Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

«Метод поиска заимствований в программной и графической реализациях алгоритма»

Студент ИУ7-55Б _____ **А. Е. Богаченко**
(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель _____ **Ю. В. Строганов**
(Подпись, дата) (И.О. Фамилия)

2021 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-7

_____ И. В. Рудаков

«____» _____ 20____г.

З А Д А Н И Е
на выполнение научно-исследовательской работы

по теме Метод поиска заимствований в программной и графической реализациях алгоритма

Студент группы ИУ7-55Б

Богаченко Артём Евгеньевич

(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

учебная

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения НИР: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Техническое задание провести обзор в области систем управления обучением и методов поиска заимствований в программной и графической реализациях алгоритма

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на 15-25 листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
Презентация на 8-10 слайдах.

Дата выдачи задания «____» _____ 20____г.

Руководитель НИР _____ (Подпись, дата) **Ю. В. Строганов** (И.О. Фамилия)

Студент ИУ7-55Б (Группа) _____ (Подпись, дата) **А. Е. Богаченко** (И.О. Фамилия)

Содержание

ВВЕДЕНИЕ

Процесс обучения — важный этап в развитии профессиональных качеств личности. Качество методов обучения, структурированность изложения материала, а также уровень вовлечённости обучаемого напрямую влияет на предполагаемый успех и усвоение учебного материала[1]. Так же на качество обучения влияет заимствование чужих работ[2].

Для организации учебного процесса существует множество интерактивных систем управления обучением (англ. Learning Managment System (LMS[3])), которые облегчают процесс обучения и повышают вовлечённость обучаемого.

Цель работы — провести обзор существующих решений в области систем управления обучением и исследовать возможности по расширению их функционала для проверки работ на предмет заимствования и корректность выполнения поставленной задачи. Правильно выполненная работа заключается в построении блок-схемы некоторого алгоритма.

Для достижения этой цели требуется решить следующие задачи НИР:

1) провести обзор существующих решений в области систем управления обучением;
2) провести обзор существующих решений в области алгоритмов поиска заимствований.

1 Анализ предметной области

1.1 Базовые понятия

1.1.1 Алгоритм

Алгоритм — конечная последовательность детерминированных арифметических и логических действий над исходными и промежуточными данными, выполнение которых приводит к получению требуемых выходных данных[4].

1.1.2 Схема алгоритма

Схема алгоритма[4] — один из способов визуального представления алгоритмов, ориентированный граф, указывающий порядок исполнения команд алгоритма. Для описания схем существовало несколько стандартов, из которых наиболее актуальным является ГОСТ 19.701-90[5], утверждённый 01 января 1992 г. Часть описанных в стандарте символов можно увидеть на рисунке 1.

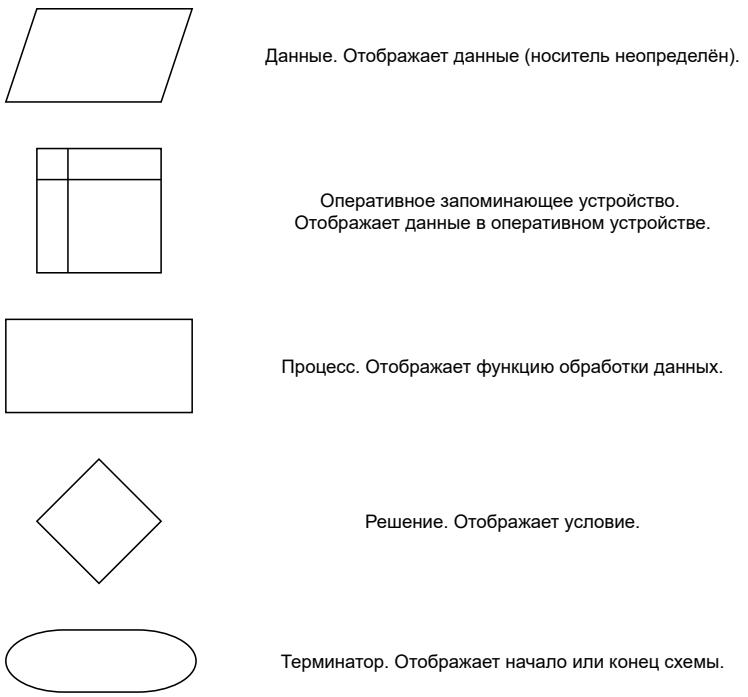

Рисунок 1 – Символы описания схемы алгоритма

Пример реализации алгоритма вычисления суммы *n* чисел представлен на рисунке 2.

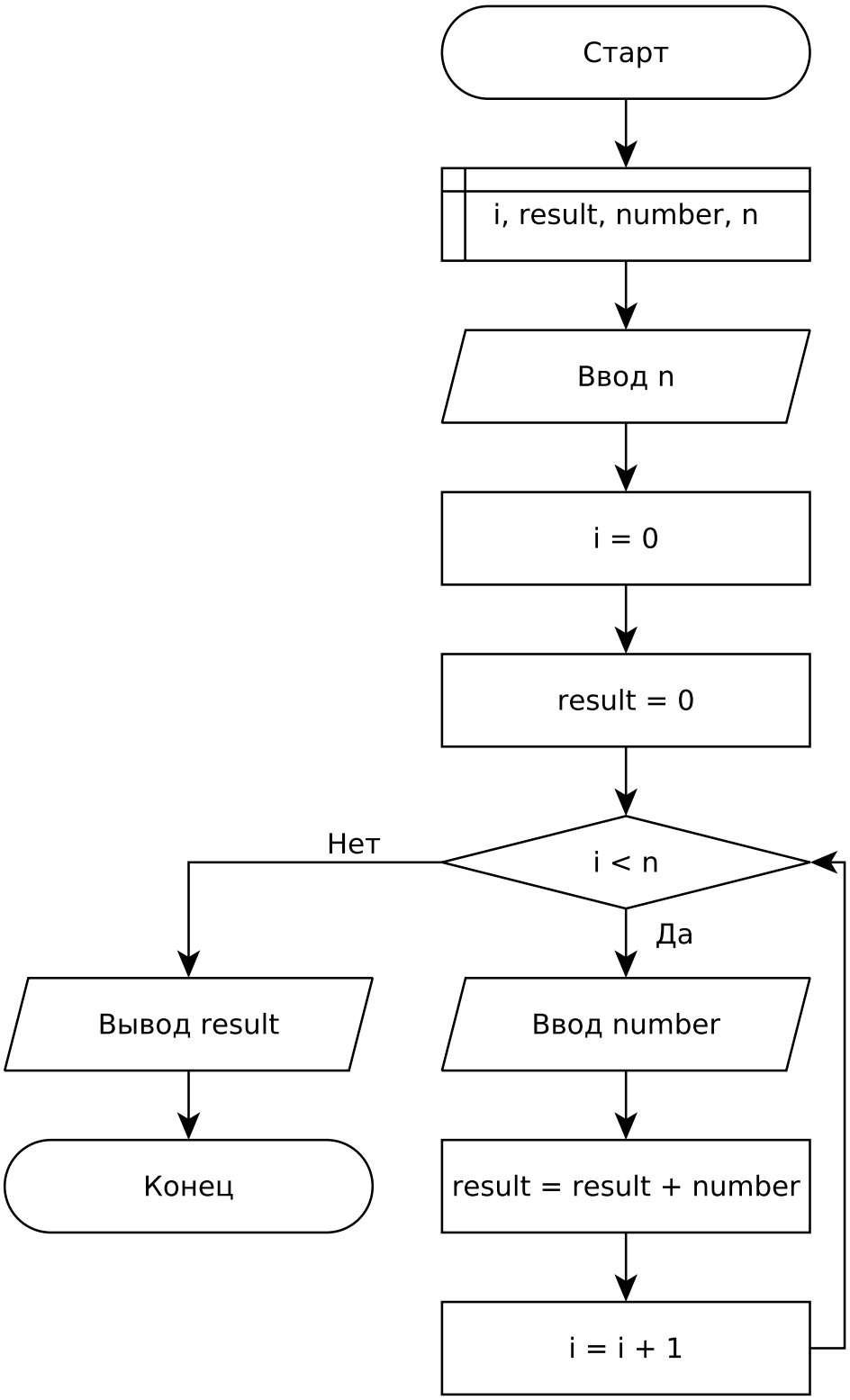

Рисунок 2 – Блок-схема алгоритма вычисления суммы *n* чисел

1.1.3 Плагиат

В общем случае плагиат — акт заимствования (использования) слов или идей другого человека без прямого упоминания первоисточника[6]. В терминах компьютерного программирования плагиатом является заимствование программной или графической (схема) реализации алгоритма.

1.1.4 Система управления обучением

Система управления обучением — веб-технология, обеспечивающая поддержку преподавателей в организации процесса обучения, организации содержания курса и поддержке студентов в процессе их обучения[7].

Пример организации интегрированной в экосистему ВУЗ-а *LMS* можно увидеть на рисунке 3.

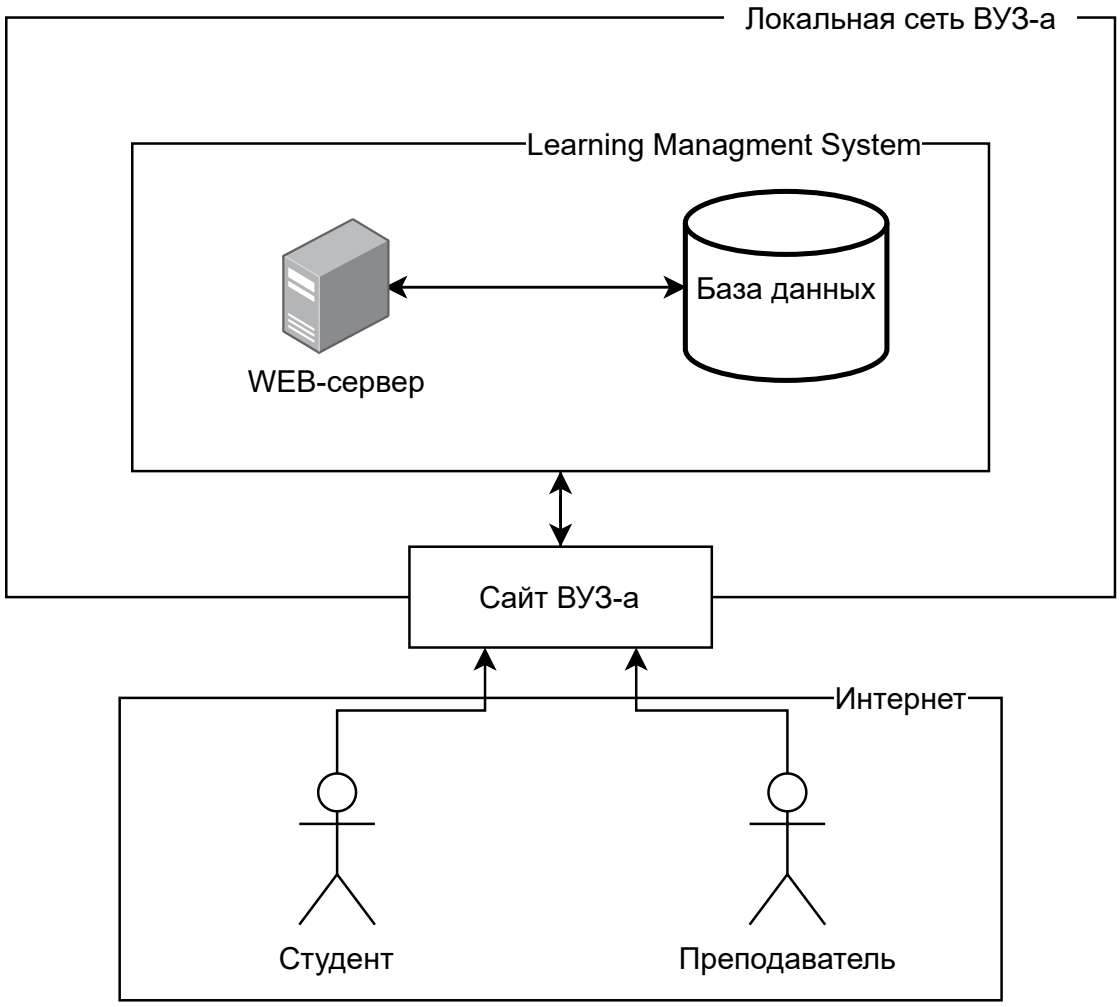

Рисунок 3 – Общая схема организации *LMS* ВУЗ-а

2 Существующие решения

2.1 Системы управления обучением

Ниже будут рассмотрены системы управления обучением в академическом домене.

2.1.1 Moodle

Moodle[8] — бесплатная система управления обучением с открытым исходным кодом. Обладает широким спектром возможностей в области организации процесса обучения.

Преимущества:

- модель с открытым исходным кодом;
- гибкость в создании учебного материала;
- простота использования;
- обширный функционал для преподавателя;
- система обратной связи для студента.

Недостатки:

- отсутствие анти-плагиат системы;
- при бесплатной системе распространения отсутствует поддержка;
- недостаточно функциональна при обширном спектре возможностей;
- конфликтует при взаимодействии со сторонними сервисами.

2.1.2 Canvas LMS

Canvas LMS[9] – система управления обучением. Включает в себя гибкий и широкий функционал не только в организации обучения, но и в коммуникации студента с преподавателем, доступна на всех платформах.

Преимущества:

- стоимость напрямую зависит от варианта использования;
- гибкость в создании учебного материала;
- кроссплатформенность;
- система поддержки продукта;

- расширенная система взаимодействия студента с преподавателем.

Недостатки:

- отсутствие анти-плагиат системы;
- перегруженный интерфейс;
- конфликтует при взаимодействии со сторонними сервисами.

2.1.3 Algorithmi

Algorithmi[10] – исключительно система по обучению программированию и построению схем алгоритмов.

Преимущества:

- анти-плагиат система исходного кода и схем;
- встроенная графическая система для создания схем;
- кроссплатформенность;
- конвертирование программного кода в схему и обратно;
- широкий спектр возможностей.

Недостатки:

- отсутствует возможность создания собственных материалов;
- перегруженный интерфейс;
- конфликтует при взаимодействии со сторонними сервисами;
- отсутствует возможность взаимодействия студента с преподавателем через платформу;
- поддержка малого количества языков программирования.

2.2 Алгоритмы поиска заимствований

Модели машинного обучения рассмотрены не будут, т.к. предварительно требуют больших объёмов размеченных данных.

2.2.1 Создание уникального отпечатка

В основе лежит процесс создания уникального отпечатка[6] исходного кода из набора метрик с последующим сравнением удалённости данного отпечатка от остальных[11]. В качестве собираемых метрик анализируются зарезервированные слова языка программирования[12] и переменные. Подсчитывается количество использований зарезервированных слов, определяется тип, если это функция учитывается порядок и количество передаваемых аргументов, анализируется необходимое количество операций ввода и вывода, определяются уникальные контексты использования зарезервированных слов, подсчитывается количество переменных. Исходный код проходит через следующие этапы:

1) реструктуризация;
2) токенизация;
3) вычисление сложности;
4) вычисление схожести.

Ниже рассмотрим каждый этап по отдельности.

Этап 1. Реструктуризация

Исходный код приводится к общему виду: убираются комментарии, лишние пробелы и производится форматирование.

Этап 2. Токенизация

На данном этапе происходит двууровневое извлечение особенностей использования зарезервированных слов языка из реструктурированного исходного кода. Алгоритм проверяет свойства синтаксиса языка, чтобы определить контекст использования инструкции. Так как список зарезервированных слов заранее известен и не может быть изменён пользователем алгоритм может отделить переменную, заданную пользователем, от инструкции языка. В контексте

использования инструкций определяется:

1) общее количество использований или вызовов (если идентификатор – функция);
2) порядок и тип передаваемых аргументов;
3) особенности использования зарезервированного слова;
4) количество операций ввода или вывода.

В общем контексте определяется количество:

1) строк;
2) символов;
3) блоков кода.

Результатом этого этапа является подробный отпечаток исходного кода.

Этап 3. Вычисление сложности

В качестве дополнительной метрики по формуле 1 вычисляется сложность исходного кода программы[13].

$$V = (N_1 + N_2)\log_2(n_1 + n_2) \tag{1}$$

где N_1 количество операторов, N_2 количество операндов, n_1 количество уникальных операторов, n_2 количество уникальных операндов.

Этап 4. Вычисление схожести

На данном этапе вычисляется процент заимствования. Пусть A и B два анализируемых исходных кода, а $f_c(A)$ и $f_c(B)$ количество особых использований оператора соответственно. Тогда соотношение совпадения может быть вычислено по формуле 2.

$$\delta_i = \frac{min(f_{ci}(A), f_{ci}(B))}{max(f_{ci}(A), f_{ci}(B))} \tag{2}$$

Используя соотношение из формулы 2, по формуле 3 можно подсчитать удалённость метрик программы A от программы B.

$$\Delta = \frac{\sum_{i}^{n} \omega_i \delta_i}{n} \quad (3)$$

где ω_i – определённый вес особенности, n – общее количество особенностей.

Результатом будет коэффициент схожести, число в диапазоне от 0 до 1. Для более точного вычисления коэффициента используется сложность, рассчитанная на предыдущем этапе. Данный алгоритм позволяет достичь точности свыше 96%.

2.2.2 Декомпозиция структуры исходного кода

Основной идеей является декомпозиция структуры исходного кода и дальнейшее его представление в виде дерева[14]. Программа состоит из трёх частей:

1) заголовки;
2) глобальные переменные;
3) функции.

Структуру в виде дерева можно увидеть на рисунке 4.

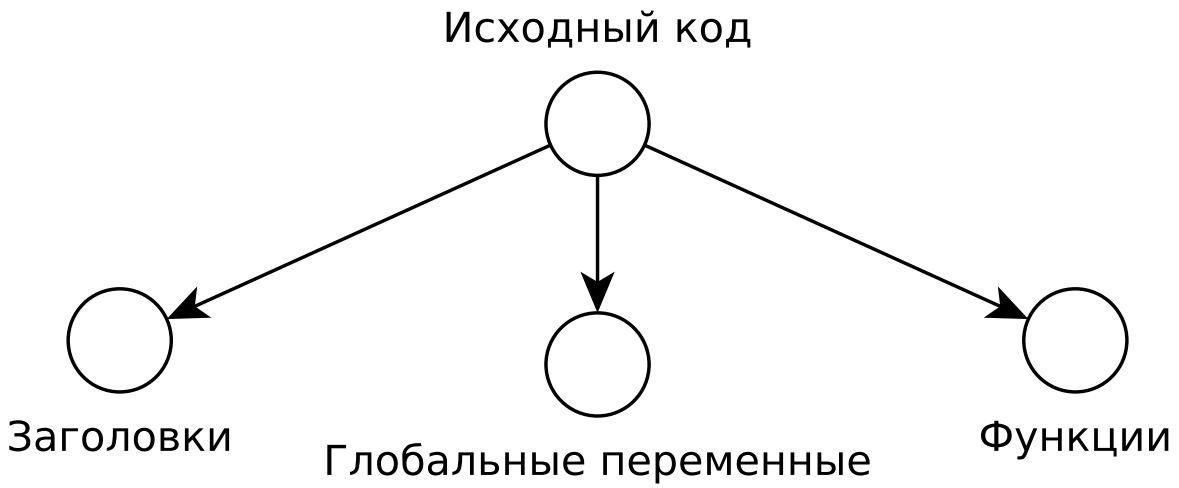

Рисунок 4 – Представление структуры исходного кода в виде дерева

Каждая часть так же представляется в виде дерева и может иметь вложенную структуру. Например, функция может содержать блок, который имеет

следующую структуру:

- локальные переменные (тоже самое, что и глобальные, но существуют и доступны только в рамках блока);
- информацию о вызываемых функциях, операциях;
- контрольный блок: может состоять из контрольного условия или содержать ещё один блок.

После построения дерева его можно преобразовать в файл формата *XML*[15] для дальнейшего анализа. Пример структуры файла можно увидеть в листинге 1.

Листинг 1: Эквивалент рисунка 4

Далее на основе узлов дерева, с помощью отображений, строятся уникальные последовательности, которые потом будут записаны в матрицу последовательностей. Так, например, для узла заголовков программы будет верно отображение из таблицы 1.

Таблица 1 – Матричное отображение узла заголовков.

Позиция	d_{i1}	...	d_{in}
Название заголовка	заголовок$_1$	...	заголовок$_n$

Аналогично для узлов глобальных переменных и функций, где каждому идентификатору присваивается свой уникальный номер. Таким образом для некоторой функции у которой тип возврата rt_2, аргументы имеют тип at_2, at_3, at_{n-k}, где каждый из типов встречается 2, 3 и $n-k$ раз соответственно, переменные lt_2, lt_7, lt_k и контрольные типы ct_1, ct_2, ct_{n-k-1} можно получить матрицу следующего вида:

тип возврата	аргументы
0 1 0 0 0 0 0 0 ... 0 0	0 2 3 0 0 0 0 0 0 0 0 0 ... 0 4
0 2 0 0 0 0 3 0 ... 0 7	1 1 0 0 0 0 0 0 0 0 0 0 ... 4 0
лок. переменные	контрольные типы

Так же на основе *XML* файла строится матрица контрольных последовательностей, где каждой контрольной инструкции функции присваивается свой уникальный номер, чтобы составить уникальную последовательность. Для данной матрицы будет верно отображение из таблицы 2

Таблица 2 – Матричное отображение контрольных последовательностей.

Контрольная инструкция	c_1	c_2	...	c_1		
Порядок следования	1	2	...	6	...	n
Номер	1	2	...	1	...	99

где значение 99 – признак того, что для конкретного номера в последовательности нет контрольной инструкции.

На последнем этапе преобразований из матрицы последовательностей и матрицы контрольных последовательностью строится интегральная матрица. По каждой строке производится операция суммирования с дальнейшей нормализацией. В конечном итоге каждой строке, которой соответствуют блоки заголовков, глобальных переменных и функций, соответствует нормализованный коэффициент схожести, который находится в диапазоне от 0 до 1. При итоговом подсчёте для каждой строки будет учитываться свой вес ω, так как, блок функций более важен в контексте определения плагиата, чем блок заголовков.

2.2.3 Анализ абстрактного синтаксического дерева

Абстрактное синтаксическое дерево (англ. Abstract Syntax Tree (AST[16])) — представление абстрактной синтаксической структуры в виде дерева, в котором внутренние вершины сопоставлены (помечены) с операторами языка про-

граммирования, а листья — с соответствующими операндами. АСД широко используются инфраструктурами компиляторов в качестве промежуточного представления (ПП) исходного кода. Ниже рассмотрим алгоритм, основанный на сравнении двух АСД, полученных с помощью утилиты *Clang*[17] фреймворка *LLVM*[18].

Так как исходный код можно представить в виде АСД, изменения можно получить используя разницу между деревьями. Такой подход перспективен потому, что точная информация о каждом изменении присутствует в АСД. Выделяют три основных типа узлов:

1) Declarations — объявления;
2) Statements — инструкции;
3) Types — типы.

Например, АСД для выражения $a = b \times c + d \div e$ может быть представлено, как на рисунке 5.

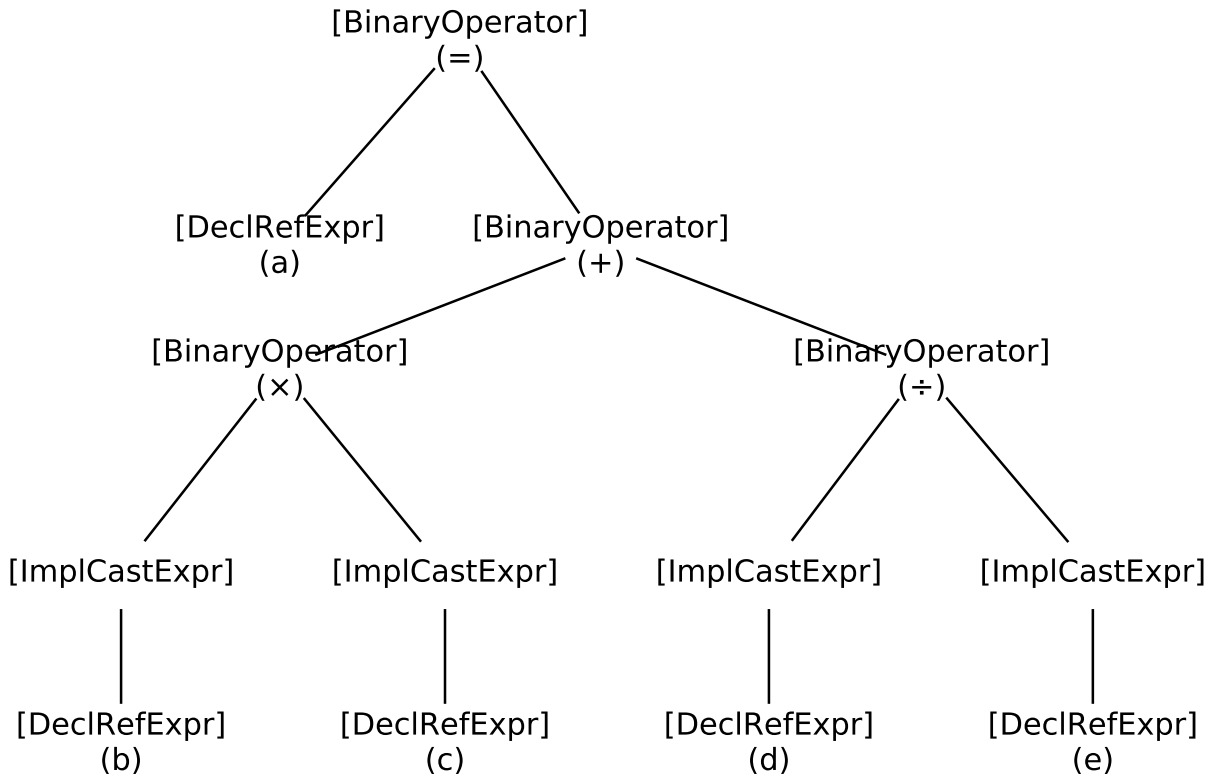

Рисунок 5 – Представление структуры исходного кода в виде дерева

Рассмотрим два алгоритма, использующих в своей основе АСД: сравнение деревьев, с помощью строковых ядер[19][20] и классификация изменений в структуре дерева[21].

Для первого метода нужно представить дерево в виде строки, таким образом, в результате прямого обхода, из дерева на рисунке 5 получим следующую строку: $\texttt{[BinaryOperator]}_1\texttt{[DeclRefExpr]}_1\texttt{[LEVEL\_UP]}_1 \ldots \texttt{[LEVEL\_UP]}_5$

Для оптимизации занимаемого пространства применяются следующие операции:

- последовательность одинаковых узлов заменяется одним с суммированием весов:

$$\texttt{[BinaryOperator]}_1\texttt{[BinaryOperator]}_1\texttt{[BinaryOperator]}_1$$
$$\downarrow$$
$$\texttt{[BinaryOperator]}_3$$

- удаляются узлы показывающие вызов функций и преобразования типов.

Количество удалённых узлов суммируется с весом потомка:

$$[\texttt{CStyleCastExpr}]_1[\texttt{CallExpr}]_1[\texttt{ImplicitCallExpr}]_1[\texttt{DeclRefExpr}]_1$$

$$\downarrow$$

$$[\texttt{DeclRefExpr}]_4$$

- удаляются узлы между [DectStmt] и [LEVEL_UP]. Количество удалённых узлов суммируется с весом родителя:

$$[\texttt{DeclStmt}]_1[\texttt{VarDecl}]_1[\texttt{DeclRefExpr}]_1[\texttt{LEVEL\_UP}]_4$$

$$\downarrow$$

$$[\texttt{DeclStmt}]_3[\texttt{LEVEL\_UP}]_4$$

- идентичные узлы суммируются между собой:

$$[\texttt{IntegerLiteral}]_2[\texttt{LEVEL\_UP}]_5$$

$$[\texttt{IntegerLiteral}]_4[\texttt{LEVEL\_UP}]_2$$

$$\downarrow$$

$$[\texttt{IntegerLiteral}]_6[\texttt{LEVEL\_UP}]_7$$

Полученные строки анализируются на вхождение идентичных блоков, при поиске вхождений учитывается полное соответствие последовательности и разница в весах частей. Таким образом можно находить процент различия отдельно взятых последовательностей. Результатом является нормализованный коэффициент схожести, который находится в диапазоне от 0 до 1.

Для второго метода можно сказать, что из одного дерева, можно получить другое путём применения некоторых операций. Определим четыре действия[21]:

1) вставка узла в дерево: $INS((l, v), y, k)$, где l - название узла, v - его значение, k - порядковый номер потомка, y - узел родителя;
2) удаление узла из дерева: $DEL(x)$, где x потомок некоторого узла $p(x)$;
3) перемещение узла: $MOV(x, y, k)$, где x - перемещаемый узел, k - новый порядковый номер потомка, y - новый родитель x;
4) обновление (изменение) узла: $UPD(x, v)$, где x - обновляемый (изменяемый) узел, v - новое значение при условии $v_{old} \neq v_{new}$.

Каждое действие имеет своё определённое влияние в контексте использо-

вания, например, изменение условия в условном блоке оказывает больше влияния, чем изменение названия переменной. Удаление какой-либо функциональности является ”критическим” изменением.

Текущая реализация — *ChangeDistiller*[22]. Алгоритм строит ещё одно дерево на основе АСД, обрабатывает его находя разницу между узлами и индексирует возможный список операций, применённый к дереву. Различия между узлами находятся с помощью редакционного расстояния[23]. Данный подход является менее эффективным, в сравнении с предыдущим, т.к. преимущественно точен для объектно-ориентированных языков и требует дополнительных вычислений.

ЗАКЛЮЧЕНИЕ

Готовые решения в области систем управления обучением, зачастую являясь системами общего назначения, предоставляют ограниченный набор функционала без возможности его расширения. Некоторые системы предоставляются ”как есть” даже без возможности обращения в поддержку. Сравнение рассмотренных систем управления обучением можно увидеть в таблице 3. Обозначения:

- Б — бесплатное распространение;
- П — платное распространение;
- Ч — частично удовлетворяет критерию.

Таблица 3 – Анализ систем управления обучением.

Критерий	**Moodle**	**Canvas**	**Algorithmi**
распространение	Б/П	Б/П	Б
поддержка	П	П	-
возможность расширения	Ч	-	-
интеграция сторонних систем	Ч	-	-
антиплагиат	-	-	+
создание учебного материала	+	+	-
организация курса	+	+	-
кроссплатформенность	Ч	+	Ч

В отличие от алгоритмов поиска заимствования в литературе, выбор подхода для определения заимствований в программировании является очень существенным [14]. Программная или графическая реализация может претерпевать до шести уровней модификаций, чтобы обойти антиплагиат. Ознакомиться с предполагаемыми уровнями модификаций можно на рисунке 6.

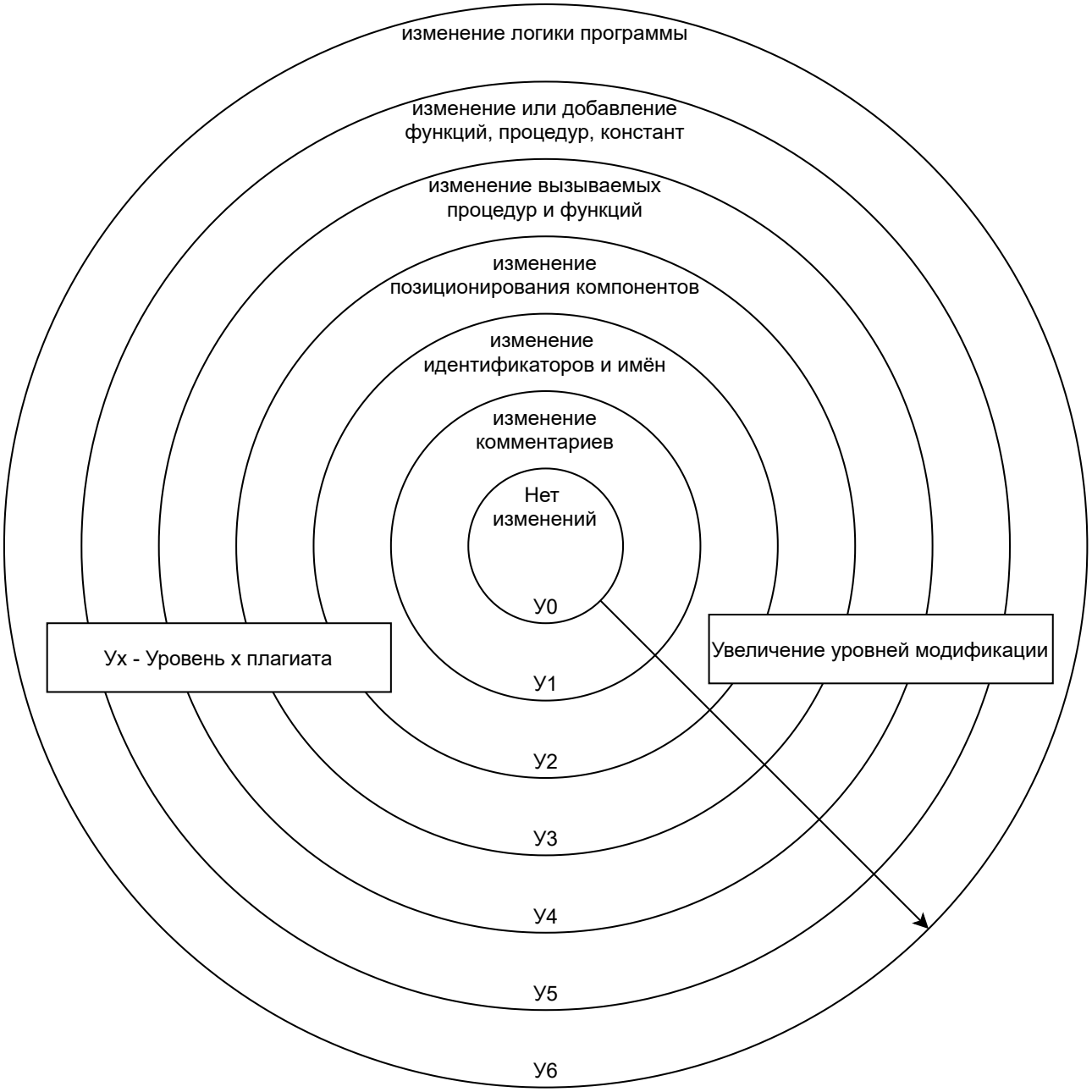

Рисунок 6 – Спектр модификаций

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Jeyaraj John. Effective Learning and Quality Teaching // SSRN Electronic Journal. 2019. 09. Т. 57. С. 30–35.

2. Chankova Mariya. Dealing with Students' Plagiarism Pre-Emptively Through Teaching Proper Information Exploitation // International Journal for the Scholarship of Teaching and Learning. 2017. 07. Т. 11.

3. Learning Management System (LMS) [Электронный ресурс]. Режим доступа: https://www.hse.ru/en/studyspravka/lms_student/, свободный (дата обращения: 27 января 2022 г.).

4. Горностаева Т.Н. // Алгоритмы. Мир науки, 2019. с. 10.

5. ГОСТ 19.701-90 [Электронный ресурс]. Режим доступа: https://docs.cntd.ru/document/9041994, свободный (дата обращения: 27 января 2022 г.).

6. Merriam-Webstem Thesaurus. Режим доступа: https://www.merriam-webster.com/dictionary, свободный (дата обращения: 27 января 2022 г.).

7. Learning Management System (LMS) success: An investigation among the university students / Seyed Jafari, Suha Salem, Mohaddece Moaddab [и др.]. 2015. 08. С. 64–69.

8. Moodle – open-source learning platform. Режим доступа: https://www.moodle.org, свободный (дата обращения: 27 января 2022 г.).

9. Canvas LMS. Режим доступа: https://www.instructure.com/canvas, свободный (дата обращения: 27 января 2022 г.).

10. Algorithmi. Режим доступа: http://algorithmi.ipt.pt/#more, свободный (дата обращения: 27 января 2022 г.).

11. Narayanan Sandhya, Surendran Simi. Source code plagiarism detection and performance analysis using fingerprint based distance measure method. 2012. 07. С. 1065–1068.

12. C reserved keyword | IBM documentation. Режим доступа: https://www.ibm.com/docs/en/wdfrhcw/1.4.0?topic=programs-c-reserved-keywords, свободный (дата обращения: 27 января 2022 г.).

13. Halstead M. H. Natural Laws Controlling Algorithm Structure // SIGPLAN Not. New York, NY, USA, 1972. feb. Т. 7, № 2. с. 19–26.

14. Noh Seo-Young, Gadia Shashi. An xml plagiarism detection model for procedural programming languages. 2022. 01.

15. Extensible Markup Language. Режим доступа: https://www.w3.org/XML/, свободный (дата обращения: 27 января 2022 г.).

16. Miller F.P., Vandome A.F., John M.B. Abstract Syntax Tree. VDM Publishing, 2010.

17. Clang C Language Family Frontend for LLVM. Режим доступа: https://clang.llvm.org/index.html, свободный (дата обращения: 27 января 2022 г.).

18. The LLVM Compiler Infrastructure Project. Режим доступа: https://llvm.org/, свободный (дата обращения: 27 января 2022 г.).

19. A novel string representation and kernel function for the comparison of I/O access patterns / Raul Torres, Julian M. Kunkel, Manuel F. Dolz [и др.] // Parallel Computing Technologies: 14th International Conference, PaCT 2017, Nizhny

Novgorod, Russia, September 4-8, 2017, Proceedings / под ред. V. Malyshkin. Springer, 2017. Т. 10421 из Lecture Notes in Computer Science. С. 500–512.

20. Comparison of Clang Abstract Syntax Trees using String Kernels / Raul Torres, Thomas Ludwig, Julian M. Kunkel [и др.] // 2018 International Conference on High Performance Computing Simulation (HPCS). 2018. С. 106–113.

21. Fluri B., Gall H. C. Classifying Change Types for Qualifying Change Couplings // 14th IEEE International Conference on Program Comprehension. Los Alamitos, CA, USA: IEEE Computer Society, 2006. jun. С. 35–45.

22. ChangeDistiller tool. Режим доступа: `https://bitbucket.org/sealuzh/tools-changedistiller/wiki/Home`, свободный (дата обращения: 27 января 2022 г.).

23. Левенштейн В. И. Двоичные коды с исправлением выпадений, вставок и замещений символов. М.: Доклады АН СССР, 1965. Т. 163. С. 845–848.