

Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б11-мм

Улучшение кода алгоритма по поиску
условных функциональных зависимостей в
“Desbordante”

Выродов Михаил Владимирович

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
асс. кафедры ИАС Г. А. Чернышев

Санкт-Петербург
2023

Оглавление

Введение	3
1. Постановка задачи	5
2. Обзор	6
2.1. Обзор предметной области	6
2.2. Анализ изначального кода	8
3. Метод	10
3.1. Исправления архитектуры кода	10
3.2. Исправления других ошибок	11
3.3. UML диаграммы	11
4. Эксперимент	14
Заключение	18
Список литературы	19

Введение

Практически все ИТ-компании в настоящее время хранят у себя табличные данные. Эти данные зачастую требуют глубокого анализа для оптимизации работы с базой данных и для её очистки от дубликатов и повреждений. Одной из частей такого анализа является выявление зависимостей между различными данными в таблице.

Закономерности между атрибутами таблицы могут быть описаны разными способами. Мы будем описывать их с помощью примитивов. Примитив представляет собой описание правил, применяемых к определенной части данных с использованием математических методов. Такой способ описания позволяет задать отношение максимально точно и ясно. Примером такого примитива являются функциональные зависимости, которые могут быть описаны как бинарные отношения на множествах столбцов таблицы.

Примитивы могут быть автоматически обнаружены с помощью различных алгоритмов. Они могут быть полезными для аналитиков и ученых, предоставляя им возможность лучше понять взаимосвязь между данными, с которыми они работают, и вдохновить их на новые эксперименты.

Инструмент для профилирования данных Desbordante¹ создан для популяризации использования примитивов и предоставления людям возможности исследовать свои данные с их помощью. В проекте установлено много стандартов разработки, которых должны придерживаться программисты, развивающие этот инструмент. Из них можно выделить несколько основных стандартов — следование Google C++ style guide², поддержка линейной, читаемой истории коммитов на github и написание кода в соответствии со стандартом C++17.

Из множества примитивов можно выделить условные функциональные зависимости (CFD). Это функциональные зависимости, которые удовлетворяют некоторому шаблону данных (условию) и которые могут

¹<https://github.com/Mstrutov/Desbordante>

²<https://google.github.io/styleguide/cppguide.html>

иметь определённую неточность (некоторое количество ошибок). Такого рода закономерности позволяют видеть зависимости на конкретном подмножестве таблицы, что может быть крайне полезно.

В результате прошлой учебной практики мною был интегрирован в Desbordante код алгоритма FD-First-DFS по поиску CFD [4]. Однако, код этого алгоритма был интегрирован не в основную ветку репозитория проекта, так как он не подходил под некоторые стандарты разработки в Desbordante. Моя задача заключается в улучшении и модернизации этого кода, чтобы он удовлетворял всем стандартам разработки в Desbordante и был интегрирован в основную ветку репозитория проекта.

1. Постановка задачи

Целью работы является улучшение интегрированного в edbt ветку Desbordante кода алгоритма по майнингу CFD [4], описанного в статье [2], до такой степени, чтобы он был интегрирован в основную ветвь репозитория проекта. Для её выполнения были поставлены следующие задачи:

1. Выделить виды ошибок и недочётов, встречающихся в коде и его архитектуре;
2. Исправить эти ошибки;
3. Сравнить качество изменённого кода и изначального кода.

2. Обзор

2.1. Обзор предметной области

Условные функциональные зависимости (CFD) являются важным понятием в области баз данных и теории зависимостей. Они определяют отношение между атрибутами внутри таблицы, где один атрибут функционально зависит от других, при условии, что определенное условие истинно.

Формально, условная функциональная зависимость записывается в виде: $A \rightarrow B | C$, что означает, что атрибуты в A функционально определяют атрибуты в B , при условии того, что C истинно. Здесь A , B представляют собой множества атрибутов, а C — это условие, заданное на некотором множестве атрибутов.

Пример 1. Рассмотрим таблицу “Заказы”, содержащую следующие атрибуты: Customer ID (идентификатор клиента), Product ID (идентификатор продукта), Quantity (количество продукта) и Status (статус заказа).

В этом примере выполняется CFD, заданная следующим образом:

$\{Customer\ ID, Product\ ID, Status\} \rightarrow \{Quantity\} | \{Status = “Выполнен”\}$.

Это означает, что количество продукта (Quantity) функционально зависит от идентификатора клиента (Customer ID) и идентификатора продукта (Product ID), при условии, что статус заказа (Status) равен “Выполнен”.

То есть, если мы знаем идентификатор клиента и идентификатор продукта, и при этом статус заказа указан как “Выполнен”, то мы можем однозначно определить количество заказанного продукта.

И правда, если мы рассмотрим строки таблицы, в которых статус заказа — “Выполнен”, то для каждой пары таких строк, если у них равны значения атрибутов Customer ID и Product ID, то значения атрибута Quantity у них тоже совпадут.

Таблица 1: Датасет для примера 1. “Заказы”

Customer ID	Product ID	Quantity	Status
1	5	20	Выполнен
2	3	45	Выполнен
2	3	45	Выполнен
2	3	28	Не выполнен
2	3	45	Не выполнен
1	5	20	Выполнен
1	2	76	Выполнен
1	5	3	Не выполнен

CFD используются для более точного определения связей между атрибутами в базе данных. Они позволяют представить более сложные отношения, где зависимость между атрибутами зависит от определенных условий.

CFD играют важную роль при проектировании и оптимизации баз данных. Они помогают улучшить её структуру, очистить её от повреждённых данных, устраниТЬ избыточность данных, повысить эффективность выполнения запросов. CFD используются аналитиками данных, разработчиками баз данных и другими специалистами, работающими с базами данных.

Существует множество алгоритмов по поиску CFD. Одним из самых популярных алгоритмов по майнингу CFD является алгоритм CTANE. Он существует более 10 лет и построен на алгоритме TANE по поиску функциональных зависимостей. Этот алгоритм уже реализован в Desbordante.

Однако, в 2019 году в статье [2] было представлено три новых алгоритма по поиску CFD и было экспериментально доказано, что по крайней мере два из них в общем случае работают быстрее, чем CTANE. В ходе выполнения прошлой курсовой работы для интеграции в проект Desbordante мною был выбран алгоритм Fd-First-DFS из статьи [2], так как в ней было экспериментально показано, что в большинстве случаев он работает быстрее других алгоритмов, представленных в этой статье.

2.2. Анализ изначального кода

Будем называть ту версию кода алгоритма FD-First-DFS, которая была интегрирована в ветку edbt репозитория Desbordante [4] в результате моей прошлой курсовой работы, изначальным кодом. Код алгоритма FD-First-DFS, который по результатам этой курсовой работы был интегрирован в ветку main репозитория Desbordante [3], будем называть интегрированным или же новым кодом.

У изначального кода архитектура является плохо подходящей под потенциальное добавление новых алгоритмов по поиску CFD в проект. В нём есть только один класс — `CFDDiscovery`, который хранит и логику алгоритма FD-First-DFS, и логику работы с табличными данными, и методы для получения параметров, нужных алгоритмам по поиску CFD, извне. Таким образом, если в последствии добавлять новые алгоритмы по поиску CFD в Desbordante, то их придётся добавлять в класс `CFDDiscovery`, т.к. только в нём содержится необходимая такими алгоритмами логика работы с табличными данными и методы для получения параметров, нужных всем таким алгоритмам. Из-за этого при добавлении новых алгоритмов по поиску CFD в Desbordante, класс `CFDDiscovery` быстро разрастётся до огромных размеров и будет трудно читаемым.

Кроме этого, класс `CFDDiscovery` в изначальном коде не является классом с единственной ответственностью. В нём также находятся методы, которые оперируют структурами, не определёнными в классе `CFDDiscovery`, при этом не используя ни одно поле или другой метод этого класса. Примерами таких методов являются `isConstRulePartition(...)`, `getPartitionSupport(...)` и `getPartitionError(...)`.

В коде были методы, длиной более 100 строк и в них код достигал седьмого уровня вложенности. Кроме того, в некоторых из таких методов были фрагменты полностью повторяющегося кода, которые могли бы быть выделены в отдельные функции. В качестве примера можно привести методы `MinePatternsBFS(...)` и `MinePatternsDFS(...)` в классе `CFDDiscovery`. Размер `MinePatternsBFS(...)` — ≈ 110 строк, и в

обоих этих методах есть одинаковые фрагменты, которые не выделены в отдельные функции.

Кроме того, в изначальном коде присутствовало множество ошибок, не связанных с архитектурой кода. Было выявлено, что он содержал реализацию функций, уже написанных в стандартной библиотеке языка C++ или в библиотеке boost, поэтому такие фрагменты были заменены на аналогичные функции из библиотеки boost или из стандартной библиотеки. Одним из таких примеров является реализация шаблонной функции хэша от пары объектов (*pairhash*) и использование этой функции в *unordered_map*. Хотя при помощи библиотеки boost можно использовать *unordered_map* от пары объектов, не реализуя самостоятельно функцию хэша от пары.

В изначальном коде было множество неявных преобразований типов. Чаще всего эти преобразования велись между разными целочисленными типами в C++. Такие преобразования могут привести к переполнению и следовательно к неправильной логике работы алгоритма.

3. Метод

В прошлом разделе были перечислены выявленные ошибки в изначальном коде алгоритма FD-First-DFS. Для их исправления, мною были изменены как архитектура кода, так и отдельные функции в нём.

3.1. Исправления архитектуры кода

В этом списке перечислены исправления проблем изначального кода, связанных с его архитектурой и выявленных в прошлом разделе.

1. Чтобы исправить то, что код не являлся легко расширяемым, класс `CFDDiscovery` был разбит на два класса — `CFDDiscovery` и `FDFirstAlgorithm`. Класс `CFDDiscovery` теперь является абстрактным классом, от которого должны наследоваться все классы алгоритмов по майнингу CFD. В нём описывается основной интерфейс, который должен поддерживать каждый такой алгоритм, последовательность действий его работы и методы по получению таблицы, в которой будет проходить поиск CFD, и параметров, нужных всем алгоритмам по майнингу CFD. Класс `FDFirstAlgorithm` теперь наследуется от класса `CFDDiscovery` и содержит всю логику, которая относится к алгоритму FD-First-DFS;
2. Ещё одной проблемой архитектуры кода было несоблюдение принципа единственной ответственности в классе `CFDDiscovery`. В нём находились методы, которые не использовали поля или другие методы класса `CFDDiscovery`, но использовались в других методах этого класса. Такие функции были выделены в отдельный класс-утилиту `PartitionUtil` и были сделаны статическими, т.к. они не задействовали ни одно поле или метод нестатического класса;
3. Последней выявленной проблемой архитектуры изначального кода являлось наличие в классе `CFDDiscovery` методов, которые были большими и при этом могли бы быть разделены на более мелкие, более понятные методы. В качестве исправления этой ошибки

ки мною в класс `FDFirstDFS` были введены шесть новых методов, которые выполняют определённые подзадачи методов в изначальном коде класса `CFDDiscovery`, а в класс `CFDRelationData` было введено два аналогичных метода.

- В классе `FDFirstAlgorithm` два новых метода используются в методе `FDFirstDFS(...)`, и четыре новых метода используются и в `MinePatternsDFS(...)`, и в `MinePatternsBFS(...)`;
- В классе `CFDRelationData` было выделено два новых метода, которые используются в разных перегрузках метода `CreateFrom(...)`.

3.2. Исправления других ошибок

В этом списке перечислены исправления проблем изначального кода, выявленные в прошлом разделе и не связанные с его архитектурой.

1. В изначальном коде содержались реализации функций, уже написанных в сторонних или в стандартной библиотеке C++. Такие фрагменты были заменены на функции из библиотеки `boost`³ и из стандартной библиотеки, которые выполняют те же задачи. Всего было заменено 10 таких кусков кода;
2. Неявные преобразования типов были заменены на явные, и преобразования численных типов в стиле языка С были заменены преобразованиями типов в стиле языка C++ с помощью `static_cast`;
3. Изначальный код был изменен так, чтобы он следовал Google C++ style guide.

3.3. UML диаграммы

Далее приведены две UML диаграммы кода алгоритма FD-FirstDFS. На Рисунке 1 изображена диаграмма классов изначального кода [4], а на Рисунке 2 приведена диаграмма классов нового кода [3].

³<https://www.boost.org/>

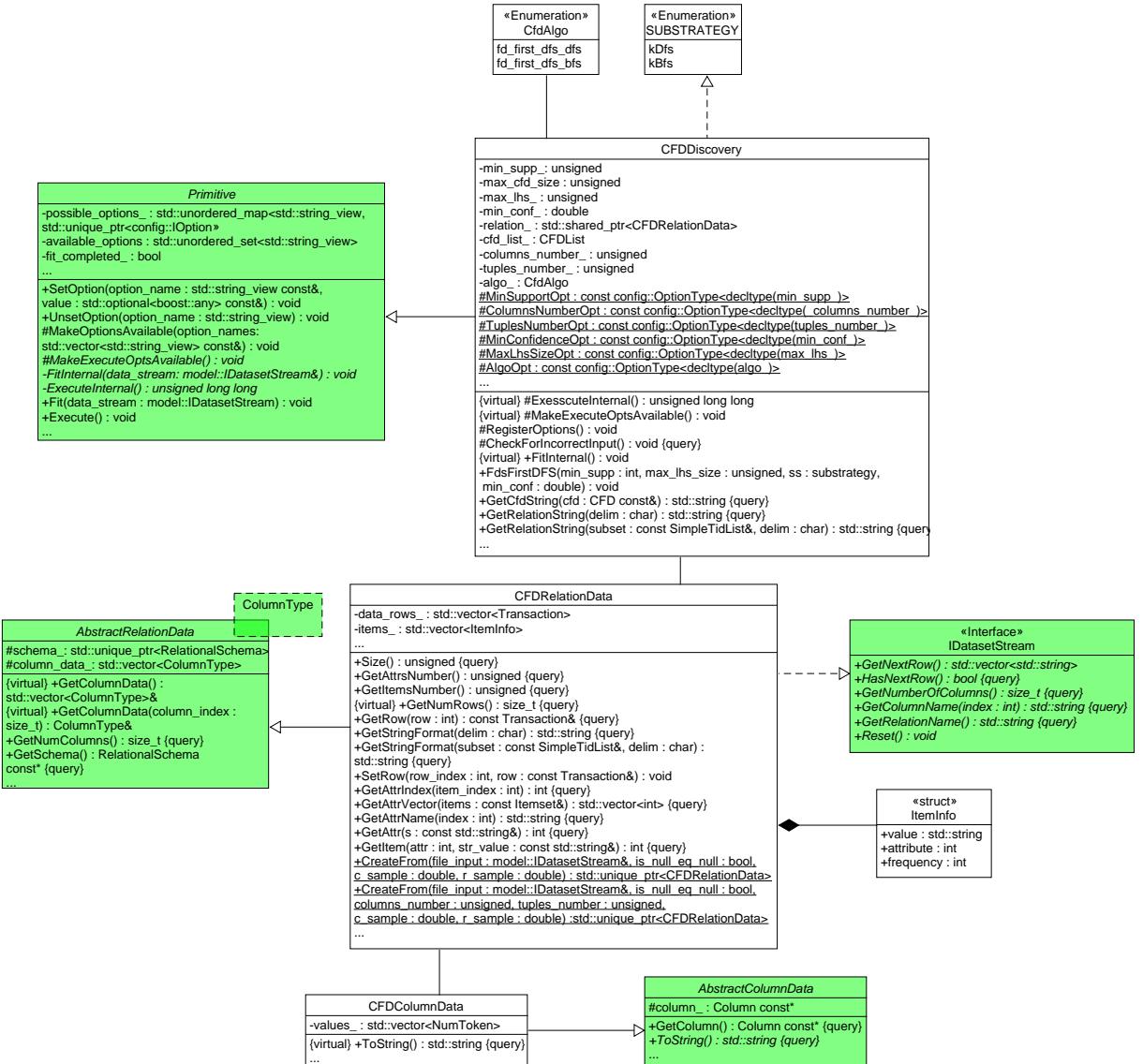


Рис. 1: Диаграмма классов изначального кода алгоритма [4]

- Используемый класс из Desbordante без изменений

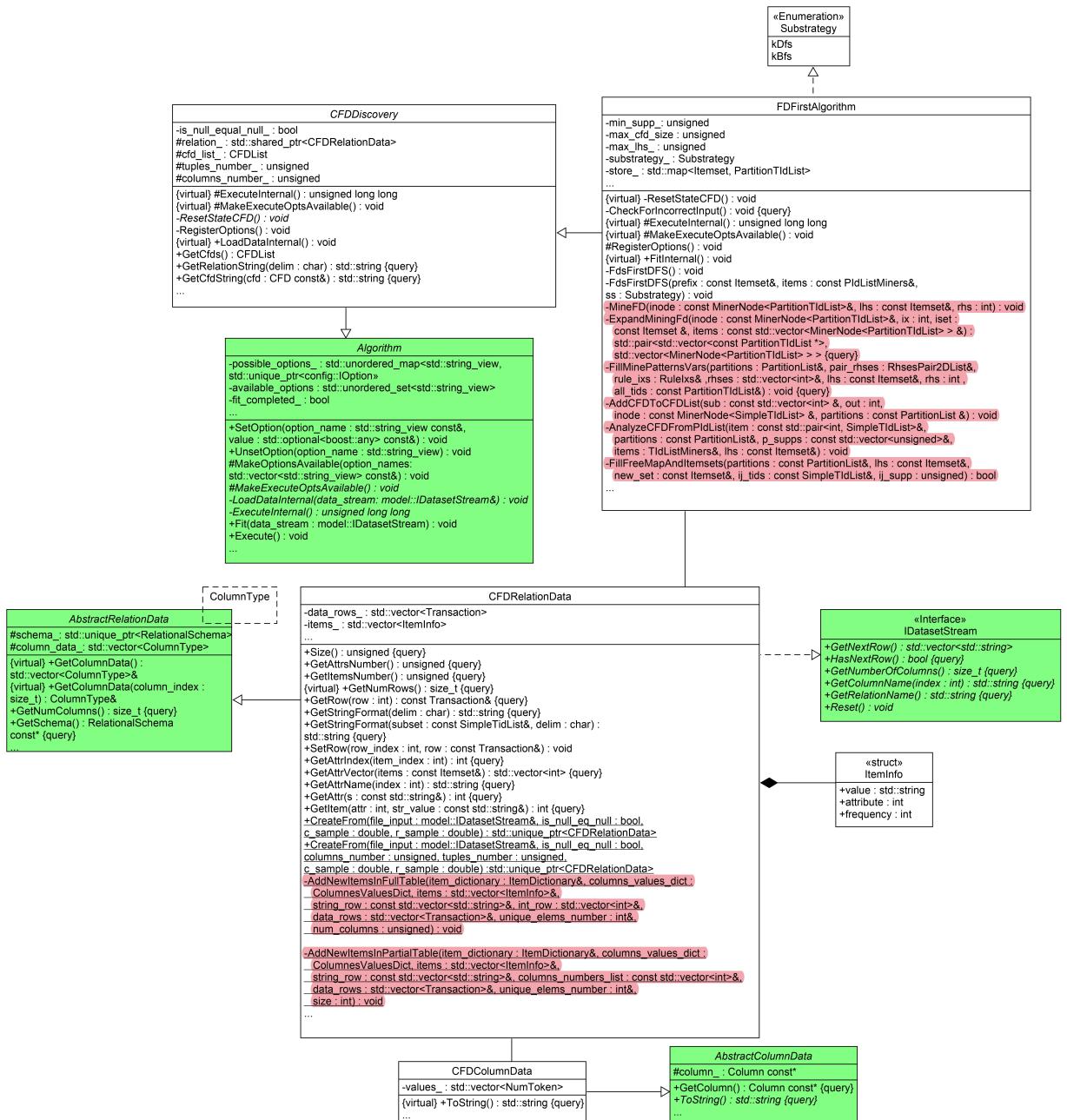


Рис. 2: Диаграмма классов интегрированного кода алгоритма [3]

- Новый метод, созданный с целью повышения читаемости кода
- Используемый класс из Desbordante без изменений

4. Эксперимент

Далее в этом разделе под изначальным кодом подразумевается код из оригинального репозитория [1], сделанный для статьи [2]. Под старым интегрированным кодом подразумевается код, который по результатам моей прошлой курсовой работы был интегрирован в ветку edbt репозитория Desbordante [4]. Под новым интегрированным кодом подразумевается код, который по результатам данной курсовой работы был интегрирован в основную ветку репозитория Desbordante [3].

Для сравнения качества кода до и после изменений было решено воспользоваться статическими анализаторами кода, так как в большинстве случаев они легки в использовании и предоставляют оценку качества кода по большому числу различных метрик.

Было принято решение воспользоваться анализаторами PVS-Studio⁴ и CppLint⁵, так как ими можно пользоваться бесплатно в учебных целях и они в совокупности позволяют глубоко проанализировать код на наличие различных ошибок, которые потенциально могут привести к неправильному поведению алгоритма. Ещё с их помощью можно проверить код на несоответствие Google C++ style guide и на наличие устаревших фрагментов кода, не соответствующих стандарту языка C++17.

CppLint предназначен для проверки, насколько хорошо в коде соблюдаются Google C++ style guide. Анализатор PVS-Studio предназначен для выявления опечаток, мёртвого кода, потенциальных уязвимостей (Static Application Security Testing, SAST). Кроме того, PVS-Studio отображает предупреждения на Common Weakness Enumeration, SEI CERT Coding Standards, а также поддерживает стандарт MISRA.

Результаты статического анализа в Таблице 3 и в Таблице 4 являются неполными. В эти таблицы были отобраны только те ошибки, которые появляются в изначальном коде чаще всего.

⁴<https://pvs-studio.com/en/>

⁵<https://github.com/cpplint/cpplint>

Изначальный код алгоритмов по поиску условных функциональных зависимостей [1], написанный по статье [2], включает в себя реализацию трёх алгоритмов по майнингу CFD, из которых мною был интегрирован только самый эффективный из них — FD-First-DFS, поэтому изначальный и интегрированный коды отличаются в размерах.

Таблица 2: Сравнение размеров изначального, интегрированного по результатам прошлой курсовой работы и интегрированного по результатам данной курсовой работы кодов алгоритма

Код	Количество файлов	Количество строк	Количество классов
Изначальный	22	\approx 4000	9
Интегрированный старый	24	\approx 2900	11
Интегрированный новый	22	\approx 2000	12

Таблица 3: Сравнение изначального, интегрированного по результатам прошлой курсовой работы и интегрированного по результатам данной курсовой работы кодов алгоритма по метрикам анализатора PVS-Studio

Название ошибки	Кол-во появлений в изначальном алгоритме	Кол-во появлений в старом интегрированном алгоритме	Кол-во появлений в новом интегрированном алгоритме
Possible error of indexing large arrays.	168	134	81
The body of a loop or conditional statement should be enclosed in braces.	120	62	30
The global namespace should only contain 'main', namespace declarations and 'extern "C"' declarations.	71	87	10
Decreased performance. Object may be created in-place in a container.	20	5	0
Operands of the logical 'and' or the 'or' operators, the '!' operator should have 'bool' type	12	4	0
Unary minus operator should not be applied to an expression of the unsigned type	9	0	0

Таблица 4: Сравнение изначального, интегрированного по результатам прошлой курсовой работы и интегрированного по результатам данной курсовой работы кодов алгоритма по метрикам анализатора CppLint

Название ошибки	Кол-во появлений в изначальном алгоритме	Кол-во появлений в старом интегрированном алгоритме	Кол-во появлений в новом интегрированном алгоритме
Tab found; better to use spaces	352	140	0
Lines should be <= 100 characters long	88	37	0
Missing space after ','	65	21	0
If an else has a brace on one side, it should have it on both	60	20	0
An else should appear on the same line as the preceding }	52	16	0
At least two spaces is best between code and comments	41	20	0
Missing spaces around '='	23	4	0
{ should almost always be at the end of the previous line	22	30	0
#ifndef header guard has wrong style	24	0	0
Is this a non-const reference? If so, make const or use a pointer	15	15	5

Таким образом, результаты статического анализа кода подтвердили то, что его качество улучшилось в ходе выполнения курсовой работы. Большинство выявленных ошибок в оригинальном коде [1] были исправлены. Некоторые виды недочётов были исправлены не полностью, но эти недочёты больше влияют на красоту, а не на правильность кода и количество появлений этих недочётов в коде было уменьшено как минимум в два раза по сравнению с изначальным кодом.

Заключение

Алгоритм FD-First-DFS, интегрированный в ветку edbt репозитория Desbordante [4], был модернизирован и улучшен и в итоге был интегрирован в основную ветку репозитория Desbordante.

Результаты:

1. В ходе рассмотрения кода на наличие недочётов были выявлены ошибки, связанные с архитектурой кода, с несоответствием современным стандартам языка C++, с плохой читаемостью кода и с нарушением Google C++ style guide.
2. Архитектура кода алгоритма FD-First-DFS была изменена так, чтобы он был легко расширяемым и чтобы он соответствовал принципу единственной ответственности. Фрагменты кода, не соответствующие современному стандарту C++17 были заменены так, чтобы код следовал современным практикам языка. Также код был изменен так, чтобы он соответствовал Google C++ style guide и был лучше читаем.
3. Был проведён анализ качества изменённого и изначального кодов с помощью статических анализаторов PVS-Studio и CppLint. Он подтвердил то, что качество кода улучшилось в ходе выполнения курсовой работы.

Код алгоритма FD-First-DFS, который был интегрирован в основную ветку репозитория Desbordante, доступен по ссылке: <https://github.com/Mstrutov/Desbordante/tree/main/src/algorithms/cfd>.

Список литературы

- [1] Rammelaere Joeri. Implementation code of algorithms in [2]. — 2018. — URL: <https://codeocean.com/capsule/6146641/tree/v1>.
- [2] Rammelaere Joeri, Geerts Floris. Revisiting Conditional Functional Dependency Discovery: Splitting the “C” from the “FD” // Machine Learning and Knowledge Discovery in Databases / Ed. by Michele Berlingario, Francesco Bonchi, Thomas Gärtner et al. — Cham : Springer International Publishing, 2019. — P. 552–568.
- [3] Vydrov Mikhail. Fixed implementation of algorithms from [2], merged into main branch of Desbordante repository. — URL: <https://github.com/Mstrutov/Desbordante/tree/main/src/algorithms/cfd>.
- [4] Vydrov Mikhail. Old implementation of algorithms from [2], merged into edbt branch of Desbordante repository. — URL: <https://github.com/Mstrutov/Desbordante/tree/edbt/src/algorithms/cfd>.