Научная статья
УДК 004.41
https://doi.org/10.31854/1813-324X-2024-10-5-118-128

Исследование распределения константных значений в исходном коде программ на языке С

Константин Евгеньевич Израилов, konstantin.izrailov@mail.ru

Санкт-Петербургский Федеральный исследовательский центр Российской академии наук,
Санкт-Петербург, 199178, Российская Федерация

Аннотация

В настоящее время ключевую роль в разработке программного обеспечения играет программная инженерия, одним из критерия развитости которой является изучение ее фактологии и различных научно-практических закономерностей. Важным аспектом данной области является логика выполнения программ, оперирующая внутренними данными, и, в частности – константными значениями, выявление закономерностей в которых и актуализирует данное исследование. В качестве основных применений данной закономерности можно отметить такие, как получение фундаментальных знаний об алгоритмах, создание новых и расширение существующих метрик оценки и сравнения программного кода, развитие методов его оптимизации, применение в генетическом программировании и др.

***Целью** настоящей статьи получение частотного распределения константных значений в исходном коде программ на языке программирования С.*

***Сущность** представленного подхода заключается в создании метода статистического анализа текста исходных кодов программ, содержащихся в датасете ExeBench (который состоит из огромного количества исходного кода функций на языке программирования С, их ассемблерного кода для различных процессорных архитектур, ошибок компиляции и другой информации).*

*Предложенный метод **базируется** на применении алгоритмов лексического и синтаксического разбора функций исходного кода, семантического определения типов констант, преобразования записи символов языка программирования в соответствующие числовые или строковые значения.*

*Метод **имеет реализацию** в виде программного средства на языке программирования Python, приведенного в виде интуитивно понятного псевдокода.*

***Эксперименты** с применением данного прототипа позволили получить искомое распределение константных значений для исходного кода программ на языке программирования С. **Анализ** полученных результатов позволил сделать ряд важных теоретико-практических выводов касательно наиболее часто используемых констант, соответствия полученного распределения закону Ципфа и близость к показательной функции, аномального появления ряда констант в Top-50 и др.*

***Научная новизна** предложенного подхода заключается в том, что распределение константных значений для исходного кода программ на языке программирования С получено впервые.*

***Теоретическая значимость** состоит в получении новых фундаментальных знаний касательно особенностей и закономерностей конструкций исходного кода, которые могут быть расширены и на другие языки программирования.*

***Практическая значимость** заключается в применении распределения для большого спектра задач, включая авторский генетический реверс-инжиниринг, который сам по себе является качественно новым направлением.*

Ключевые слова: *исходный код, константные значения, распределение, метод, прототип, эксперимент, генетический реверс-инжиниринг*

Ссылка для цитирования: Израилов К.Е. Исследование распределения константных значений в исходном коде программ на языке С // Труды учебных заведений связи. 2024. Т. 10. № 5. С. 119–129. DOI:10.31854/1813-324X-2024-10-5-118-128. EDN:KARAVM

Original research
https://doi.org/10.31854/1813-324X-2024-10-5-118-128

Constant Values Distribution Investigation in the C Programs Source Code

Konstantin E. Izrailov, konstantin.izrailov@mail.ru

Saint-Petersburg Federal Research Center of the Russian Academy of Sciences,
St. Petersburg, 199178, Russian Federation

Annotation

Currently, software engineering plays a key role in software development, one of the criteria for the development of which is the investigation of its factology and various scientific and practical patterns. An important aspect of this area is the logic of program execution, operating with internal data, and, in particular, constant values, the identification of patterns in which actualizes this research. The main applications of this pattern include obtaining fundamental knowledge about algorithms, creating new and expanding existing metrics for evaluating and comparing program code, developing methods for its optimization, using it in genetic programming, etc.

*The **purpose** of this article is to obtain the frequency distribution of constant values in the source code of programs in the C programming language.*

*The **essence** of the presented approach is to create a method for statistical analysis of the text of the source codes of programs contained in the ExeBench dataset (which consists of a huge amount of source code of functions in the C programming language, their assembler code for various processor architectures, compilation errors and other information).*

*The proposed method is **based** on the use of algorithms for lexical and syntactic analysis of source code functions, semantic definition of constant types, and conversion of the recording of programming language symbols into the corresponding numeric or string values.*

*The method **has an implementation** in the form of a software tool in the Python programming language, given in the form of an intuitive pseudocode. **Experiments** using this prototype allowed us to obtain the desired distribution of constant values for the source code of programs in the C programming language. **Analysis** of the obtained results allowed us to make a number of important theoretical and practical conclusions regarding the most frequently used constants, the correspondence of the obtained distribution to the Zipf law and its proximity to the exponential function, the anomalous appearance of a number of constants in the Top 50, etc.*

*The **scientific novelty** of the proposed approach lies in the fact that the distribution of constant values for the source code of programs in the C programming language is obtained for the first time.*

*The **theoretical significance** consists in obtaining new fundamental knowledge regarding the features and patterns of source code constructions, which can be extended to other programming languages.*

*The **practical significance** consists in applying the distribution to a wide range of tasks, including the author's genetic reverse engineering, which in itself is a qualitatively new direction.*

Keywords: *source code, constant values, distribution, method, prototype, experiment, genetic reverse engineering*

For citation: Izrailov K.E. Constant Values Distribution Investigation in the C Programs Source Code. *Proceedings of Telecommunication Universities.* 2024;10(5):119–129. (in Russ.) DOI:10.31854/1813-324X-2024-10-5-118-128. EDN:KARAVM

Введение

Центральным звеном информационных технологий, обеспечивающих функционирование практически всех сфер общества, является программное обеспечение (далее – ПО), логика работы которого в преобладающем большинстве случаев задается исходным кодом (далее – ИК), создаваемом на некотором языке программирования. Для качественного улучшения самого ПО и процесса его разработки была сформирована область программной инженерии, требующая не только практического применения, но и изучение существую-

щих в ней фактов, законов и других научных аспектов. Это позволит как повышать эффективность разрабатываемых программ (например, создавая высокопроизводительные алгоритмы), так и обеспечивать их безопасность (например, выявляя критичные ошибки в коде).

Исходя из достаточно продолжительной истории разработки программ (одни из первых языков Fortran и Алгол были созданы еще в 60-х годах прошлого столетия [1]), а также из многообразия парадигм программирования и огромного количества выпущенного ПО, сформировался некоторый пул связанных с ними знаний. Так, существует подобласть, направленная на предотвращение угроз безопасности информации, в рамках которой созданы базы сигнатур наиболее известных уязвимостей ИК и машинного кода (далее – МК). В рамках теории создания компиляторов созданы специализированные представления как самих языков программирования: например, форма Бэкуса – Наура, так и программ: дерево абстрактного синтаксиса (далее – ДАС). Адаптация искусственного интеллекта в части эволюционных алгоритмов позволила развить область генетического программирования для автоматического создания программы путем их итеративного улучшения согласно заданным критериям. Все эти подобласти так или иначе оперируют определенной логикой выполнения программ, важной частью которой являются внутренние данные, и, в частности – константные значения. В ином случае (т. е. если бы константы в ИК отсутствовали, как класс) любая сложная программа была бы крайне абстрактной (с позиции логики своего выполнения) или должна была настраиваться внешними параметрами, что снизило бы удобство ее использования до неприемлемого уровня; впрочем, стоит отметить, что в случае небольших задач использование константных значений не является необходимым, например, для функции нахождения максимального из нескольких чисел.

Исходя из вышесказанного, изучение фактов и законов применения константных значений в ИК является актуальным в рамках программной инженерии. А поскольку одним из наиболее популярных является язык C, позволяющий создавать программы для широкого спектра устройств – стационарных, мобильных, встроенных и т. п., то в рамках исследования будет использоваться именно он. Задача текущего исследования (далее – Задача) была сформулирована следующим образом: «Требуется получить частотное распределение константных значений в исходном коде программ на языке программирования C».

Применение распределения константных значений

Для обоснования значимости будущего результата исследования приведем некоторые из возможных применений распределения константных значений (далее – РКЗ).

Фундаментальные знания

Само по себе РКЗ расширяет область знаний о параметрах алгоритмов на языках программирования, применяемых для решения большого круга задач. Так, уже сейчас очевидно, что для программ, алгоритмы которых построены на логических операторах, наиболее частыми константами должны быть 1 и 0, соответствующие булевским значениям «Истина» (*аналог на англ.* – «True») и «Ложь» (*аналог на англ.* – «False»); а для нечеткой логики может добавиться третья константа – «–1».

Метрики кода

Частотное РКЗ для отдельно взятой программы может служить ее некоторой метрикой (по аналогии с другими [2]), а в ряде случаев – и «цифровым портретом», поскольку отражает специфику реализованных алгоритмов. Как результат, распределение можно использовать для частичной классификации программ по различным типам, а также для их сравнения (например, с целью нахождения дубликатов). Естественно, РКЗ будет не основным способом получения и сравнения метрик, а одним из дополнительных инструментов.

Оптимизация кода

Понимание наиболее «популярных» констант в программах позволит частично оптимизировать их выполнение, например, адаптацией процессорных команд под работу с именно этими значениями; например, для сравнения переменных с числом «0» (в противовес сравнению с произвольным числом) в большинстве процессоров существует отдельная инструкция.

Генетическое программирование

Суть генетического программирования заключается в автоматическом создании программ для решения целевой задачи за счет применения генетических алгоритмов (далее – ГА) [3]. В процессе генетического программирования создается большое количество экземпляров ИК, который итеративно приближается к искомому, решающему целевую задачу. Основными операциями ГА являются скрещивание и мутация; первая «смешивает» экземпляры двух промежуточных программ, а вторая – изменяет случайные конструкции в них. Как результат, знания о константных значениях, наиболее часто используемых в программах, позволят ускорить процесс генетического программирования путем более быстрой генерации подходящего ИК.

Генетический реинжиниринг машинного кода

Ответвлением от генетического программирования можно считать авторское направление, заключающееся в проведении реверс-инжиниринга МК для получения его ИК с применением ГА – названное *генетической деэволюцией* или *реинжинирингом* (в случае дальнейшего получения из ИК его алгоритмов, архитектуры, концептуальной модели и т. д.) [4]. В этом случае решается обратная оптимизационная задача эволюционного подбора такого ИК, который бы в точности компилировался в заданный МК. Очевидно, что перебор всех возможных константных значений будет крайне неэффективным, что потребует информации об их наиболее часто применяемых вариантах (например, чисел 0, 1, –1, 255 и пр.). В этом случае использование РКЗ в ИК существенно снизит работное время генетического реинжиниринга (далее – ГРИ).

Отметим, что именно необходимость в развитии последнего и послужила основной предпосылкой к постановке указанной Задачи, решение которой приводится далее в статье.

Обзор релевантных работ

Как показал анализ публикаций, в общедоступном информационном научном пространстве отсутствуют ссылки на какие-либо исследования, непосредственно посвященные решению Задачи, что может обосновываться ее некоторой узостью и специфичностью. Поэтому, далее проведем краткий обзор работ, в которых в принципе упоминаются распределения, устанавливающие какие-либо закономерности для ИК и МК программ.

Машинный код для процессоров

В авторском исследовании, частично опубликованном в работе [5], устанавливается взаимосвязь частотного распределения байтового представления инструкций МК с его семейством процессоров. Сравнение такого распределения для некоторой программы с заранее вычисленными и известными «цифровыми портретами» (за счет применения машинного обучения в части классификации) позволяет с большой долей вероятности определять тип процессора, на котором программа выполняется. Похожим образом в работе [6] предлагается оценивать принадлежность файлов к следующим классам: выполняемые (т. е. МК), скриптовые (т. е. ИК), текстовые (т. е. частично ИК) и бинарные (т. е. данные).

Размер машинного и исходного кодов

Также в авторском исследовании [7] установлена взаимосвязь между размерами двух представлений программы – ее ИК на языке С и МК. Как результат, по имеющемуся МК можно предсказать размер ИК, близкого к тому, из которого была скомпилирована программа. Данное частное исследование входит в более общее, проводимое в рамках развития ГРИ, и предназначено для прогнозирования длины хромосомы (основного объекта эволюции в ГА), что существенно ускоряет подбор искомого ИК.

Закон Ципфа

Закон Ципфа, описывающий эмпирическую зависимость распределения частотности слов естественных языков [8], может быть частично применен и к области программной инженерии в части распределения количества конструкций ИК или идентификаторов переменных в нем.

Статистические закономерности

Статистическая обработка метаинформации из баз уязвимостей (например, NVD) с ранжированием частоты упоминания ключевых слов в них может построить вероятностные связи между элементами информационных систем и угрозами информационной безопасности [9].

Графовые зависимости

Существует целый пул исследований [10–12], посвященных получению зависимостей (как для ИК, так и МК) между вызовами функций, переходами между инструкциями, вычислениями переменных и т. п. Данные зависимости, как правило, отражаются в виде соответствующих графов, а их частым применением является поиск уязвимостей в ПО.

Частота выполнения фрагментов кода

В интересах оптимизации программ, их распараллеливания, тестирования, профилирования и других подобных задач исследователи в [13] оценивают частоту выполнения линейных участков кода программы, представленной в виде графа потока управления.

Таким образом, существует достаточно ограниченный пул способов получения и сравнения различных внутри программных зависимостей, а частотное РКЗ позволит расширить его новым оценочным инструментарием.

Датасет ExeBench

Получение РКЗ полностью теоретическим способом, скорее всего, невозможно, по причине отсутствия детализированной систематизации всех возможных задач, решаемых с помощью программной инженерии, а также применяемых в них алгоритмов. Таким образом, данное распределение может быть вычислено статистически – путем анализа большого набора программ, разработанных на языке программирования С. Подходящим для этого датасетом может служить собранный в рамках проекта ExeBench [14], содержащий огромное количество С-функций, их ассемблерного кода,

ошибок компиляции и другой информации. Все метаданные о функциях в датасете представлены в формате JSON, пример которой приведен ниже (символы «...» использованы для компактности описания):

```
{
  "text":
  {
    "path": "chris-se/dietlibc-
packaging/libugly/asctime_r.c",
    "func_def": "static void num2str(char *c, int i)\n{\n
c[0] = i / 10 + '0';\n c[1] = i % 10 + '0';\n}",
    "func_head": "void num2str(char *c, inti)",
    "fname": "num2str",
    "asm":
    {
      "real_gcc_x86_O0":
      {
        "pre_asm":
"\t.file\t\"\"\n\t.text\n\t.type\tnum2str, @function\n",
        "func_asm": "...",
        "target":
        {
          "impl": "gcc",
          "bits": 64,
          "lang": "gas",
          "o": "0"
        }
      },
      ...
    }
    ...
  }
  ...
}
```

Так, в JSON-примере содержится метаинформация о функции «num2str()» в виде следующих полей: путь к соответствующему файлу с ИК («path»), ее имя («fname»), сам ИК («funct_def»), сигнатура («funct_head»), результаты компиляции с различными настройками (раздел «asm» и компилятор «real_gcc» для «x86» с опцией без оптимизации «O0» с получением ассемблерного кода в «target») и др. Анализ функций датасета из проекта ExeBench позволит собрать информацию об используемых конструкциях языка программирования, выделить их необходимые типы (включая константы) и построить РКЗ.

Метод: шаги, прототип, эксперимент

Для построения РКЗ необходим анализ датасета с ИК нетривиальными алгоритмами, производящими лексический и синтаксический разбор ИК функций, семантическое определение типов констант, преобразование записи символов языка программирования в соответствующие числовые или строковые значения (например, ASCII-символ '*x*' имеет значение 120) и др. Для объединения же алгоритмов в единый процесс выполнения с корректной связью их параметров и результатов работы требуется соответствующий метод построения РКЗ (далее – Метод), описание которого, а также его программной реализации в виде прототипа (далее – Прототип) приведено далее.

Метод состоит из 7 следующих основных *шагов*.

Шаг 1. Загрузка датасета ExeBench

На данном шаге описания функций, содержащиеся в датасете ExeBench, загружаются во внутреннее представление и подготавливаются для дальнейшей обработки. В частности, изначально отбрасываются функции, приводящие к ошибкам компиляции (т. е. не содержащие корректный ИК).

Шаг 2. Выделение тела функций

Из отобранных описаний функций выделяется ИК их тела; шаг является формальным, поскольку не содержит существенной логики.

Шаг 3. Токенизация ИК функций

Для каждого выделенного ИК применяется операция токенизации [15], которая преобразует его текст в ДАС. Последующий обход ДАС, а также учет синтаксических типов его узлов, позволяет получить список используемых констант.

Шаг 4. Предобработка констант

На данном шаге происходит обработка полученного списка констант и получение их реальных значений с учетом синтаксиса языка программирования C. Так, значения строковых констант остаются без изменений (например, «"abc"»), а символьные (например, «'*x*'») и десятеричные (например, «123») константы преобразуются в числа; бинарные (начинающиеся с «0b»), восьмеричные (начинающиеся с «0o») и шестнадцатеричные (начинающиеся с «0x») константы также конвертируются в десятичные. Отдельным образом обрабатываются экранируемые символы, такие, как перевод строки («'\n'» с кодом 10), табуляции («'\t'» с кодом 9) и др.

Шаг 5. Сбор статистики

Собирается общая статистика появления константных значений в ИК по всем функциям, полученным из датасета ExeBench.

Шаг 6. Формирование таблицы с РКЗ

Используя статистику о появлении константных значений в ИК, на данном шаге строится итоговая таблица их распределения – от наиболее частых к наименее.

Шаг 7. Выполнение аппроксимации для РКЗ

Используя табличное представление РКЗ, на данном шаге вычисляется его закономерность; в простейшем случае для этого может использоваться функционал, встроенный в Microsoft Excel, отвечающий за построение трендов для гистограмм; однако более «правильным» будет применение специализированных (программных) инструментов. Также научное предвидение позволяет предположить, что аппроксимирующей функцией должна стать показательная.

Метод был реализован в виде соответствующего *Прототипа* на языке Python (версия 3.11). Для

этого, в частности, использовались следующие программные библиотеки:

– json для загрузки и разбора описаний функций из датасета ExeBench, хранящихся в файлах формата JSON;

– pycparser для парсинга ИК функций, построения ДАС, определения типов узлов, выделения констант и пр.;

– numpy, scipy.optimize для аппроксимации РКЗ к заданной функции [16];

– matplotlib.pyplot для построения графиков РКЗ и иных функций;

– datetime, os, glob, pathlib для вспомогательных целей (доступ к датасету, отладочный вывод, сохранение результатов работы шагов Метода в файл и пр.).

На момент написания статьи по Прототипу подана заявка на получение свидетельства о регистрации программы для ЭВМ.

Псевдокод основного алгоритма Прототипа представлен на Листинге 1. Алгоритм на вход принимает путь к датасету ExeBench (через параметр «path»), содержащему JSON-файлы с описанием функций, а на выходе возвращает таблицу с РКЗ (*table*) и параметры аппроксимирующей показательной функции (*I*, *b* и *c*). И хотя псевдокод является интуитивно понятным, тем не менее, дадим краткие комментарии к его строкам.

Со строки 1 начинается реализация Шага 1, где создается пустой список для хранения записей датасета.

В строке 2 происходит загрузка датасета согласно заданному пути.

В строке 3 все записи датасета добавляются в созданную для хранения этого структуру.

Со строки 4 начинается реализация Шага 2, где создается пустой список для хранения описания функций.

В строке 5 начинается цикл по обходу сохраненных записей датасета.

В строке 6 достается описание функции из текущей записи датасета.

В строке 7 описание функции добавляется в созданную для хранения этого структуру.

В строке 8 заканчивается цикл, начатый в строке 5.

Со строки 9 начинается реализация Шага 3, где создается пустой список для хранения констант из ИК.

В строке 10 начинается цикл по обходу сохраненных описаний функций.

В строке 11 производится токенизация ИК тела текущей функции с получением списка его токенов.

Листинг 1. Алгоритм Прототипа

```
Input:
  path - путь к датасету ExeBench

Output:
  table - распределение константных значений (в виде таблицы)
  (a, b, c) - параметры формулы аппроксимации распределения константных значений

Begin
    // Шаг 1. Загрузка датасета ExeBench
 1: List<Record> records;
 2: dataset = LoadExeBench(path);
 3: records = dataset.Records();

    // Шаг 2. Выделение тела функций
 4: List<Function> functions;
 5: ForEach (rec In records) {
 6:   funct = GetFunction(record);
 7:   functions += funct;
 8: }

    // Шаг 3. Токенизация ИК функций
 9: List<Constant> constants;
10: ForEach (funct In functions) {
11:   toks = Tokenize(funct.Body)
12:   ForEach (tok In toks) {
13:     If (IsConstant(tok)) {
14:       constants += tok.Value;
15:     }
16:   }
17: }

    // Шаг 4. Предобработка констант
18: ForEach (&const In constants) {
19:   const = Preprocess(const);
20: }

    // Шаг 5. Сбор статистики
21: Dictionary<String, Integer> statistic;
22: ForEach (cnst In constants) {
23:   If (statistic.Has(cnst) == False) {
24:     statistic[cnst] = 0;
25:  }
26:  statistic[cnst] += 1;
27: }

    // Шаг 6. Формирование таблицы с РКЗ
28: Table<2> table;
29: ForEach ((value, count) In statistic.Items()) {
30:   table.AddRow([value, count]);
31: }

    // Шаг 7. Выполнение апроксимации для РКЗ
32: List<Integer> counts;
33: counts = statistic.Values();

34: Function<(x), (a, b, c)> funct;
35: funct = { a * b ^ x + c }
36: (a, b, c) = MakeApproximation(funct, counts)

37: Return table, (a, b, c);
End
```

В строке 12 начинается цикл по обходу всех полученных токенов ИК функции.

В строке 13 проверяется, является ли текущий токен константой.

В строке 14, в случае успешности проверки в строке 13, значение токена-константы добавляется в созданную для хранения этого структуру.

В строке 15 заканчивается ветка условия, начатого в строке 13.

В строке 16 заканчивается цикл, начатый в строке 12.

В строке 17 заканчивается цикл, начатый в строке 10.

Со строки 18 начинается реализация Шага 4, где начинается цикл по обходу сохраненных констант; символ «&» означает, что текущий итератор цикла является ссылкой на элемент списка констант (а не копией) и, следовательно, его значение может быть изменено.

В строке 19 производится предобработка текущей константы с сохранением результата в их исходный список (через ссылку на объект-константу).

В строке 20 заканчивается цикл, начатый в строке 18.

Со строки 21 начинается реализация Шага 5, где создается пустой словарь для хранения статистики использования константных значений в ИК; ключами словаря являются значения константы (в строковой форме), а значениями словаря – целое число с общим количеством их вхождений.

В строке 22 начинается цикл по обходу сохраненных констант.

В строке 23 проверяется, отсутствует ли текущая константа в содержащем их словаре.

В строке 24, в случае успешности проверки в строке 23, текущая константа добавляется в словарь, а количество ее вхождений в ИК указывается равным 0.

В строке 25 заканчивается ветка условия, начатая в строке 23.

В строке 26 количество вхождений текущей константы ИК увеличивается на 1 (через инкрементацию значения в словаре по данному ключу).

В строке 27 заканчивается цикл, начатый в строке 22.

Со строки 28 начинается реализация Шага 6, где создается пустая таблица для хранения РКЗ с указанием количества столбцов, равных 2 (для хранения значения константы и количества ее вхождений, соответственно).

В строке 29 начинается цикл по обходу словаря со статистикой использования константных значений; итератором цикла является кортеж из пары «значение константы, количество вхождений».

В строке 30 в таблицу добавляется строка из двух элементов – значение константы и количество ее вхождений.

В строке 31 заканчивается цикл, начатый в строке 30.

Со строки 32 начинается реализация Шага 7, где создается пустой список для хранения всех вхождений констант, отсортированных по убыванию их количества.

В строке 33 список количества вхождений приравнивается к списку значений словаря со статистикой использования константных значений.

В строке 34 создается функция аппроксимации, принимающая на вход один аргумент «x» и имеющая 3 параметра «a, b, c».

В строке 35 функция аппроксимации задается, как показательная – «$a \times b^x + c$» (символ «^» в коде соответствует возведению в степень).

В строке 36 осуществляется приближение функции аппроксимации к РКЗ путем подбора ее параметров.

В строке 37 из алгоритма возвращается таблица с РКЗ, а также кортеж параметров функции аппроксимации.

Используя Прототип, реализующий Метод, а также датасет ExeBench, был проведен *эксперимент* по построению РКЗ. Также были отобраны группы тестов, содержащих функции с ИК, предназначенным для компиляции, а именно следующие: «real_test», «valid_real», «synth_test», «valid_synth», «train_real_simple_io», «train_synth_simple_io».

Ход эксперимента с Прототипом приведен в Листинге 2; запись в круглых скобках в начале каждой строки содержит время запуска операции, а префикс «Step N» (*перев. на русс.* Шаг) указывает на выполняемый шаг Метода.

Листинг 2. Ход эксперимента с Прототипом

```
(20:54:04) Start
(20:54:04) Step 1. Loading the ExeBench dataset:
(20:54:04)   from
'\real_test\data_0_time1678114487_default.jsonl' --> OK
(2132 functions)
...
(20:59:53)   from
'\train_synth_simple_io\data_0_time1677914260_default.jsonl'
--> OK (4786 functions)
(21:00:00) Step 2. Extracting the function body (219077
functions) --> OK (219077 bodies)
(21:00:01) Step 3. Tokenizing the SC functions (219077 bod-
ies) --> OK (63968 constants)
(21:06:46) Step 4. Preprocessing constants (63968 constants)
--> OK
(21:06:47) Step 5. Collecting statistics (63968 constants) -
-> OK
(21:06:48) Step 6. Forming a table with the CVD (63968 con-
stants) --> OK
(21:06:49) Step 7. Make approximation of the CVD (63968) ...
OK (Y = 182451.24926513588 * 0.5822164967795688 ^ X +
1297.6581713063692, error = [5.73335455e+03 1.06349873e-02
2.14300283e+02]) --> OK
(21:06:49) Finish
```

Согласно Листингу 2, работа Прототипа заняла 12 минут 45 секунд, в процессе чего было загружено 219077 функций (на языке программирования C), из которых было выделено 63968 константных значений; предварительно, данную выборку можно считать удовлетворительной для обоснованного построения РКН.

Аппроксимация РКН функцией «$y = a \times b^x + c$» дала следующую формулу (с округлением коэффициентов):

$$y = 182451.25 \times 0.58^x + 1297.66,$$

где x – порядковый номер константного значения; y – количество встретившихся констант в датасете

ExeBench; ошибки каждого из параметров аппроксимирующей функции (т. е. a, b и c) указаны в Листинге 2 в прямоугольных скобках после префикса «error».

Результаты

В результате работы Прототипа был построен график РКЗ (красного цвета), а также его аппроксимация к функции «$y = a \times b^x + c$» (зеленого цвета), что для первых 100 констант представлено на рисунке 1.; также на рисунке присутствует рафик, выражающий закон Ципфа (синего цвета).

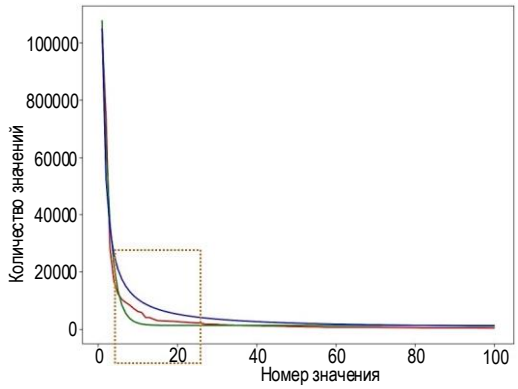

Рис. 1. Графики распределения константных значений для исходного кода программ на языке программирования C

Fig. 1. Distribution Graphs of Constant Values for the Source Code of Programs in the C Programming Language

Полученное РКЗ достаточно близко к аппроксимирующей функции за исключением области графика, выделенной оранжевым пунктирным прямоугольником – т. е. между константными значениями с порядковыми номерами (примерно) 5 и 25. Аналогичным образом РКЗ приблизительно соответствует закону Ципфа, что подчеркивает его корректность.

РКЗ для первых 50 элементов (отсортированных на Шаге 6 по мере убывания их количества) приведено в таблице 1; значение константы с 49-м номером соответствует двойным кавычкам.

ТАБЛИЦА 1. Распределение Топ-50 константных значений в ИК программ

TABLE 1. Distribution of Top-50 Constant Values in Programs Source Code

№ п/п	Значение	Количество	№	Значение	Количество
1	0	104731	26	11	2125
2	1	74503	27	64	1759
3	2	28142	28	14	1741
4	3	16724	29	«%d\n»	1681
5	4	12371	30	60	1677
6	10	10358	31	128	1495
7	5	9281	32	57	1491
8	8	8382	33	30	1393
9	16	7182	34	31	1365
10	7	6184	35	1000	1284
11	6	5781	36	45	1281
12	9	4040	37	17	1239
13	32	4028	38	24	1223
14	48	3459	39	«\n»	1178
15	100	2975	40	50	1162
16	15	2898	41	0.5	1151
17	65535	2835	42	47	1076
18	12	2751	43	42	1076
19	255	2716	44	1024	1072
20	65	2620	45	90	1051
21	«%d»	2488	46	256	1032
22	20	2445	47	99	1013
23	97	2306	48	40	919
24	4294967295	2244	49	«"»»	911
25	13	2172	50	70	907

Непосредственная частота использования каждого константного значения в ИК может быть получена тривиальным образом путем деления всех значений в распределении на максимальное – т. е. соответствующее самой 1-й константе.

Предварительные выводы

Согласно построенному РКН (см. рисунок 1), а также значениям самих констант (см. таблицу 1), можно сделать следующие предварительные, но важные теоретико-практические выводы.

Во-первых, наиболее частым с позиции использования оказалась нулевая константа (на позиции 1), что довольно закономерно, поскольку «0», как правило, является специальным значением в алгоритмах (по крайней мере, на языке программирования C); например, для внутреннего представления булевского значения «Ложь», определения знака числа, сравнения указателя с нулевым, обнуления объекта и т. п.

Во-вторых, следующей по частоте константой идет значение «1», что частично обосновывается логикой, близкой к объяснению расположения константы «0» – внутренним представлением булевского значения «Истина»; также достаточно частой операцией во многих алгоритмах с циклами является инкрементация и декрементация целочисленных переменных (т. е. прибавление или вычитание из них единицы).

В-третьих, отличие РКЗ от двух других, теоретически полученных графиков объяснимо спецификой представления констант в реальных программах, в частности тем, что первые по порядку значения соответствуют числам 0 и 1, которые интерпретируются не только как элементы счет-

ного множества, но и как числовое представление булевских «Истина» и «Ложь»; как результат, аномальное превышение встречи в ИК «опускает» аппроксимирующий график (из-за нарушения в закономерности показательной функции) и «поднимает» график Ципфа (из-за того, что он строится по 1-му элементу). Это, в том числе, приводит к существенному (визуально) расхождению полученного распределения от других графиков на участке, выделенном пунктирным прямоугольником (с диапазоном абсциссы от 5 до 25).

В-четвертых, первые 5 констант имеют значения, соответствующие своим порядковым номерам в таблице, т. е. 0, 1, 2, 3 и 4. Такая закономерность объяснима тем, что алгоритмы зачастую содержат признаки реального мира и его субъектов (т. е. людей), которые достаточно часто оперируют небольшим количеством объектов – до 5; впрочем, между константами «4» и «5» расположена другая константа, о чем будет сказано далее.

В-пятых, нахождение значения «10» на высоком месте в РКН между «4» и «5» объяснимо принятым в мире использованием десятичного исчисления; также данное значение соответствует коду символа перевода строки, используемого для создания многострочных текстов.

В-шестых, на высоких местах в РКЗ оказались числа, соответствующие числу 2 в различных степенях: 2^0 – на позиции 2; 2^1 – на позиции 3; 2^2 – на позиции 5; 2^3 – на позиции 8; 2^4 – на позиции 9; 2^5 – на позиции 13; 2^6 – на позиции 27; 2^7 – на позиции 31; 2^8 – на позиции 19 и т. д. Данная закономерность может быть объяснена применением битовых полей и масок с одним установленным разрядом, что достаточно часто применяется в различных алгоритмах.

В-седьмых, нахождение в Топ-25 значений 255, 65535 и 4294967295 также вполне логично, т. к. данные числа соответствуют записи значения «–1» в переменных с размером 1, 2 и 4 байта.

В-восьмых, нахождение в Топ-50 значений 100 и 1000 соответствует той же логике, что и высокая позиция числа 10 – применимость чисел в реальной жизни, для решения задач которой ИК алгоритмов фактически и создается.

В-девятых, в Топ-50 попали следующие строковые (или символьные) константы – «"%d"», «"%d\n"», «"\n"» и «""», которые соответствуют специальным конструкциям для указания формата вывода целочисленных значений, таких значений с переводом строки, самому переводу строки и пустой строке. Данные константы являются достаточно часто используемыми в ИК программ, формирующих и выводящих человекочитаемый текст.

И, в-десятых, среди первых 50 констант большинство оказалось целочисленными (55 значений), несколько – строковыми (4 значения) и только одно дробным – «0.5». Такую особенность можно обосновать распространенностью алгоритмов с нетривиальными формулами, логика которых основана на оценке близости дробного значения к ближайшему целому; например, при округлении числа.

Все десять сделанных основных выводов были подтверждены экспертно путем анализа ИК функций из датасета.

Генетический реинжиниринг

Несмотря на определенную значимость проведенного исследования и его результатов, основное применение РКЗ, как указывалось, заключается в развитии авторского направления ГРИ, предназначенного для получения ИК программы (а также более высокоуровневых представлений – алгоритмов, архитектуры и пр.) из ее МК. В его рамках решается оптимизационная задача подбора такого экземпляра ИК, который бы в точности компилировался в заданный МК. Один из подходов к решению данной задачи состоит в применении ГА, для чего генерируется популяция особей-экземпляров ИК, из которой выбираются программы, дающие после компиляции МК, наиболее близкий к исследуемому. Затем, над отобранными подобным образом «наилучшими» представителями популяции осуществляются операции скрещивания – перемешиванием конструкций для пары ИК, и мутации – случайным изменением конструкций ИК. Для качественного повышения оперативности работы ГРИ конструкции ИК программ создаются согласно формальному синтаксису языка программирования. При этом имена переменных и значения констант, очевидно, не относятся к области синтаксиса, а являются лексическим значением соответствующего токена. И если точные имена переменных для корректного выполнения программ не имеют существенного значения (поскольку назначаются разработчиками для улучшения понятности кода), то значения констант непосредственно влияют на логику выполнения. А т. к. полный перебор констант займет недопустимо большой промежуток времени, его ускорение путем использования списка наиболее часто используемых значений (т. е. полученного РКЗ) будет считаться качественной оптимизацией всего ГРИ. Таким образом, полученное РКЗ с точки зрения ГРИ является существенно значимым результатом.

Необходимо отметить, что, согласно полученным ранее результатам автора, реверс-инжиниринг МК в ИК для небольшого, но нетривиального математического выражения:

$$z = x^*(y - (x/y))$$

вручную экспертом занимает около 5 минут. Применение полного перебора конструкций ИК по следующему формальному синтаксису такого рода выражений:

```
1: expr_assign ::= ident, '=', expr ;
2: expr ::= ident | expr_oper ;
3: expr_oper = ident, oper, ident | ident, oper, '(',
expr_oper, ')' ;
4: oper = '+' | '-' | '*' | '/' ;
5: ident = 'x' | 'y' | 'z' ;
```

в полностью автоматическом режиме займет соизмеримые 10 минут. Первые же эксперименты с прототипом для проведения ГРИ позволили получить ИК менее чем за 10 секунд, что подтверждает перспективность направления (по крайне мере, для отдельных условия и сценариев применения).

Заключение

В работе решается частная задача получения РКЗ для ИК программ на языке С, вследствие чего описывается соответствующий Метод и его реализация в виде Прототипа, а также проводится эксперимент с получением искомого распределения. Близость РКЗ к аппроксимирующей показательной функции (т. е. наличие закономерности) и закону Ципфа (т. е. отражение эмпирических зависимостей) частично обосновывает корректность результата (в частности, выбор датасета ExeBench). Новизна исследования заключается в том, что РКН для ИК программ на языке программирования С получено впервые. Основная теоретическая значимость состоит в получении новых фундаментальных знаний касательно особенностей и закономерностей конструкций ИК (которые могут быть расширены и на другие языки программирования), а практическая значимость – в применении распределения для большого спектра задач (указанных ранее во введении), включая ГРИ, который сам по себе является качественно новым направлением.

Продолжением работы может стать получение подобных РКН для ИК других языков программирования с применением датасетов большего объема, их сравнение и, гипотетически, обобщение с целью получения единой закономерности, которая будет являться не только прикладной таблицей-справочником, но и отражать ключевые «связки» в мыслительной деятельности человека (интерпретируемые, как «переплетения нитей», говоря словами одного из величайших философов XX века Л. Витгенштейна [17]) по решению практических задач вне зависимости от их специфики и области применения (например, использование исключительно бинарной логики, оперирующей значениями 0 и 1, определяя тем самым дискретность и, следовательно, ограниченность восприятия человеком окружающего мира).

Список источников

1. Касторнов А.Ф., Касторнова В.А. Языки программирования и их роль в становлении предметной области "Информатика" // Педагогическая информатика. 2016. № 1. С. 59–68. EDN:VUUFHV
2. Коновалов Г.Г. Измерение качества чистого кода: метрики и инструменты анализа // Тенденции развития науки и образования. 2023. № 102-5. С. 25–28. DOI:10.18411/trnio-10-2023-244. EDN:GDPWLC
3. Хлыстов И.С., Жарова О.Ю. Генетическое программирование // Электронный журнал: наука, техника и образование. 2016. № 4(9). С. 62–67. EDN:XHJVHH
4. Израилов К.Е. Концепция генетической декомпиляции машинного кода телекоммуникационных устройств // Труды учебных заведений связи. 2021. Т. 7. № 4. С. 10–17. DOI:10.31854/1813-324X-2021-7-4-95-109. EDN:AIOFPM
5. Kotenko I., Izrailov K., Buinevich M. Analytical Modeling for Identification of the Machine Code Architecture of Cyber-physical Devices in Smart Homes // Sensors. 2022. Vol. 22. Iss. 3. PP. 1017. DOI:10.3390/s22031017
6. Буйневич М.В., Израилов К.Е. Способ классификации файлов на базе технологии машинного обучения // Вестник Санкт-Петербургского государственного университета технологии и дизайна. Серия 1: Естественные и технические науки. 2020. № 1. С. 34–41. DOI:10.46418/2079-8199_2020_1_6. EDN:MDPYTW
7. Израилов К.Е. Прогнозирование размера исходного кода бинарной программы в интересах ее интеллектуального реверс-инжиниринга // Вопросы кибербезопасности. 2024. № 4(62). С. 13–25. DOI:10.21681/2311-3456-2024-4-13-25. EDN:NRFCND
8. Кучерова С.В. Закон Ципфа и его приложения в области лингвистики // Некоторые вопросы анализа, алгебры, геометрии и математического образования. 2020. № 10. С. 107–108. EDN:QRNCOY
9. Leonov N., Buinevich M., Chechulin A. Top-20 Weakest from Cybersecurity Elements of the Industry Production and Technology Platform 4.0 Information Systems // Proceedings of the International Russian Smart Industry Conference (SmartIndustryCon, Sochi, Russian, 25–29 March 2024). IEEE, 2024. PP. 668–675. DOI:10.1109/SmartIndustryCon61328.2024.10515678
10. Фомин А.И. Оценка сложности исследования дизассемблированного кода исполняемых программ // Естественные и технические науки. 2021. № 7(158). С. 210–211. EDN:UBNPCY
11. Ормонова Э.М. Определение качества программного продукта на основе теории графов // Наука. Образование. Техника. 2021. № 1(70). С. 37–44. EDN:ITSANI
12. Лебедев В.В. Деобфускация control flow flattening средствами символьного исполнения // Прикладная дискретная математика. Приложение. 2021. № 14. С. 134–138. DOI:10.17223/2226308X/14/29. EDN:ITNATQ
13. Королев В.Ю., Смелянский Р.Л., Смелянский Т.Р., Шалимов А.В. Об оценивании частоты выполнения фрагментов кода последовательной программы // Известия Российской академии наук. Теория и системы управления. 2015. № 4. С. 39. DOI:10.7868/S0002338815040095. EDN:RXZZRT

14. Armengol-Estapé J., Woodruff J., Brauckmann A., Magalhães J.W.S., O'Boyle M.F.P. ExeBench: an ML-scale dataset of executable C functions // Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (New York, USA, 13 June 2022). ACM, 2022. PP. 50–59. DOI:10.1145/3520312.3534867

15. Toomey W. Ctcompare: Code clone detection using hashed token sequences // Proceedings of the 6th International Workshop on Software Clones (IWSC, Zurich, Switzerland, 04 June 2012). IEEE, 2012. PP. 92–93. DOI:10.1109/IWSC.2012.6227881

16. Samuelsson C. Comparative evaluation of the stochastic simplex bisection algorithm and the SciPy.Optimize module // Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS, Lodz, Poland, 13–16 September 2015). IEEE, 2015. PP. 573–578. DOI:10.15439/2015F47

17. Барляева Е.А. Мыслительная деятельность человека в метафорах и сравнениях // Вестник Воронежского государственного университета. Серия: Лингвистика и межкультурная коммуникация. 2016. № 3. С. 15–18. EDN:WKNUBD

References

1. Kastornov A.F.1, Kastornova V.A. Programming languages and their role in formation of subject domain of "Information Scientist. *Pedagogical Informatics*. 2016;1:59–68. (in Russ.) EDN:VUUFHV

2. Konovalov G.G. Measuring the quality of clean code: Metrics and analysis tools. Tendentsii razvitiia nauki i obrazovaniia. 2023;102-5:25-28. (in Russ.) DOI:10.18411/trnio-10-2023-244. EDN:GDPWLC

3. Hlystov I.S., Zharova O.Y. Genetic programming. *Electronic Journal: Science, Technology and Education*. 2016;4(9):62–67. (in Russ.) EDN:XHJVHH

4. Izrailov K. The Genetic Decompilation Concept of the Telecommunication Devices Machine Code. *Proceedings of Telecommunication Universities*. 2021;7(4):95–109. (in Russ.) DOI:10.31854/1813-324X-2021-7-4-95-109. EDN:AIOFPM

5. Kotenko I., Izrailov K., Buinevich M. Analytical Modeling for Identification of the Machine Code Architecture of Cyber-physical Devices in Smart Homes. *Sensors*. 2022;22.(3):1017. DOI:10.3390/s22031017

6. Buynevich M.V., Izrailov K.E. Method for classification of files on the basis of machine training technology. *Vestnik of St. Petersburg State University of Technology and Design. Series 1. Natural and technical sciences*. 2020;1:34–41. (in Russ.) DOI:10.46418/2079-8199_2020_1_6. EDN:MDPYTW

7. Izrailov K.E. Predicting the size of the source code of a binary program in the interests of its intellectual reverse engineering. *Voprosy kiberbezopasnosti*. 2024;4(62):13–25. (in Russ.) DOI:10.21681/2311-3456-2024-4-13-25. EDN:NRFCND

8. Kucherova S.V. Zipf's law and its applications in the field of linguistics. *Nekotorye voprosy analiza algebry geometrii i matematicheskogo obrazovaniia*. 2020;10:107–108. (in Russ.) EDN:QRNCOY

9. Leonov N., Buinevich M., Chechulin A. Top-20 Weakest from Cybersecurity Elements of the Industry Production and Technology Platform 4.0 Information Systems. *Proceedings of the International Russian Smart Industry Conference, SmartIndustryCon, 25–29 March 2024, Sochi, Russian*. IEEE; 2024. p.668–675. DOI:10.1109/SmartIndustryCon61328. 2024.10515678

10. Fomin A.I. Estimation of the difficulty of the disassembled code of the executed programs. *Natural and technical sciences*. 2021;7(158):210–211. (in Russ.) EDN:UBNPCY

11. Ormonova E.M. Determining the quality of the software product based on the theory of graphs. *Science. Education. Technology*. 2021;1(70):37–44. (in Russ.) EDN:ITSANI

12. Lebedev V.V. Control flow flattening deobfuscation using symbolic execution. *Applied Discrete Mathematics. Supplement*. 2021;14:134–138. (in Russ.) DOI:10.17223/2226308X/14/29. EDN:ITNATQ

13. Korolev V.Y., Smelyanskii R.L., Smelyanskii T.R., Shalimov A.V. On the estimation of the execution frequency of sequential program code snippets. *Journal of Computer and Systems Sciences International*. 2015;54(4):540–545. DOI:10.1134/S10 6230715040097. EDN:UFCQZB

14. Armengol-Estapé J., Woodruff J., Brauckmann A., Magalhães J.W.S., O'Boyle M.F.P. ExeBench: an ML-scale dataset of executable C functions. *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, 13 June 2022, New York, USA*. ACM;2022. p.50–59. DOI:10.1145/3520312.3534867

15. Toomey W. Ctcompare: Code clone detection using hashed token sequences. *Proceedings of the 6th International Workshop on Software Clones, IWSC, 04 June 2012, Zurich, Switzerland*. IEEE;2012. p.92–93. DOI:10.1109/IWSC.2012.6227881

16. Samuelsson C. Comparative evaluation of the stochastic simplex bisection algorithm and the SciPy.Optimize module. *Proceedings of the Federated Conference on Computer Science and Information Systems, FedCSIS, 13–16 September 2015, Lodz, Poland*. IEEE;2015. p.573–578. DOI:10.15439/2015F47

17. Barlyaeva E.A. Human mental activity in metaphors and similies. *Proceedings of Voronezh State University. Series: Linguistics and intercultural communication*. 2016;3:15–18. (in Russ.) EDN:WKNUBD

Статья поступила в редакцию 23.09.2024; одобрена после рецензирования 28.10.2024; принята к публикации 31.10.2024.

The article was submitted 23.09.2024; approved after reviewing 28.10.2024; accepted for publication 31.10.2024.

Информация об авторе:

ИЗРАИЛОВ Константин Евгеньевич

кандидат технических наук, доцент, старший научный сотрудник лаборатории проблем компьютерной безопасности Санкт-Петербургского Федерального исследовательского центра Российской академии наук

https://orcid.org/0000-0002-9412-5693

Автор сообщает об отсутствии конфликтов интересов.

The author declares no conflicts of interests.