or variables, modifying control structures, changing data types and modification of data structures, method inlining and method refactoring, redundancy and so on. To counteract these changes, plagiarism detection tools examine the program's structure and its lexical elements, using techniques such as tokenization, abstract syntax tree comparison, and semantic analysis to identify suspicious patterns and similarities. The system proposed in the article showed more promising results than JPlag when considering the structural and lexical modifications. While considering these structural modifications, it is essential to know the programming language that will be used so that modifications do not result in compilation errors or run-time exceptions

(Đurić & Gasevic, 2013). Since the IOI contest we are preparing for will be held in C/C++, our system was adopted to consider the use of language.

The source code similarity detection tools can be categorized into two operational modes: online and offline (Đurić and Gasevic, 2013, p. 7). Based on this categorization, our project follows the online paradigm. Every process happens in real time and with the use of a socket.

According to (Đurić and Gasevic, 2013), there are several types of plagiarism tools, such as text-based plagiarism detection, attribute-oriented plagiarism detection, and so on. Since the text-based plagiarism tools ignore the code syntax and just compare English words, it is not the most helpful approach to take when it comes to coding contests. A better approach would be using attribute-oriented similarity detection, where fundamental properties are used, such as the number of unique operands, operators, and so on. However, there are more suitable approaches than this since the same number of variables, loops, or conditional statements might be considered plagiarism. The approach best suited for these conditions would be structure-oriented similarity detection. It includes tokenization and string-matching algorithms to determine plagiarism (Đurić and Gasevic, 2013). While reviewing the existing approaches, it became evident that the approach we will use for our project is a structure-oriented similarity detection tool.

2.1. *Challenges of Similarity Detection*

Detecting plagiarism in code is challenging, and similarity rates might be excessively high. This happens due to the concise nature of simple tasks which require fewer lines of code. Hence, the contestants unintentionally produce remarkably similar code pieces. For instance, a very primitive example could be printing "Hello World" in a console in a few lines of code, which is possible, thus contributing to the prevalence of similar code structures. One potential resolution strategy for this issue involves exempting code segments with fewer than 10 to 15 lines from plagiarism detection or adjusting the plagiarism assessment based on the code's line count.

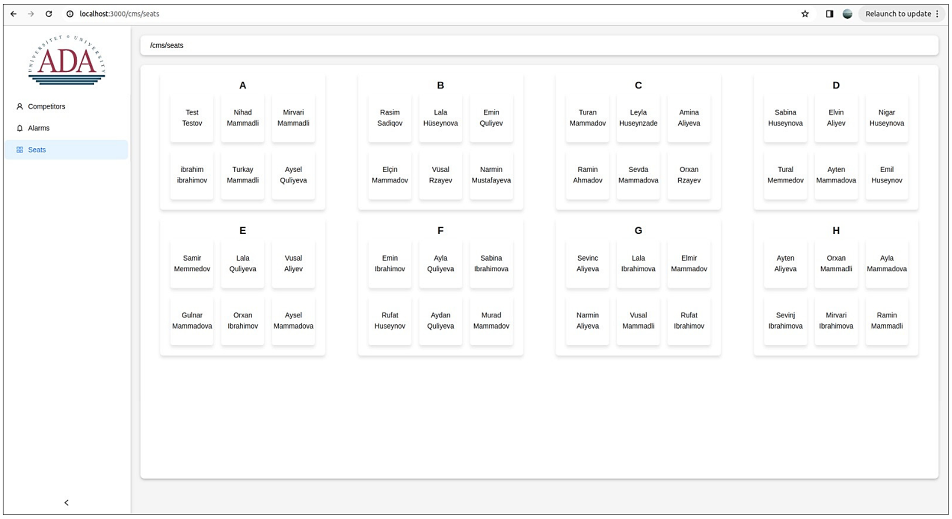
Moreover, a challenge arises in cases where contestants are required to implement functions within pre-provided code. In this particular situation, the tool will count the provided code as part of the similarity, thus artificially inflating the similarity rate. A potential mitigation strategy involves providing the plagiarism detector with a predetermined code sample in such scenarios. Subsequently, the detector tool would be configured to disregard the designated code piece when evaluating similarity, further enhancing the assessment's accuracy.

3. Design of an Alarm and Monitoring Dashboard

One of the goals of our project is a visual dashboard that provides a far friendlier user experience and more analytical possibilities. The seats will be displayed in the monitoring dashboard (Fig. 1). There will be sections from A to H, and there will be ~ six seats per section (the sections and number of seats will be adjusted to the contest requirements):

When one of the seats is clicked, the information about the contestant sitting there will appear on the screen (Fig. 2). The information includes user ID, username, submission, and similarity rate in comparison with others:

A notification will pop up on top of the screen as soon as the contestant submits. Along with this, on the Alarms page, the history log of the alarms will be kept, and the colors will be according to the type of the alarm (Fig. 3):



A: Test Testov, Nihad Mammadli, Mirvari Mammadli, ibrahim ibrahimov, Turkay Mammadli, Aysel Quliyeva

B: Rasim Sadiqov, Lala Hüseynova, Emin Quliyev, Elçin Mammadov, Vüsal Rzayev, Narmin Mustafayeva

C: Turan Mammadov, Leyla Huseynzade, Amina Aliyeva, Ramin Ahmadov, Sevda Mammadova, Orxan Rzayev

D: Sabina Huseynova, Elvin Aliyev, Nigar Huseynova, Tural Memmedov, Ayten Mammadova, Emil Huseynov

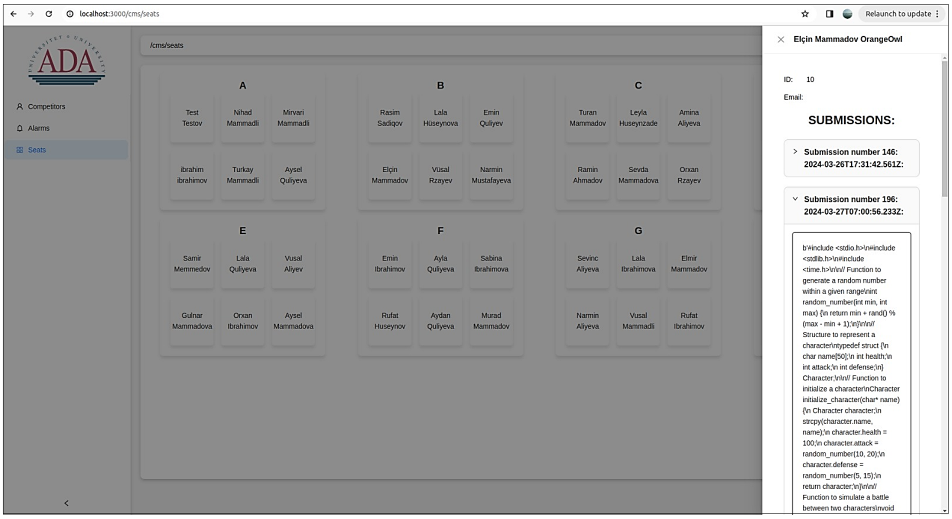
E: Samir Memmedov, Lala Quliyeva, Vusal Aliyev, Gulnar Mammadova, Orxan Ibrahimov, Aysel Mammadova

F: Emin Ibrahimov, Ayla Quliyeva, Sabina Ibrahimova, Rufat Huseynov, Aydan Quliyeva, Murad Mammadov

G: Sevinc Aliyeva, Lala Ibrahimova, Elmir Mammadov, Narmin Aliyeva, Vusal Mammadli, Rufat Ibrahimov

H: Ayten Aliyeva, Orxan Mammadli, Ayla Mammadova, Sevinj Ibrahimova, Mirvari Ibrahimova, Ramin Mammadli

Fig. 1. Seats Page (Monitoring Dashboard).



Elçin Mammadov OrangeOwl

ID: 10

Email:

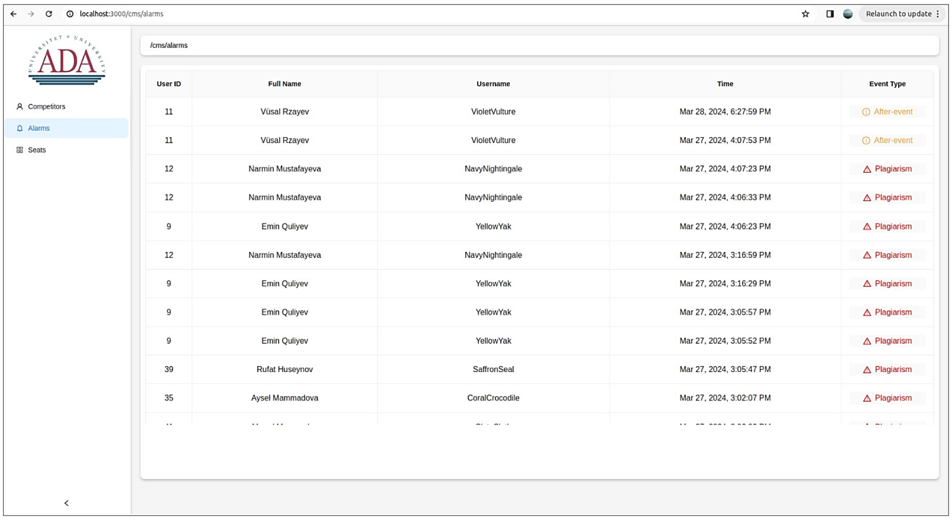
SUBMISSIONS:

Submission number 146: 2024-03-26T17:31:42.561Z:

Submission number 196: 2024-03-27T07:00:56.233Z:

```
b'#include <stdio.h>\n#include <stdlib.h>\n#include <time.h>\n\n// Function to generate a random number within a given range\nint random_number(int min, int max) {\n return min + rand() % (max - min + 1);\n}\n\n// Structure to represent a character\ntypedef struct {\n char name[50];\n int health;\n int attack;\n int defense;\n} Character;\n\n// Function to initialize a character\nCharacter initialize_character(char* name) {\n Character character;\n strcpy(character.name, name);\n character.health = 100;\n character.attack = random_number(10, 20);\n character.defense = random_number(5, 15);\n return character;\n}\n\n// Function to simulate a battle between two characters\nvoid
```

Fig. 2. Contestant information (By clicking on the seat).



User ID	Full Name	Username	Time	Event Type
11	Vüsal Rzayev	VioletVulture	Mar 28, 2024, 6:27:59 PM	After-event
11	Vüsal Rzayev	VioletVulture	Mar 27, 2024, 4:07:53 PM	After-event
12	Narmin Mustafayeva	NavyNightingale	Mar 27, 2024, 4:07:23 PM	Plagiarism
12	Narmin Mustafayeva	NavyNightingale	Mar 27, 2024, 4:06:33 PM	Plagiarism
9	Emin Quliyev	YellowYak	Mar 27, 2024, 4:06:23 PM	Plagiarism
12	Narmin Mustafayeva	NavyNightingale	Mar 27, 2024, 3:16:59 PM	Plagiarism
9	Emin Quliyev	YellowYak	Mar 27, 2024, 3:16:29 PM	Plagiarism
9	Emin Quliyev	YellowYak	Mar 27, 2024, 3:05:57 PM	Plagiarism
9	Emin Quliyev	YellowYak	Mar 27, 2024, 3:05:52 PM	Plagiarism
39	Rufat Huseynov	SaffronSeal	Mar 27, 2024, 3:05:47 PM	Plagiarism
35	Aysel Mammadova	CoralCrocodile	Mar 27, 2024, 3:02:07 PM	Plagiarism

Fig. 3. Alarms Log.

For the submission, we have several kinds of alarms:

- **Similarity alarm** – When the contestant submits their work, it goes through the process of checking for plagiarism. When the similarity rate is more than or equal to 70%, a notification on top of the page will pop up, and the seat color will blink red.
- **Scoring preceding another event** – If any kind of event (such as going to WC, print, and so on) happened within 45 minutes before the submission and the similarity rate is more than or equal to 50%, the seat will blink yellow, and the notification with similarity rates between two submissions will be seen.

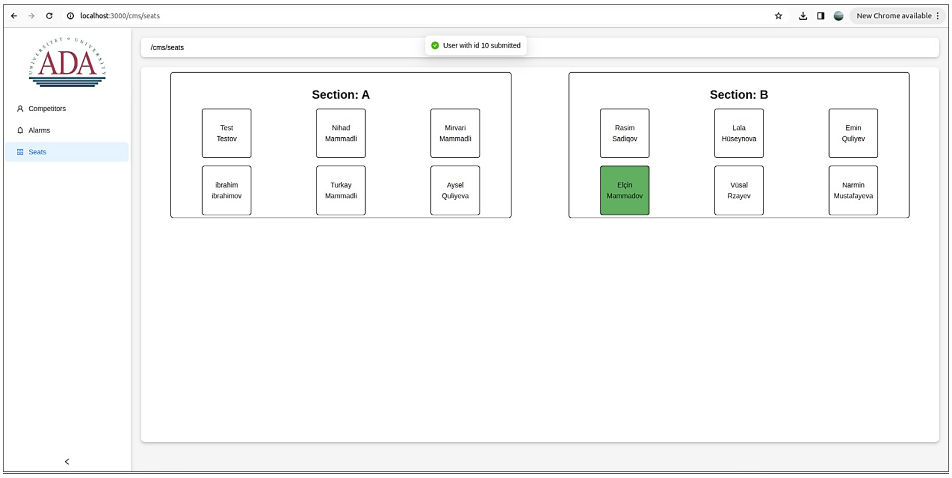


Fig. 4. Submission without any kind of possible plagiarism.

- **Sudden increase alarm –** After the submission, the contestant's similarity rate increases by 30% or more. The notification about the sudden increase will appear, and the similarity rate before and now will be shown in the message above. The color representing this alarm will be orange.

When none of the conditions are met for alarm during the submission process, the seat will blink in green, and a notification about the submission will be seen (Fig. 4):

Every single process described works with the help of a socket, which gives us real-time data.

4. Experiments

4.1. *How our Code Operates*

Our project's innovative system presents a novel approach to plagiarism detection. The remover method pre-processes code submissions by filtering out common structures and removing comments. The cppChecker method handles C++ code, demonstrating adaptability to different syntaxes. The comparer method manages real-time comparisons using async for concurrent processing. This systematic process ensures a comprehensive evaluation of multiple C++ files.

As mentioned above, the plagiarism checker employs the Longest Common Sequence algorithm. No external libraries were implemented when writing the plagiarism checker tool. Moreover, when creating the system, Node.js was chosen because of its advantages, versatility, and benefits. Also, the Express library was selected to facilitate API development. React, and AntDesign were used on the front end to create a clear and visually appealing UI (user interface).

The project is organized in a repository[1], containing the backend and the front end.

The backend includes the plagiarism checker mentioned above and the APIs.

At the front end of the web application, there are three pages: Users, Alarms, and Seats. Corresponding to those pages, there are three primary APIs: the "Sections", "Alarms", and the "Users" API. The corresponding API is called on each page, displaying the data on the screen. The Users page has been designed to showcase the relevant information retrieved from the "Users" API. The information (Names, Surnames, Usernames, and emails) is in a tabular format. All the alarms related to the submissions are displayed on the alarms page. There are three main types of alarms: plagiarism, which is shown in red; scoring preceding after-event alarm, which is shown in yellow; and sudden increase alarm, shown in orange. The alarms page derives the needed data from the "Alarms" API, which takes the data from alarms, users, and alarm_types tables. Finally, there is a Seats page that displays the seats and the configuration of the contestants. This page requires data from the "Sections" API. The "Sections" API takes data from submissions.json and the sections, users, and submissions tables.

[1] `https://github.com/NihadMammadli/SDP`

On the seat page, upon user selection, a collapsed form displays the contestant's most recent submission, revealing the code and common line similarity rate. This structured approach ensures that cases of plagiarism are clearly seen at the end of the process.

4.2. *CMS Structure*

The Content Management System (CMS) is a foundational element, providing a structured framework for managing and organizing code submissions. Its architecture facilitates efficient real-time comparisons and assessments, contributing to the overall success of the plagiarism detection tool. CMS has been used in contests such as IOI since 2017 and has become a strong nominee, becoming the established standard for the IOI (Hasanov *et al.*, 2021). Our implementation of the CMS is crucial for conducting efficient real-time comparisons in our project. It dramatically enhances the effectiveness of our plagiarism detection tool. Incorporating CMS into our project involves following a specific procedure. This included setting up and running the CMS, creating separate admin and contestant user accounts, and defining commands designed for the contestant users. As part of our workflow, we generated content tests, established test cases, and executed the code submission process using the contestant user's credentials. The data generated during this submission process is redirected to a dedicated database integrated with our Node.js code. This integration allows us to perform real-time plagiarism checks as data flows through our system.

4.3. *Our Architecture*

The architecture goes as follows (see Fig. 5):

1. Competitor makes their submission to CMS.
2. The newly submitted code gets stored in the "Submissions" table in DB (SQL Postgres).
3. Socket.js continuously checks for new submissions in real time. DB returns the submission ID to socket.js when a new submission has been made.
4. Socket.js requests the newly submitted code from Python Downloader.py.
5. Python Downloader.py requests the newly submitted code from CMS and returns the requested code to Python Downloader.py.
6. Python Downloader.py sends the code it got from CMS to the Socket.js.
7. Socket.js sends the code to Checker.js to check for plagiarism in new submissions.
8. After checking for plagiarism, it sends the alarm to the "Alarms" table and the similarity score to the "Similarities" table, and it gets stored in DB. At the same time, for quicker access, the Checker inserts submissions in JSON format to Submissions JSON.
9. The Admin requests the data (Alarms/Seats/Competitors) through React CMS, which takes it from Data.js. Data.js requests the submissions from Submissions

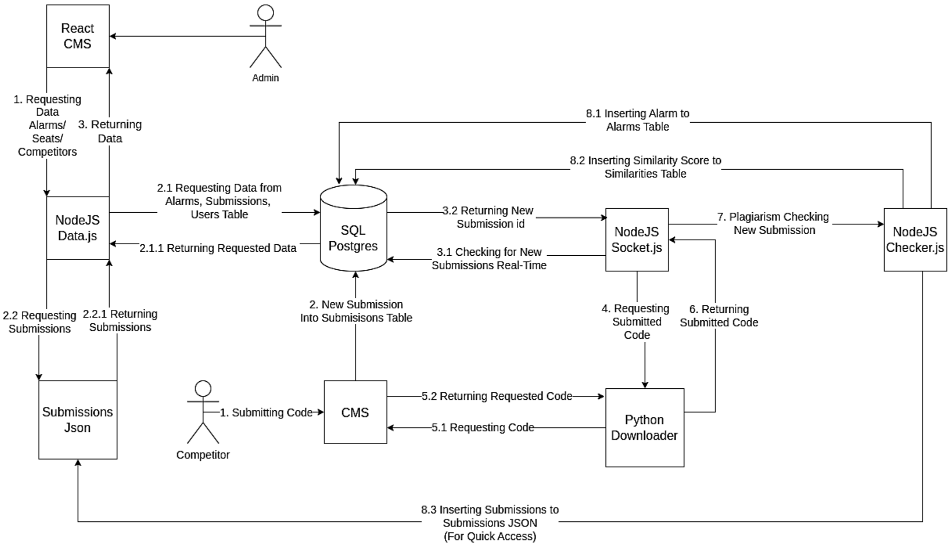

React CMS
Admin
1. Requesting Data Alarms/ Seats/ Competitors
3. Returning Data
8.1 Inserting Alarm to Alarms Table
8.2 Inserting Similarity Score to Similarities Table
2.1 Requesting Data from Alarms, Submissions, Users Table
NodeJS Data.js
SQL Postgres
3.2 Returning New Submission id
NodeJS Socket.js
7. Plagiarism Checking New Submission
NodeJS Checker.js
2.1.1 Returning Requested Data
3.1 Checking for New Submissions Real-Time
2.2 Requesting Submissions
2.2.1 Returning Submissions
2. New Submission Into Submisisons Table
4. Requesting Submitted Code
6. Returning Submitted Code
Submissions Json
1. Submitting Code
CMS
5.2 Returning Requested Code
5.1 Requesting Code
Python Downloader
Competitor
8.3 Inserting Submissions to Submissions JSON (For Quick Access)

Fig. 5. Architecture Diagram.

JSON and returns them to Node.js. Later, Node.js returns the data to React CMS, and the Admin sees the needed data.

4.4. *Our Setup*

Our team uses the most recent version of CMS in the system. We have integrated the plagiarism checker tool, visual boards to monitor submissions, and alarms to detect instances of cheating. The alarm mechanism mentioned above provides real-time alerts to identify potential cheating events of the contestants. Moreover, the logs with the alarm type and submissions are kept for further investigation of cheating cases when needed.

The upcoming IOI contest of 2024 will be held in Egypt in September. We intend to refine our system even more to ensure it is ready for use in coding competitions such as IOI. Our system for "Analysis and evaluation of the contestant's progress in real-time Coding Contests" has been tested by simulating all possible scenarios. It included identifying different kinds of alarms and automating the users' submission process. The testing aimed to replicate the potential occurrences at IOI24, demonstrating the effectiveness of our system in maintaining fairness and integrity in coding competitions.

5. Conclusion

The literature review provides a comprehensive background on plagiarism detection in programming. The increase in the usage of online platforms for contests has brought a considerable challenge regarding the integrity of the evaluation process. Our team proposed a novel system that uses sophisticated methods when analyzing the structure of codes submitted by the contestants and their behavior throughout the process to address the issue arising. It is essential to ensure fairness and authenticity in such competitions. Our system aims to provide a broad solution to plagiarism detection in competitions by using tools like the LCS (Longest Common Subsequence) algorithm and Node.js, as well as complex and sophisticated methodologies like structure-oriented similarity detection.

Through experiments and simulations, the system was validated for the accuracy of the approach used. It demonstrated the ability to detect instances of plagiarism accurately in real time. Thanks to alarms and monitoring dashboards, the system offers a framework for contest organizers to detect and analyze potential cheating incidents in real time.

We are committed to refining and enhancing our system since Egypt has an upcoming IOI 2024 competition. We hope to contribute to the coding competition society and offer a fair environment for competitive programming contestants.

The studies highlight the importance of integrity during competitions and mechanisms and technologies that ensure the quality of the competitive nature of the competitions. A robust framework will be provided through the system proposed by our team to maintain a fair and competitive environment.

References

Burrows, S., Tahaghoghi, S.M.M., Zobel, J. (2007). Efficient and effective plagiarism detection for large code repositories. *Software-Practice & Experience*, 37(2), 151–175

Đurić, Z., Gasevic, D. (2013). A Source Code Similarity System for Plagiarism Detection. *The Computer Journal*, 56(1), 70–86. `https://doi.org/10.1093/comjnl/bxs018`

Gitchell, D., Tran, N. (1999). Sim: a utility for detecting similarity in computer programs. *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education, New Orleans, Louisiana, USA, 24–28 March*, pp. 266–270. ACM New York, NY, USA.

Hage, J., Rademaker, P., van Vugt, N. (2010). A comparison of plagiarism detection tools. ISSN: 0924-3275. Retrieved from `http://www.cs.uu.nl/research/techreps/repo/CS2010/2010-015.pdf`

Hage, J., Rademaker, P., Van Vugt, N. (2011). Plagiarism detection for Java: a tool comparison. In: *Proceedings of the 1st Computer Science Education Research Conference, CSERC '11, Heerlen, The Netherlands, 7–8 April*, pp. 33–46, ACM New York, NY, USA.

Hasanov, Jamaladdin, Gadirli, Habil, Bagiyev, Aydin. (2021). On Using Real-Time and Post-Contest Data to Improve the Contest Organization, Technical/Scientific Procedures and Build an Efficient Contestant Preparation Strategy. *Olympiads in Informatics*. 23–36. DOI: 10.15388/ioi.2021.03.

Jeske, H.J., Lall, M., Kogeda, O.P. (2018). A real-time plagiarism detection tool for computer-based assessments. *Journal of Information Technology Education. Innovations in Practice*, 17, 23.
`https://jite.org/documents/Vol17/JITEv17IIPp023-035Jeske3991.pdf`

Luke, D., P.S., D., Johnson, S.L., Sreeprabha, Varghese, E.B. (2014). Software Plagiarism Detection Techniques: A Comparative Study. ISSN: 0975–9646. Retrieved from
`https://ijcsit.com/docs/Volume%205/vol5issue04/ijcsit2014050441.pdf`

Mammadli, N. (2024). SDP. GitHub. `https://github.com/NihadMammadli/SDP`

Mozgovoy, M., Frederiksson, K., White, D.R., Joy, M.S., Sutinen, E. (2005). Fast plagiarism detection system. *Lecture Notes in Computer Science, 3772/2005*, 267–270.

Myers, E.W. (1986). An O(ND) Difference Algorithm and Its Variations. *Algorithmica,* 2(1–4), pp. 251–266.
`http://www.xmailserver.org/diff2.pdf`

Prechelt, L., Malpohl, G., Philippsen, M. (2002). Finding Plagiarisms among a Set of Programs with JPlag. *Journal of Universal Computer Science,* 8(11), 1016–1038. `https://www.researchgate.net/`

`publication/2832828_Finding_Plagiarisms_among_a_Set_of_Programs_with_JPlag`

Prechelt, L., Malpohl, G., Philippsen, M. (2000, March 28). JPlag: Finding Plagiarisms among a Set of Programs. DOI: 10.3217/jucs-008-11-1016. URL: `https://www.jucs.org/jucs_8_11/finding_plagiarisms_among_a/Prechelt_L.html`

Prechelt, L., Malpohl, G., Philippsen, M. (2002). Finding Plagiarisms among a Set of Programs with JPlag. *Journal of Universal Computer Science*, 8(11), 1016–1038.

Schleimer, S., Wilkerson, D.S., Aiken, A. (2003). Winnowing: Local Algorithms for Document Fingerprinting. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, San Diego, California, USA, 9–12 June*, pp. 76–85. ACM New York, NY, USA.

Sharma, T., Kechagia, M., Georgiou, S., Tiwari, R., Sarro, F. (2021). A Survey on Machine Learning Techniques for Source Code Analysis. *ACM Transactions on Software Engineering and Methodology*, 0(0), 0–0. `https://doi.org/10.1145/nnnnnnn.nnnnnnn`

Všianský, R., Dlabolová, D., Foltýnek, T. (2017). Source Code Plagiarism Detection for PHP Language. *European Journal of Business Science and Technology*, 3(2), 106–117. DOI: 10.11118/ejobsat.v3i2.100

Wise, M.J. (1996). YAP3: Improved Detection of Similarities in Computer Programs and Other Texts. *ACM SIGCSE Bulletin*, 28(1), 130–134.

WU, Runfan & LV, Aohui & Zhao, Qiyang. (2022). Detecting Plagiarism as Out-of-distribution Samples for Large-scale Programming Contests. Olympiads in Informatics. 89–106. DOI: 10.15388/ioi.2022.08.

Yang, H., Lian, W., Wang, S., Cai, H. (2023, May). Demystifying Issues, Challenges, and Solutions for Multilingual Software Development. In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)* (pp. 1840–1852). IEEE. `https://chapering.github.io/pubs/icse23haoran.pdf`

Yetthapu, Sudheer (2023). Source Code Plagiarism Detection Using JPlag & Stack Overflow Data. *Masters Theses & Specialist Projects*. Paper 3620. `https://digitalcommons.wku.edu/theses/3620`

M. Mammadli is currently a last year student at ADA University pursuing her bachelor's degree in Information Technology. She holds the position of an IT business analyst at ERP-INTEL LLC.

N. Mammadli is a last year student in the bachelor's program of "Computer Science" offered by ADA University. Concurrently, he works as a software developer at ERP-INTEL LLC.

J. Hasanov is an Associate Professor of Computer and Information Sciences in the School of IT and Engineering at ADA University. Dr. Hasanov is mainly focused on computer vision problems medical imaging and video captioning domains. Additional to the research field, Dr. Hasanov teaches the management aspects of the IT in production and operation. Dr. Hasanov has been an ITC member for the period of 2017–2020 and led HTC during IOI 2019.