Липецкий государственный технический университет

Кафедра автоматизированных систем управления

АКТУАЛЬНЫЕ ПРОБЛЕМЫ ТЕХНОЛОГИЙ ИНФОРМАТИЗАЦИИ И АВТОМАТИЗАЦИИ

СБОРНИК

материалов научно-технической конференции студентов кафедры АСУ

18 апреля 2022 г.

Материалы публикуются в авторской редакции с частичным сохранением авторской верстки

Липецк, 2022 г.

Содержание

ПРЕИМУЩЕСТВА UNIX-СИСТЕМ ПЕРЕД WINDOWS

Григорьев Иван Николаевич, группа АИ-21-1

Аннотация: В статье приведен сравнительный анализ отличий Linux от Windows. Рассмотрены важные критерии, которые покажут почему стоит попрощаться с Windows в пользу Linux.

Ключевые слова: Windows, Linux, причина, ОС, дистрибутив.

Данная статья может показать людям, что Windows, является далеко не самой удобной ОС. Да, он существует давно, но не для всех он является полезным в нынешнее время. Поэтому привожу альтернативу. Linux

Приведу ряд отличительных особенностей в пользу Linux. Одна из них это, что операционные системы на базе Linux имеют открытый исходный код и распространяются свободно, но это не единственное их преимущество.

А если говорить в общем, то начнем перечислять их

1) **Бесплатность ОС**. С финансовой точки зрения Linux обладает одним весьма существенным достоинством – она не коммерческая. Система имеет бесплатную открытую базу своих версий ОС и в процессе установки, у нас, как в windows не вылетает окно ключа активации. То есть за установку этой ОС нам не придётся платить никаких денег. И благодаря этой бесплатности вы можете попробовать все дистрибутивы Linux и выбрать, что нравится.

2) **Системные требования ОС**. Поговорим о мощности оборудования, которая нужна для нормальной работы ОС. Если говорить о стареньких ноутбуках, то можно забыть про Windows 10. Даже если там и запуститься ОС, то она всё равно будет очень сильно тормозить. Это будет вызывать очень сильный дискомфорт и будет отпадать желание работать за своим ноутбуком. Но если обратить внимание внимание на системные требования Linux, то здесь дистрибутивы более приветливы к слабому железу. Там существуют лёгкие рабочие столы и легкие программы, даже браузеры, могут работать там без остановки процесса, который приводит к зависанию. Однако стоит полагать, что даже Linux не спасёт старенький netbook с 1 гигабайтом оперативной памяти и одноядерным процессором.

3) **Стабильность ОС.** Иногда в работе ОС нас посещают различные проблемы, например, мы можем неаккуратно удалить или установить ПО, в следствии чего возникают сбои. Например, синий экран смерти, который заставляет твой компьютер выключится. Также можно столкнуться с несовместимостью программ. А еще есть такая вещь, которая называется перепадом напряжения, и после этого у вас может не просто выключится компьютер, но может и слететь ОС. Linux же сам сломаться и выключится не может. Это может произойти, только специально, при удалении жизненно-важных компонентов системы. Поэтому просто не стоит трогать незнакомые файлы, которые могут привести к сбоям

4) **Защита**. Windows является самой популярной средой для киберпреступников. Если ваш доход исчисляется в долларах, то посещения браузера без защиты может стоить вам очень дорого. В Linux же развитая архитектура, и особый способ разрешений пользователей, что делает его безопаснее. Да, слабость архитектуры Windows делает Linux на строчку выше. Также не стоит забывать про обычные вирусы, которых неисчисляемое количество на самой популярной ОС. Стоит что-то установить в браузере, так компьютер и полетит. Конечно на Linux тоже имеются вирусы, но вероятность наткнуться на них крайне мала. И в общем одна из причин грандиозного успеха Linux ОС как-раз-таки состоит в достаточно высокой степени безопасности ядра

5) **Бесплатность софта**. Обратим внимание на софт в сравниваемых нами ОС. Хм, возьмем самые важные программы, которые могут понадобится в работе. Текстовые редакторы, программа для подготовки презентация, электронные таблицы. В Windows это Word, Excel, PowerPoint соответственно. Этот софт можно приобрести, если вы купите Office, который в нынешнее время стоит 100 долларов. В Linux же эти программы не стоят ничего, кроме времени на установку. И это не единственные бесплатные приложения. Здесь может не быть популярных программ, но есть альтернативы им. Конечно, платные приложения тоже имеются в Linux, но их ничтожное количество

6) **Среда для программистов.** Есть миф, что Linux это чистая среда для программистов, следуя тому, что указано выше, мы выяснили, что это не только программирование. Но безусловно здесь отличная среда для них, которая открывает множество возможностей. В Linux не нужно устанавливать приложения для программирования, здесь уже

есть встроенный терминал. Одна из главных отличий систем под управлением ядра Linux – это открытый исходный код, благодаря этому можно индивидуально интегрировать систему под себя.

На основе моего сравнительного анализа, хочется сделать вывод и подчеркнуть остальные вещи. В пределах моей статьи невозможно рассмотреть все нюансы и критерии, касающиеся данной темы, но обозначение основных аспектов поможет в выборе ОС. Было приведено множество причин. И с каждым годом их будет становиться всё больше. Когда-то Linux был создан только для системы серверов. Сейчас же он используется и в IT компаниях и в быту.

Список литературы

1. Болл Билл, «Linux за 24 часа. Освой самостоятельно», Пер. с англ., М., Издательский дом «Вильямс», 1999. — 480 с.
2. Системы защиты Linux https://habr.com/ru/company/ruvds/blog/523872/
3. Таненбаум Э., Остин Т. Архитектура компьютера. 6-е изд. — СПб.: Питер, 2013. — 816 с.

ADVANTAGES OF UNIX SYSTEMS OVER WINDOWS

GRIGORIEV IVAN

student gr. AI-21-1

Lipetsk State Technical University

Annotation: The article presents a comparative analysis of the differences between Linux and Windows. Important criteria are considered that will show why it is worth saying goodbye to Windows in favor of Linux.

Keywords: Windows, Linux, reason, OC, distribution.

ИСТОРИЯ BLOCKCHAIN. МАЙНИНГ КРИПТОВАЛЮТ НА LINUX

Морозов Александр Александрович, группа АИ-21-1

Кургасов Василий Владимирович, канд. пед. наук, доцент кафедры АСУ

Аннотация: В статье рассматривается история появления блокчейна в интернете и, как следствие, криптовалюты. А также описан алгоритм установки скрипта для майнинга на Linux.

Ключевые слова: блокчейн, Линукс, криптография, майнер, криптовалюта.

Сначала разберёмся с самим интернетом. Сама технология интернета началась ещё очень давно. Ещё в 1974 году. Однако даже у этой технологии были некие предшественники, например интернет, который соединял между собой два компьютера. Далее появился протокол TCP/IP который позволял уже объединять между собой более большое количество компьютером. Это по своей сути позволяло обмениваться информацией более широкому количеству участников сети. Но потом в 1990 году появился HTTP - протокол, который позволяет коммуницировать с помощью различного контента. С помощью веб-контента, например, гипертекст и тому подобное. С разработкой этой технологии связан Вин Серф.

Это были первые три уровня развития. Но потом началось и коммерческое развитие, такие компании, как например Amazon, CISKO, 3COM. Которые мы и сегодня видим, но происходила ещё коммерциализация интернета.

В 1995 году выходит фильм THE NET. В этом фильме показывается момент, где была упомянута компания пицца – HUTS, это была ассоциация с первой в мире онлайн продажей, т.е. можно было с любой точки мира заказать пиццу. Однако была проблема. Люди тогда ещё не знали, как переводить деньги онлайн. Поэтому им приходилось оплачивать пиццу после доставки.

Теперь поговорим о криптографии, это очень важная тема, потому что основная тема - блокчейн, а криптография напрямую связана с этим. Криптография - технология шифрования и расшифровки той или иной информации. Делается это для того, чтобы люди не могли прочесть ту или иную информацию без наличия кода. Криптография занимается тем, что шифрует

информацию, чтобы нежелательные люди её не расшифровали. По сути, это коммуникация в присутствии противника.

Т.е. у нас есть какая - то вражеская сторона, которая хочет получить нашу зашифрованную информацию, но нам нужно коммуницировать так чтобы вражеская сторона эту информацию не получила.

В 1976 году в MIT была изобретена симметрическая криптография. Это сердце всех криптовалют, это сердце интернета. Ключевой момент этой технологии – это то, о чём говорили ранее, коммуникация в присутствии врагов. Простыми словами, как сохранить секрет, когда все его хотят узнать.

Однако криптография – это то, благодаря чему работает сейчас интернет.

SSL – это протокол, который используется для асимметрического шифрования. В целом технологии SSL и TLS защищают весь интернет полностью.

Вдруг внезапно после появления этой технологии можно было уже заказывать пиццу онлайн и онлайн с помощью кода оплачивать. С коммерческой стороны в 1998 году появилась компания PAYPAL, ранее были и другие компании, но большинство из них были безуспешны. Также стоит вспомнить такого человек как Adam Back который создал hash cache. Были некоторые инновации, которые действительно работали и были полезны — это Alipay и mpesa.

Движение денег без централизатора и есть основа блокчейна.

Блокчейн представляет собой децентрализованную распределенную запись (реестр) транзакций, в которой информация о транзакциях хранится в постоянном и почти неизменном виде с использованием криптографических методов. По сути, это база данных с отметками о времени, это значит, что вы можете туда добавлять небольшие куски информации и в этих блоках находится информация. И кстати самый долго живущий блокчейн — это не блокчейн биткоина.

Эти блоки данных создают прозрачную базу данных, которую всегда можно просматривать. База данных - защищена криптографией.

Главная часть блокчейна — это hash функция, изначально её использовали для того, чтобы находить какую-то информацию в базах данных и сохранять структурированную базу данных.

Но в блокчейне их стали использовать иначе. Во-первых, их используют для присоединения старых блоков к новому блоку.

Во-вторых, для подтверждения транзакций и для более наглядного хранения информации в базе данных.

Также стоит упомянуть очень важную часть блокчейна - консенсус соглашения т.е. это та часть программного кода, которая определяет, когда мы добавляем блок за блоком, и кто будет добавлять следующий блок. Это и называется консенсус протокол.

Транзакцию и эмиссию регулирует peer-to-peer сеть. Это одноранговая сеть, основанная на равноправии участников. В такой сети каждый узел (т.е. машина) может выполнять как функции клиента, так и сервера. Это значит, что каждая из машин-участников может как отправлять запросы другим машинам, так и должна быть способной обрабатывать запросы других машин сети.

Через полтора года как Сатоси Накамото опубликовал свой биткоин и запустил его в сеть, в сети появляется сообщение от информатика с города Флорида. Он хотел купить две пиццы за 10.000 биткоинов.

Основной момент, что кто-то пытается использовать биткоины, чтобы оплатить что-то в интернете, и 16 месяцев спустя выхода биткоина это ещё никто не делал. Спустя три дня он так и не получил свою пиццу. Проходит ещё один день. И он наконец получает свои пиццы.

Сейчас 10.000 биткоинов – это около 400 миллионов долларов. Получается он отдал 200 миллионов за одну пиццу. В честь этого, 22 мая - теперь биткоин-пицца день.

И-так, блокчейн достоверно перемещает данные внутри централизованной сети. И экономика блокчейна заключается в двух основных терминах: экономика сети и экономика верификации т.е. достоверности. Во многих случаях блокчейн добавляет некоторые стоимости к процессу верификации, но в тоже время он снижает другие стоимости. Потому что, нам, например, не нужно оплачивать некие расходы на какую-то третью сторону, которая будет проверять транзакции.

Теперь поговорим о его связи с Linux. Одно из популярных решений проблемы защиты криптовалют основано на использовании Linux, который считается практически непробиваемым для вирусов и хакерских атак. Все мы знаем, что криптовалюты можно майнить - искать в сети. Можно это делать и на

основе операционной сети Linux. Сейчас существует большое количество готовых сборок Linux, как правило, на базе Ubuntu, с достаточно простой установкой и управлением. Но самостоятельными дистрибутивами их назвать сложно.

После установки Ubuntu, система устанавливается с минимально необходимым для работы сервера набором программного обеспечения.

Надо для удобства добавить дополнительного пользователя и выдать ему права sudo. Также стоит добавить его в группу sudo. Делается это, чтобы не засорять основного пользователя.

Нам потребуются проприетарные драйверы, т.к. только они смогут обеспечить необходимую производительность, функционал и стабильность. В свободных драйверах на данный момент отсутствует полноценная поддержка OpenGL и CUDA, в связи с чем потребуются именно они. Также управление частотами памяти и графического ядра, и скоростью оборотов системы охлаждения доступны только в проприетарных драйверах. Сделать это можно несколькими способами. Скачать с сайта NVIDIA или из репозитория xorg-edgers.

Далее, стоит настроить свою видеокарту. Вот тут Linix не совсем хорош, в нём нет такой программы как MSI Afterburner, в которой достаточно просто и удобно проводить разгон видеокарты путем перемещения ползунка по шкале.

Разгон будет осуществляться через драйверы путем изменения необходимых значений рабочих частот и скорости оборотов вентиляторов системы охлаждения. Изначально, изменение всех этих параметров заблокировано. Для разблокировки необходимо выполнить команды, которые откроют нам доступ к управлению картами (кулеры, частоты).

Далее, остаётся только скачать сам майнер. Сделать это можно с GitHub’a. Надо распаковать его, копируем полученные данные в /usr/bin для дальнейшей простоты в использовании. Создаем скрипт с расширением .sh, который будет выполнять запуск майнера в утилите screen. После этих действий, наш компьютер готов искать деньги на пиццу.

Список литературы

1. Гари Генслер, лекция «MIT – blockchain», пер. с англ., 2018

2. Электронный ресурс – Режим доступа – «MIT» - https://www.mit.edu/innovation/

3. Электронный ресурс – Режим доступа – «Майнинг на Linux» - https://selectel.ru/blog/nachinaem-majnit-v-linux/

4. Электронный ресурс – Режим доступа –«Криптовалюта: история и перспективы», «ОСНОВНАЯ ХАРАКТЕРИСТИКА ЗАЩИТЫ КРИПТОВАЛЮТЫ В СОВРЕМЕННОМ МИРЕ», «Блокчейн и будущее международной торговли» - https://www.elibrary.ru/

HISTORY OF BLOCKCHAIN. CRYPTOCURRENCY MINING ON LINUX

Morozov Aleksandr Aleksandrovich

Student of group AI-21-1

Kurgasov Vasily Vladimirovich

Assistant professor

Lipetsk State Technical University

Annotation: The article discusses the history of the emergence of blockchain on the Internet and, as a result, cryptocurrencies. It also describes the algorithm for installing a script for mining on Linux.

Keywords: Blockchain, Linux, cryptography, miner, cryptocurrency.

ASTRA LINUX SPECIAL EDITION РЕЛИЗ «НОВОРОССИЙСК»

Морозов Даниил Сергеевич, группа ПИ-21-1

Аннотация: Ядро ОС предоставляет каждому системному процессу свое адресное пространство. Это основано на механизме подкачки, чтобы память была защищена и на механизме, который преобразует адрес виртуальный в адрес физический. Каждый доступ процессов к одной и той же ячейке памяти контролируется диспетчером, отвечающим за доступ, соблюдая дискреционные и обязательные правила управления доступом.

Ключевые слова: релиз, Новороссийск, astra, linux.

Многие страны в последнее время предпринимают попытки создания своей операционной системы. Это связано, прежде всего с тем, что код ОС Windows закрыт и никто не может сказать, что придет вместе с очередным обновлением.

Astra Linux — это операционная система на базе Linux, представленная в России как альтернатива Microsoft Windows.

В связи с последними событиями становится актуальна тема относительно Astra Linux.

Эта операционная система имеет множество релизов, но мы будем рассматривать релиз Novorossiysk.

Релиз «Новороссийск»— предназначена для ноутбуков и встраиваемых компьютеров с процессорной архитектурой ARM. Он имеет дружественный графический интерфейс для использования на устройствах с сенсорным экраном.

Ядро ОС предоставляет каждому системному процессу свое адресное пространство. Это основано на механизме подкачки, чтобы память была защищена и на механизме, который преобразует адрес виртуальный в адрес физический. Каждый доступ процессов к одной и той же ячейке памяти контролируется диспетчером, отвечающим за доступ, соблюдая дискреционные и обязательные правила управления доступом.

Он имеет отличную подсистему реестра, интегрированную со всеми частями ОС.

Информация графической подсистемы имеет защиту, обеспечивающую принудительное управление доступом в графических приложениях.

Также есть режим «киоска», который ограничивает действия. Которые может совершать пользователь. Жесткость ограничений определяется маской киоска, которая накладывается на права доступа к файлу каждый раз, когда пользователь пытается получить к нему доступ.

Когда пользователь входит в систему, права доступа к файлу конфигурации устанавливаются автоматически.

Новороссийская версия имеет хорошую защиту адресного пространства процесса, а также возможно использование NX-Bit. Существует проверка неизменности и аутентичности файлов ELF. Верификация основана на векторах аутентификации.

Другие разработчики могут встраивать векторы подлинности в создаваемое ими ПО.

Существует набор почтовых программ, который включает в себя почтовый сервер, состоящий из почты Exim и почты Dovecot, а также почтовый клиент Mozilla Thunderbird, предоставляющий:

• Интеграцию с ядром и основными библиотеками операционной системы для обеспечения обязательного контроля доступа к электронной почте, хранящейся в формате Maildir;

• Автоматическую маркировку пользовательских электронных писем с их текущим идентификационным контекстом.

Агент пересылки почты использует протокол SMTP и выполняет следующие задачи:

• доставка исходящей электронной почты от авторизованных клиентов на сервер, который является местом назначения обработки почтового домена получателя;

• получать и обрабатывать электронные сообщения с доменов, для которых они предназначены;

• перенаправление входящих электронных писем на обработку агентом доставки почты;

Таким образом можно сделать вывод, что дистрибутивы ОС, основанных на Linux, в частности релиз «Новороссийск», способны конкурировать с иностранными аналогами и в дальнейшем способны заменить их полностью.

Список литературы

1) Уорд Брайан. «Внутреннее устройство Linux». – М.: Наука и техника, 2021. 384 с.
2) Вовк Елена. «Astra Linux. Руководство по национальной операционной системе и совместимым офисным программам». – М.: Манн, Иванов и Фербер, 2022. 400 с.
3) Девянин Петр. «Безопасность операционной системы специального назначения Astra Linux Special Edition. Учебное пособие для вузов». – М.: Горячая линия - Телеком, 2022. 404 с.

ASTRA LINUX SPECIAL EDITION RELEASE "NOVOROSSIYSK"

Morozov Daniil Sergeevich
student of group PI-21-1
Lipetsk State Technical University

Summary: The OS kernel provides each system process with its own address space. It relies on a swap mechanism to keep the memory protected and a mechanism that converts a virtual address into a physical address. Each process access to the same cell is invoked by the dispatcher, observing the access, negotiating discrete and mandatory access control rules.

Keywords: release, Novorossiysk, astra, linux.

ОБ ОСНОВНЫХ КРИТЕРИЯХ ВЫБОРА ОПРЕДЕЛЕННОГО ВИДА БАЗЫ ДАННЫХ ПРИ РАЗРАБОТКЕ АВТОМАТИЗИРОВАННОЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Микаелян Анна Руслановна, группа ПИ-19-1

Аннотация: Статья содержит материал об основных критериях выбора между реляционной и документо-ориентированной БД с учетом того, что реляционные БД являются общепризнанным лидером и достоинства данного вида системы управления базами данных (СУБД) известны аудитории читателей.

Современная классификация баз данных (БД) подразделяет их на две группы: SQL и noSQL. Под SQL базами данных подразумеваются БД, основным языком запросов которых служит SQL. Это, как правило, БД, основанные на реляционной модели данных. Под NoSQL (Not Only SQL) подразумеваются все остальные виды. В наше время из noSQL БД основной акцент направлен на документо-ориентированные БД.

В 1980-х годах стали популярными реляционные базы данных, в 1990-х годах за ними последовали объектно-ориентированные базы данных. Совсем недавно вследствие роста Интернета и возникновения необходимости анализа неструктурированных данных появились базы данных NoSQL. В настоящее время облачные базы данных и автономные базы данных открывают новые возможности в отношении способов сбора, хранения, использования данных и управления ими.

На основе рейтингов PYPL [1] и StackOverflow [2] можно сделать вывод, что за 2020 и 2021 год самыми популярными СУБД являются реляционные и документо-ориентированные СУБД.

Реляционная модель данных — это не что иное как набор реляционных отношений, состоящих из неупорядоченных множеств атрибутов - кортежей и имеющих определенные отношения друг с другом. На основе этой модели создаются базы данных, в основе которых лежат таблицы. Соответствующая СУБД будет оперировать данными таблицами при исполнении SQL запросов.

В документно-ориентированных БД главным объектом является документ, который представляет собой набор пар «ключ-значение». В качестве ключа выступает имя атрибута данных, а в качестве значений - либо некое атомарное (простое) значение определенного атрибута, либо вложенный объект, состоящий из пар ключ-значение, либо массив атомарных атрибутов, либо массив вложенных объектов. В этом случае у нас появляется понятие вложенности. Данная структура может быть описана при помощи таких языков, как XML или JSON. Конкретный пример документно-ориентированных СУБД - MongoDB, CouchDB, Couchbase.

Выделим основные критерии, на которые можно опираться при выборе БД на начальном этапе проектирования информационной системы.

Большинство реляционных СУБД (РСУБД) являются проверенным во времени программным обеспечением. MySQL используется крупными компаниями более 15 лет, а стандарт SQL позволяет в случае чего легко мигрировать на другие РСУБД.

Если разработка автоматизированной информационной системы (АИС) ведется в команде, то в первую очередь стоит учесть опыт и предпочтения членов команды. К примеру, если реализацией проекта будет заниматься группа разработчиков с опытом работы только БД одного типа, то при ее выборе будет потрачено меньше ресурсов при разработке. В любом случае решение остается на стороне команды.

Далее рассмотрим вопрос производительности. Материал взят из Национальной библиотеки им. Н. Э. Б`аумана по состоянию на 25 июня 2018. Для автоматизации тестирования использовались пакеты Python для MySQL и MongoDB. Методика [3] и код для тестирования описаны сайте библиотеки. С индексацией в операциях добавления, поиска, изменения и удаления MySQL оказался быстрее MongoDB в среднем на 60%, а в операции соединения и вовсе в 12 раз. Без индексации результаты в разных операциях разнились, но чаще более быстрым оказывался MySql.

Конечно, больший практический интерес представляет работа с проиндексированной таблицей, но даже в этом случае однозначного победителя среди тестируемых СУБД выявить нельзя, потому что MongoDB подкупает своей гибкостью в создании документов, не привязанных к конкретной схеме.

Итак, следующий критерий - простота в создании и изменении структуры данных в СУБД. Документо-ориентированные БД держат первенство в этом плане, так как JSON-формат документов легче воспринимается человеком, а MongoDB не требует поддержания строгой структуры документов, хранимых в рамках одной коллекции (коллекция - аналог таблицы в документо-ориентированной БД). То есть в документо-ориентированной СУБД не нужно тратить усилия на описание структуры определенной БД. Можно сразу сконцентрироваться на описание данных в коде бизнес-приложения и на сохранение их в БД напрямую.

Далее затронем вопрос масштабируемости в плане расширения возможностей БД для использования АИС всё большим количеством пользователей. Масштабирование в нашем случае можно разделить на два вида:

- Вертикальное масштабирование — увеличение мощности машины. Оно проще реализуется, но не подходит для высоконагруженных систем.

- Горизонтальное масштабирование — добавление новых машин и распределение данных между ними. Оно более сложен в конфигурации, но может обеспечить более высокую производительность системы и большую безопасность данных при использовании дублирования.

Также имеет смысл говорить о том, масштабируем ли мы чтение, запись или объем данных.

Для MySQL добиться успешной масштабируемости можно путем использования маленьких транзакций и высокого числа ядер процессоров. Традиционно чтение в MySQL масштабируется путем дублирования, а запись — через распределение запросов между машинами.

MongoDB же изначально создавалась с расчетом на использование горизонтальной масштабируемости, поэтому она уже на уровне ядра будет выполнять её более эффективно в самых разных случаях.

Итак, подведем итог: оба вида БД имеют место для существования и используются для разных целей. Выбор РСУБД обосновывается скоростью работы, стандартизацией и разнообразием СУБД, позволяющей более гибко выбрать нужную технологию. С другой стороны, документо-ориентированная СУБД позволяет проще спроектировать и изменять структуру БД под будущие потребности АИС.

Список литературы

1) TOPDB Top Database index - TOPDB Top Database index

2) Stack Overflow site use - URL: https://insights.stackoverflow.com/survey/2021#stack-overflow-site-use-newso sites.

3) Сравнение производительности MongoDB vs MySQL – URL: https://ru.bmstu.wiki/Сравнение_производительности_MongoDB_vs_MySQL

4) Сравнение производительности MongoDB vs PostgreSQL. Часть I: No index - URL: https://habr.com/ru/post/197590/

СОСТАВЛЕНИЕ ГРАФА ЗАВИСИМОСТЕЙ ИСХОДНОГО КОДА НА ОСНОВЕ АБСТРАКТНОГО СИНТАКСИЧЕСКОГО ДЕРЕВА

Посаднев Виталий Витальевич, группа АС-18-1

1 Рассмотрение проблемы и выделение заимствования исходного кода

Плагиат программного кода является одной из серьезных проблем в образовательной сфере. В первую очередь это касается преподавателей тех дисциплин, на которых требуется писать программных код для выполнения поставленных задач. При этом используя чужие фрагменты кода в своих работах, учащиеся не всегда осознают, какой ущерб они наносят своим собственным знаниям и всему процессу их последующего обучения. Кроме непонимания важности самостоятельного выполнения работ можно выделить широкий спектр других причин, по которым студенты занимаются плагиатом кода, среди которых: отсутствие мотивации, теоретических и практических базовых знаний, времени на обучение и т.д.

Плагиат в программном коде определяется как «воспроизведение (копирование) исходного кода без внесения каких-либо изменений или с внесением, но незначительным». Для понимания того, что подразумевается под незначительными изменениями, следует рассмотреть, какие изменения возможны в программном коде.

Выделяют четыре типа заимствования фрагментов кода:

- первый тип – два фрагмента кода являются идентичными, за исключением изменения в пробельных символах, отступах и комментариях;
- второй тип – два фрагмента кода являются синтаксически идентичными, за исключением имен идентификаторов, строковых литералов, типов переменных и всех изменениях, свойственных первому типу;

- третий тип – один фрагмент кода был получен путем копирования другого фрагмента кода, но последующим добавлением и/или удалением операторов языка. При этом сохраняются все изменения, свойственные второму типу;
- четвертый тип – два фрагмента синтаксически реализованы по-разному, но выполняют одинаковые вычисления.

Исходя из приведенной классификации, можно сделать вывод, что к незначительным изменениям относятся первый, второй и третий тип дублирующих фрагментов кода.

2 Методы для обнаружения плагиата

Анализируя рынок можно сделать вывод, что программные средства выявления плагиата различаются по методам обнаружения дублирующих фрагментов, а именно:

- String-based – подход заключается в поиске точных строковых совпадениях. Данных подход является очень чувствительным к изменению имен идентификаторов;
- Metrics-based – логика метода заключается в оценке метрик программы с последующим сравнением. Например: количество используемых переменных, циклов, условных операторов и т.д.;
- Token-based – логикой данного метода является предварительная трансляция исходного кода программы в последовательность токенов, с последующим применением одного из алгоритмов строкового сравнения для поиска дублирующих фрагментов;
- Tree-based – заключается в том, что для каждого файла с исходным кодом строится абстрактное синтаксическое дерево, с последующим сравнением между собой;

- PDG-based (Program Dependency Graph) – данный подход заключается в построении графа зависимости, который отражает зависимость между управляющими конструкциями программы (control dependency), т.е. описание того как происходит переход управления из одной точки программы в другую, а также между потоками данных (data dependency). После построения данного графа для каждого файла с программным кодом происходит сравнение между собой;
- Hybrid-based – гибридные подходы основаны на сочетании нескольких из выше перечисленных подходов.

Каждый из перечисленных методов имеет свои характеристики. Наиболее важной характеристикой является ответ на вопрос: «Какого типа дублирующие фрагменты кода способен обнаруживать данный метод?» Например, методы, основанные на сравнении строк, способы обнаруживать дублирующие фрагменты кода первого типа. Методы основанные на токенизации и построении абстрактных синтаксических деревьев, работают для первого и второго типов. Методы, основанные на построении графов зависимостей, работают для первого, второго и третьего типов. Методы, основанные на метриках, практически не используются в настоящий момент времени, так как имеют слишком высокую вероятность ошибки.

2.1 String-based метод

Данный метод хорошо подходит для обучения студентов. Алгоритм прост в реализации, так как требуется сравнивать код двух программ без внесения изменений в семантику программы. Но также имеет и ряд недостатков, среди которых главным является – достаточно низкая точность выявления плагиата (так как данный метод чувствителен к внесению изменений будто комментариев или лишних отступов).

2.2 Metrics-based метод

Данный метод аналогичен string-based подходу. К преимуществам относится достаточно простая реализация, но к минусам можно отнести невысокую точность сравнимую со string-based подходом. Так как можно добавить несколько циклов и переменных к ним, которые никак не повлияют на семантическую целостность программы, но сильно повлияют на рассчитанную метрику.

2.3 Token-based метод

Данный метод достаточно широко используется в современных программных решениях. Для его реализации требуется составить список токенов (ключевые конструкции языка, идентификаторы, операторы и числовые значения), а также значения, которым они будут соответствовать. В сравнении с ранее представленными методами данный подход показывает себя в разы лучше, так как оперирует уже с преобразованными и нормализованными данными. Но также, данный метод имеет недостатки, среди которых можно выделить главный – семантика программы при неправильном составлении списка токенов может не учитываться, что повлечет за собой выявления плагиата у абсолютно различных программ.

2.4 Tree-based метод

Данный метод также известен как AST-based (abstract syntax tree). Данное дерево представляет собой программу вершинами которого является операторы языка программирования, а листья являются операндами. Данный подход также широко используется в статических анализаторах кода, который затем используется в качестве внутреннего представления в компиляторе или интерпретаторе компьютерной программы для оптимизации и генерации кода.

2.5 PDG-based метод

Данный метод используется в оптимизаторах для различных языков программирования. Задачей данного метода является составления графа на основе исходного кода программы для последующей обработки. Недостаток данного метода заключается в том, что для его работы зачастую приходится компилировать или интерпретировать программу для получения доступа к исходному коду. К преимуществам относится высокий показатель сведения программного кода к единообразию путем его оптимизации.

3 Разработка алгоритма составления графа зависимостей

Для разработки алгоритма составления графа зависимостей было принято решение комбинировать метод AST-based и PDG-based подход. При разработке алгоритма следует учитывать следующие этапы:

- этап подготовки (нормализации) исходного кода – задачей данного этапа является подготовка файла с исходным кодом к дальнейшим преобразованиям путем удаления комментариев и пустых фрагментов кода;
- этап построения AST – после того как исходный код был нормализован, происходит построение абстрактного синтаксического дерева. Необходимо выделить подграфы для каждой из функций исходного кода, ассоциировать каждую семантическую единицу с соответствующей строчкой кода для последующего его восстановления;
- этап построения PDG – после получения абстрактного синтаксического дерева происходит преобразование в граф с выделением подграфов после соответствующей оптимизации и удаления неиспользуемого (мертвого) кода;
- этап выделения максимально индуцированного подграфа относительно каждой функции – необходимо решить NP-полную задачу

с целью нахождения одинаковых подграфов. Для решения данного этапа был выбран алгоритм VF2;

- этап восстановления исходного кода – после выделения подграфов и обнаружения совпадений между ними происходит восстановление исходного кода с указанием мест с плагиатом.

4 Построение абстрактного синтаксического дерева

Для построения абстрактного синтаксического дерева (AST) необходимо составить словарь аналогичный тому, что ранее упоминалось в token-based подходе. AST отличается от дерева разбора тем, что в нём отсутствуют узлы и рёбра для тех синтаксических правил, которые не влияют на семантику программы. Классическим примером такого отсутствия являются группирующие скобки, так как в AST группировка операндов явно задаётся структурой дерева.

Для составления самого дерева можно использовать регулярные выражения с описанными правилами для конкретных конструкций языка. Таким образом нам необходимо добиться следующего:

- ассоциировать каждую семантическую единицу с фрагментом кода, а также идентификацией его с целью построения самого дерева вершинами которого и будут являться эти идентификаторы;
- разделить исходный код на функциональные единицы, иными словами – выделить функции, которые будут представлены отдельными подграфами данного дерева;
- построить само дерево с помощью указания вершин и ребер из ранее полученного набора данных.

Рассмотрим построение данного дерева на конкретном примере. Для примера был выбран алгоритм нахождение наибольшего общего делителя (НОД) для двух целых чисел [приложение А], составленное абстрактное синтаксическое дерево представлено на рисунке 1. Также, представлен пример составленного AST кода для функции расчета *n*-го числа Фибоначчи [приложение Б].

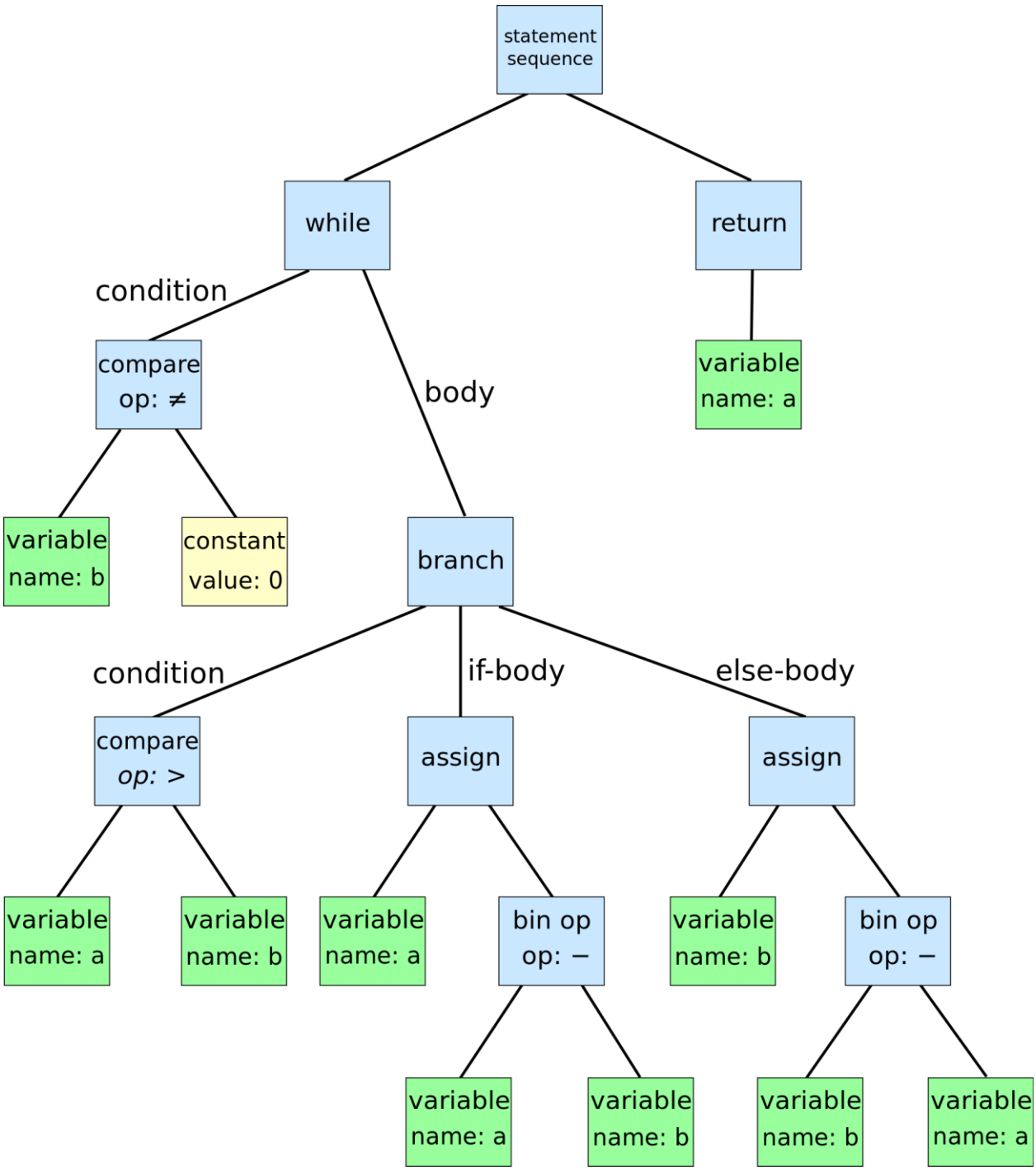

Рисунок 1 – Абстрактное синтаксическое дерево алгоритма нахождения НОД для двух целых чисел

5 Составление графа зависимостей

Для составления PDG необходимо воспользоваться данными, которые были получены на предыдущем шаге составления AST. Так как на предыдущем шаге были получены подграфы для каждой функциональной единицы, то необходимо нормализовать их нумерацию с целью единообразия между функциями перехода. Также представлен пример сформированного графа зависимостей для ранее представленного кода функции расчета n-го числа Фибоначчи [приложение В].

После подготовительной части необходимо привести AST в PDG представление, для этого следует придерживаться следующим правилам:

- каждый подграф представлен в виде отдельной единицы;
    - каждая отдельная единица представляет собой набор из следующих объектов:
        - список вершин в следующем формате: $v\ i\ j$, где v – вершина (если это точка входа в функцию, то i и $j = 0$), i – номер вершины откуда происходит вызов функциональной единицы (точка начала) и j – номер вершины, где находится вызываемая функциональная единицы (точка окончания);
        - список ребер в следующем формате: $e\ i\ j\ k$, где e – ребро, i – вершина откуда начинается передача данных, j – вершина назначения данных, k – будут ли указанные данные использоваться в семантике программы (1 – да, 0 – нет).
    - набор отдельных единиц должны идти строго друг за другом и разделяться подграфами.

6 Решение NP-полной задачи нахождения максимально индуцированного подграфа

После составления PDG необходимо выделить подграфы, которые «повторяются» в двух программах, для этого нужно решить NP-полную задачу нахождения максимально индуцированного подграфа.

В теории графов и теоретической информатике, максимальный общий индуцированный подграф двух графов G_1 и G_2 является графом, который является индуцированным подграфом как G_1, так и G_2, и имеет как можно больше вершин.

Найти этот граф NP-сложно, иными словами – для данной задачи не существует эффективных алгоритмов решения. В данной задаче принятия решения входными данными являются два графа G_1 и G_2 и число k. Задача состоит в том, чтобы определить, имеют ли G_1 и G_2 общий индуцированный подграф с как минимум k вершинами.

Пример результата работы VF2 алгоритма представлен на рисунке 2. Сам алгоритм на конкретном примере представлен на рисунке 3.

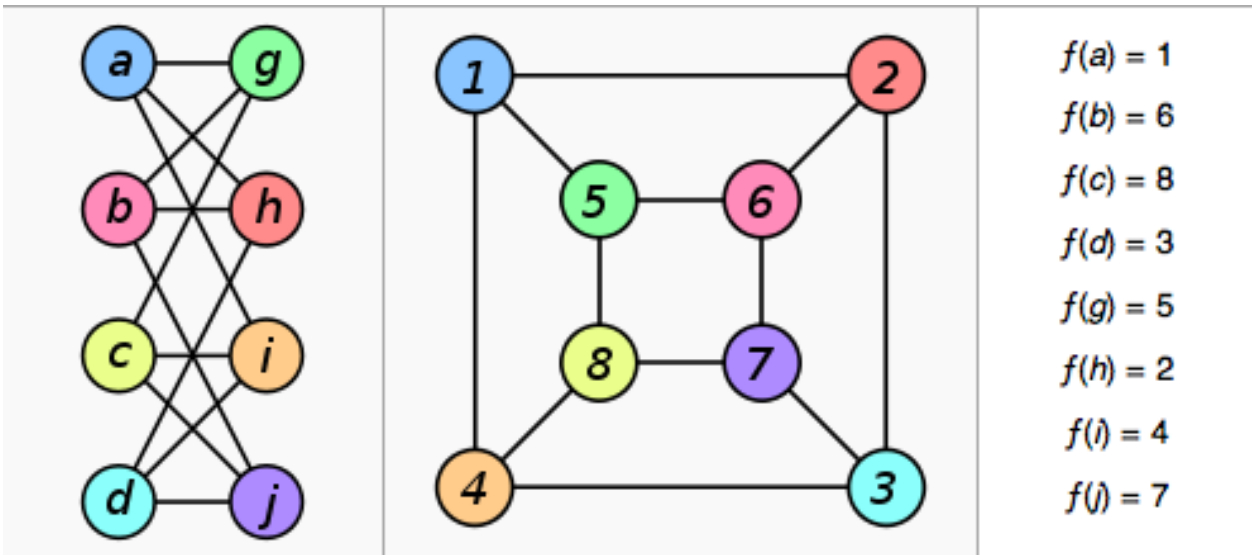

Рисунок 2 – Пример результата работы VF2 алгоритма

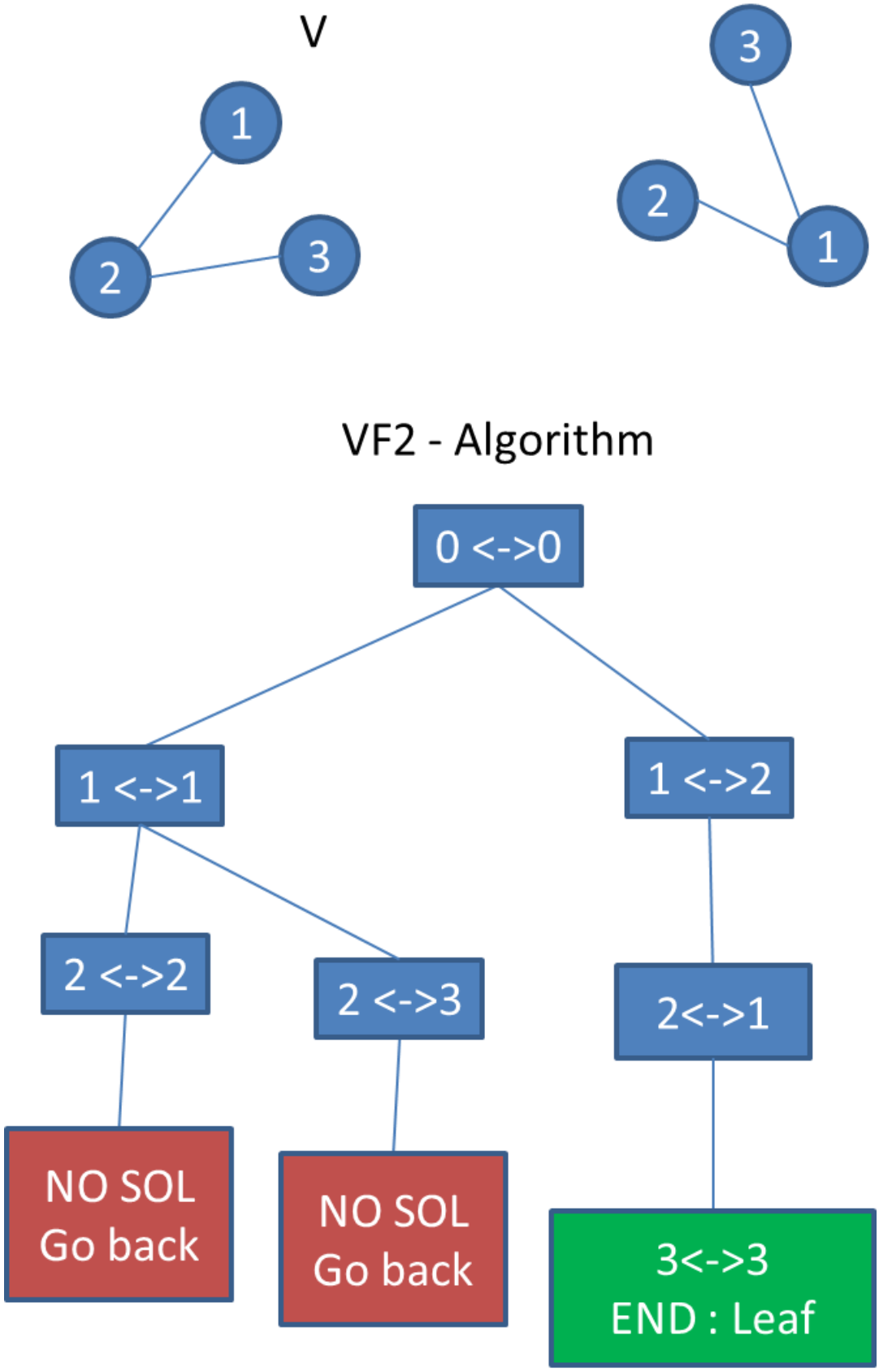

Рисунок 3 – Алгоритм работы VF2 алгоритма

7 Итоги и интерпретация результатов

В данной статье было рассмотрен алгоритм составления графа зависимостей исходного кода на основе абстрактного синтаксического дерева. В ходе смотра было выявлено, что комбинация абстрактного синтаксического дерева и графа зависимостей исходного кода покрывает большую часть типов заимствования фрагментов кода, а именно первые три.

Для примера была реализована программа, демонстрирующая работу данного алгоритма. Пример работы алгоритма для программы вычисляющей *n*-е число Фибоначчи представлен на рисунке 4. Также был рассмотрен более объемный вариант программы – реализация алгоритма Дейкстры, который представлен на рисунке 5.

В выводах данного алгоритма можно выделить следующие аспекты:

- данная комбинация хорошо работает на объемных задачах с большим количеством кода, так как построенные графы имеют больший объем;
- в случаях изменения кода без внесения явных семантических изменений данные фрагменты кода будут проигнорированы на этапе построения AST и PDG.

Результат сравнения

Общая доля заимствований: 92%

Заимствования по функциям:

`main(n)  main(n)  92%`

Заимствования в исходном тексте:

a.py

```
def main(n):
    a = 0
    b = 1
    c = 0
    if n <= 1:
        return 1
    while n > 1:
        c = a + b
        a, b = b, c
        n -= 1
    return c
if __name__ == "__main__":
    print(main(11))
```

b.py

```
def main(n):
    a = 0
    b = 1
    c = 0
    t = 0
    if n <= 1:
        return 1
    while True:
        t = n * n % 65537
        for i in range(15):
            t = t * t % 65537
        if n * t <= 1:
            break
        c = a + b
        a, b = b, c
        n -= 1
    return c
if __name__ == "__main__":
    print(main(11))
```

Рисунок 4 – Алгоритм работы для вычисления числа Фибоначчи

Результат сравнения

Общая доля заимствований: 100%

Заимствования по функциям:

`dijkstra(nodes, distances, current)`	`dijkstra(nodes, distances, current)`	100%
`main()`	`main()`	100%

Заимствования в исходном тексте:

dijkstra.py

```
def dijkstra(nodes, distances, current):
    unvisited = {node: None for node in nodes}
    visited = {}
    current_distance = 0
    unvisited[current] = current_distance
    while True:
        for neighbour, distance in distances[current].items():
            if neighbour not in unvisited:
                continue
            new_distance = current_distance + distance
            if unvisited[neighbour] is None or unvisited[neighbour] > new_distance:
                unvisited[neighbour] = new_distance
        visited[current] = current_distance
        del unvisited[current]
        if not unvisited:
            break
        candidates = [node for node in unvisited.items() if node[1]]
        current, current_distance = sorted(candidates, key=lambda x: x[1])[0]
    return visited
def main():
    nodes = ('A', 'B', 'C', 'D', 'E', 'F', 'G')
    distances = {
        'B': {'A': 5, 'D': 1, 'G': 2},
        'A': {'B': 5, 'D': 3, 'E': 12, 'F': 5},
        'D': {'B': 1, 'G': 1, 'E': 1, 'A': 3},
        'G': {'B': 2, 'D': 1, 'C': 2},
        'C': {'G': 2, 'E': 1, 'F': 16},
        'E': {'A': 12, 'D': 1, 'C': 1, 'F': 2},
        'F': {'A': 5, 'E': 2, 'C': 16}}
    print(dijkstra(nodes, distances, 'B'))
if __name__ == "__main__":
    main()
```

names_noise.py

```
import time
def main():
    nodes = ('A', 'B', 'C', 'D', 'E', 'F', 'G')
    distances = {
        'B': {'A': 5, 'D': 1, 'G': 2},
        'A': {'B': 5, 'D': 3, 'E': 12, 'F': 5},
        'D': {'B': 1, 'G': 1, 'E': 1, 'A': 3},
        'G': {'B': 2, 'D': 1, 'C': 2},
        'C': {'G': 2, 'E': 1, 'F': 16},
        'E': {'A': 12, 'D': 1, 'C': 1, 'F': 2},
        'F': {'A': 5, 'E': 2, 'C': 16}}
    print(dijkstra(nodes, distances, 'B'))
def dijkstra(nodes, distances, current):
    const_0 = 0
    const_1 = 1
    not_visited = {node: None for node in nodes}
    visited = {}
    current_distance = const_0
    not_visited[current] = current_distance
    while True:
        for neighbour, distance in distances[current].items():
            if neighbour not in not_visited:
                continue
            new_distance = current_distance + distance
            k = 0
            for it in range(200):
                k+= it
            if not_visited[neighbour] is None or not_visited[neighbour] > new_distance:
                not_visited[neighbour] = new_distance
        visited[current] = current_distance
        del not_visited[current]
        useless_var = True
        if useless_var:
            time.time()
        if not not_visited:
            break
        candidates = [node for node in not_visited.items() if node[const_1]]
        current, current_distance = sorted(candidates, key=lambda x: x[const_1])[const_0]
    return visited
if __name__ == "__main__":
    main()
```

Рисунок 5 – Алгоритм работы для программы Дейкстры

Приложение А

(Алгоритм нахождение НОД для двух целых чисел)

```
while b ≠ 0
    if a > b
        a := a − b
    else
        b := b − a
return a
```

Приложение Б

(Составленный AST код для функции расчета *n*-го числа Фибоначчи)

```
digraph {
   subgraph sub_0 {
   }
   subgraph sub_1 {
      "f9609cf7-c362-4185-89e1-7a6dbd05c4ef" [label="ENTER def main(n)$1:1"]
      "2f1e8516-e426-46b7-9296-a2e89f1ad5ad" [label="a = 0$2:2"]
      "06ac778b-4fc1-4455-aeac-f5f7aee77f42" [label="b = 1$3:3"]
      "0b80e6ec-d705-4b9c-ae69-5c632f9c5015" [label="c = 0$4:4"]
      "a4259f85-7016-42b5-9824-db162e9c758c" [label="If n <= 1$6:7"]
      "a3f23de0-4d6c-46d5-8fad-c7ddee585a05"
[label="CDGRegionNodeType.then_block$6:7"]
      "5cd6afca-69d6-45cb-95c1-2bbe9d277673" [label="return 1$7:7"]
      "1031f424-cb04-4852-8c88-89b6cd1b57d5"
[label="CDGRegionNodeType.exit$1:1"]
      "32da74ab-9f15-43be-a68b-19be56200eb1" [label="While True$8:8"]
      "49645975-9392-4591-afb7-7a7e329991db"
[label="CDGRegionNodeType.loop_body$8:8"]
      "727b00a4-06a0-4919-aa24-7fca6b56b4b0" [label="t = n * n % 65537$9:9"]
      "0747577e-991f-4896-a6f0-c1075a822e0a" [label="For i in range(15)$10:10"]
      "70269f86-7cf5-440c-b2d1-61a09452c5e0"
[label="CDGRegionNodeType.loop_body$10:10"]
      "26b06203-fce3-4211-9bf3-eb269ad492ff" [label="t = t * t % 65537$11:11"]
      "ae868cf8-75d6-48e0-9e05-5e3615f34b37" [label="If n * t <= 1$12:13"]
      "4fa6e7ec-c08e-4693-97b1-056226761f35"
[label="CDGRegionNodeType.then_block$12:13"]
      "156ea958-00f0-4f4c-b7db-e01a4256de09" [label="Break$13:13"]
      "7c170431-4549-40bf-b0f5-793398219a59" [label="c = a + b$14:14"]
      "abd30d28-63b7-4069-8274-6faa68925ad3" [label="a, b = b, c$15:15"]
      "ddf33ddd-311e-4130-8ab0-54e6b9171b22" [label="n -= 1$16:16"]
      "89903129-af82-4325-afa1-e31e8be8d344" [label="return c$17:17"]
      "f9609cf7-c362-4185-89e1-7a6dbd05c4ef" -> "2f1e8516-e426-46b7-9296-
a2e89f1ad5ad" [style=solid]
      "f9609cf7-c362-4185-89e1-7a6dbd05c4ef" -> "06ac778b-4fc1-4455-aeac-
f5f7aee77f42" [style=solid]
      "f9609cf7-c362-4185-89e1-7a6dbd05c4ef" -> "0b80e6ec-d705-4b9c-ae69-
5c632f9c5015" [style=solid]
      "f9609cf7-c362-4185-89e1-7a6dbd05c4ef" -> "a4259f85-7016-42b5-9824-
db162e9c758c" [style=solid]
      "a4259f85-7016-42b5-9824-db162e9c758c" -> "a3f23de0-4d6c-46d5-8fad-
c7ddee585a05" [label=T style=solid]
      "a3f23de0-4d6c-46d5-8fad-c7ddee585a05" -> "5cd6afca-69d6-45cb-95c1-
2bbe9d277673" [style=solid]
      "5cd6afca-69d6-45cb-95c1-2bbe9d277673" -> "1031f424-cb04-4852-8c88-
89b6cd1b57d5" [style=solid]
      "f9609cf7-c362-4185-89e1-7a6dbd05c4ef" -> "32da74ab-9f15-43be-a68b-
19be56200eb1" [style=solid]
      "49645975-9392-4591-afb7-7a7e329991db" -> "32da74ab-9f15-43be-a68b-
19be56200eb1" [style=solid]
      "32da74ab-9f15-43be-a68b-19be56200eb1" -> "49645975-9392-4591-afb7-
7a7e329991db" [label=T style=solid]
      "49645975-9392-4591-afb7-7a7e329991db" -> "727b00a4-06a0-4919-aa24-
7fca6b56b4b0" [style=solid]
      "49645975-9392-4591-afb7-7a7e329991db" -> "0747577e-991f-4896-a6f0-
c1075a822e0a" [style=solid]
      "70269f86-7cf5-440c-b2d1-61a09452c5e0" -> "0747577e-991f-4896-a6f0-
c1075a822e0a" [style=solid]
      "0747577e-991f-4896-a6f0-c1075a822e0a" -> "70269f86-7cf5-440c-b2d1-
61a09452c5e0" [label=Cont style=solid]
      "70269f86-7cf5-440c-b2d1-61a09452c5e0" -> "26b06203-fce3-4211-9bf3-
```

```
eb269ad492ff" [style=solid]
      "49645975-9392-4591-afb7-7a7e329991db" -> "ae868cf8-75d6-48e0-9e05-
5e3615f34b37" [style=solid]
      "ae868cf8-75d6-48e0-9e05-5e3615f34b37" -> "4fa6e7ec-c08e-4693-97b1-
056226761f35" [label=T style=solid]
      "4fa6e7ec-c08e-4693-97b1-056226761f35" -> "156ea958-00f0-4f4c-b7db-
e01a4256de09" [style=solid]
      "49645975-9392-4591-afb7-7a7e329991db" -> "7c170431-4549-40bf-b0f5-
793398219a59" [style=solid]
      "49645975-9392-4591-afb7-7a7e329991db" -> "abd30d28-63b7-4069-8274-
6faa68925ad3" [style=solid]
      "49645975-9392-4591-afb7-7a7e329991db" -> "ddf33ddd-311e-4130-8ab0-
54e6b9171b22" [style=solid]
      "f9609cf7-c362-4185-89e1-7a6dbd05c4ef" -> "89903129-af82-4325-afa1-
e31e8be8d344" [style=solid]
      "89903129-af82-4325-afa1-e31e8be8d344" -> "1031f424-cb04-4852-8c88-
89b6cd1b57d5" [style=solid]
      "156ea958-00f0-4f4c-b7db-e01a4256de09" -> "89903129-af82-4325-afa1-
e31e8be8d344" [style=solid]
      "f9609cf7-c362-4185-89e1-7a6dbd05c4ef" -> "1031f424-cb04-4852-8c88-
89b6cd1b57d5" [style=solid]
      "a4259f85-7016-42b5-9824-db162e9c758c" -> "f9609cf7-c362-4185-89e1-
7a6dbd05c4ef" [style=dotted]
      "727b00a4-06a0-4919-aa24-7fca6b56b4b0" -> "f9609cf7-c362-4185-89e1-
7a6dbd05c4ef" [style=dotted]
      "727b00a4-06a0-4919-aa24-7fca6b56b4b0" -> "ddf33ddd-311e-4130-8ab0-
54e6b9171b22" [style=dotted]
      "26b06203-fce3-4211-9bf3-eb269ad492ff" -> "26b06203-fce3-4211-9bf3-
eb269ad492ff" [style=dotted]
      "26b06203-fce3-4211-9bf3-eb269ad492ff" -> "727b00a4-06a0-4919-aa24-
7fca6b56b4b0" [style=dotted]
      "ae868cf8-75d6-48e0-9e05-5e3615f34b37" -> "f9609cf7-c362-4185-89e1-
7a6dbd05c4ef" [style=dotted]
      "ae868cf8-75d6-48e0-9e05-5e3615f34b37" -> "ddf33ddd-311e-4130-8ab0-
54e6b9171b22" [style=dotted]
      "ae868cf8-75d6-48e0-9e05-5e3615f34b37" -> "26b06203-fce3-4211-9bf3-
eb269ad492ff" [style=dotted]
      "ae868cf8-75d6-48e0-9e05-5e3615f34b37" -> "727b00a4-06a0-4919-aa24-
7fca6b56b4b0" [style=dotted]
      "7c170431-4549-40bf-b0f5-793398219a59" -> "abd30d28-63b7-4069-8274-
6faa68925ad3" [style=dotted]
      "7c170431-4549-40bf-b0f5-793398219a59" -> "06ac778b-4fc1-4455-aeac-
f5f7aee77f42" [style=dotted]
      "7c170431-4549-40bf-b0f5-793398219a59" -> "2f1e8516-e426-46b7-9296-
a2e89f1ad5ad" [style=dotted]
      "abd30d28-63b7-4069-8274-6faa68925ad3" -> "7c170431-4549-40bf-b0f5-
793398219a59" [style=dotted]
      "abd30d28-63b7-4069-8274-6faa68925ad3" -> "06ac778b-4fc1-4455-aeac-
f5f7aee77f42" [style=dotted]
      "abd30d28-63b7-4069-8274-6faa68925ad3" -> "abd30d28-63b7-4069-8274-
6faa68925ad3" [style=dotted]
      "ddf33ddd-311e-4130-8ab0-54e6b9171b22" -> "f9609cf7-c362-4185-89e1-
7a6dbd05c4ef" [style=dotted]
      "ddf33ddd-311e-4130-8ab0-54e6b9171b22" -> "ddf33ddd-311e-4130-8ab0-
54e6b9171b22" [style=dotted]
      "89903129-af82-4325-afa1-e31e8be8d344" -> "7c170431-4549-40bf-b0f5-
793398219a59" [style=dotted]
      "89903129-af82-4325-afa1-e31e8be8d344" -> "0b80e6ec-d705-4b9c-ae69-
5c632f9c5015" [style=dotted]
   }
}
```

Приложение В

(Сформированный граф зависимостей для кода функции расчета *n*-го числа Фибоначчи)

```
t
v 0 0
v 1 6
v 2 6
v 3 6
v 4 2
v 5 3
v 6 8
v 7 10
v 8 4
v 9 3
v 10 9
v 11 11
v 12 7
v 13 8
e 0 1 0
e 0 2 0
e 0 3 0
e 0 4 0
e 0 7 0
e 0 8 0
e 0 13 0
e 0 4 1
e 0 8 1
e 0 12 1
e 4 0 1
e 4 5 0
e 5 6 0
e 6 7 0
e 8 0 1
e 8 9 0
e 8 12 1
e 9 8 0
e 9 10 0
e 9 11 0
e 9 12 0
e 10 1 1
e 10 2 1
e 10 11 1
e 11 2 1
e 12 0 1
e 12 8 1
e 13 3 1
e 13 7 0
e 13 10 1
t
```

СРАВНЕНИЕ КРОССПЛАТФОРМЕННОГО И НАТИВНОГО РЕШЕНИЯ ПРИ РАЗРАБОТКЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ В СФЕРЕ ТУРИЗМА

Проскурина Софья Алексеевна, группа ПИ-20-1

Гаев Леонид Витальевич, канд. техн. наук, доцент кафедры АСУ

Аннотация. В статье рассматриваются сильные и слабые стороны нативного и кроссплатформенного подходов в разработке мобильных приложений. Проводится анализ наиболее популярных фреймворков для кроссплатформенной разработки и определяется наиболее подходящий подход для разработки приложения в сфере туризма.

Ключевые слова: фреймворк, мобильное приложение, кроссплатформенная разработка, Flutter.

Сфера мобильной разработки активно развивается. В настоящее время многие компании заинтересованы в создании продуктов под две наиболее популярные мобильные операционные системы — Android и iOS. В частности, компании, работающие в сфере туризма, для решения своих задач все чаще принимают решения о разработке персональных мобильных приложений.

В рамках договора подряда была поставлена задача по разработке приложения для продвижения и популяризации туризма Липецкой области в сжатые сроки (т. к. было важно успеть к следующему туристическому сезону).

Мобильное приложение должно быть доступно максимальному числу пользователей, оно должно быть реализовано как для ОС Android, так и для ОС iOS. Поэтому до начала разработки необходимо было сделать выбор между нативным и кросплатформенным подходами.

Нативный подход заключается в написании независимых приложений под каждую из двух платформ на разных языках. Так, для разработки на ОС Android используют языки kotlin и java, а для ОС iOS применяют swift и Objective-C.

Данный подход является надежным и проверенным, его основные плюсы — это высокая производительность, большое количество стабильных библиотек

для решения многих задач, интерфейс приложений, соответствующий особенностям системы, и широкий рынок разработчиков [1].

Однако у нативного подхода есть и свои минусы, которые делают его использование в определенных ситуациях нецелесообразным. Так как разработка приложений на системы Android и iOS ведется параллельно и использование общей кодовой базы невозможно, увеличивается время разработки и количество дублирования кода, связанного с бизнес-логикой, которая для приложений на разные платформы остается единой. Кроме того, из-за большего объема работы и требования наличия специалистов, обладающих навыками разработки под две разные платформы, нативная разработка выходит дороже. [1]

Альтернативой нативному подходу является кроссплатформенная разработка. Она подразумевает написание единой кодовой базы для Android и iOS. На данный момент наиболее популярными фреймворками для кроссплатформенной разработки являются React Native, созданный Facebook, и Flutter, являющийся проектом Google. [2] В React Native разработка основана на JavaScript, а в Flutter используется язык Dart. Кроссплатформенная разработка является более молодым подходом, однако уже существуют крупные проекты, применяющие его. Так, React Native применяли в работе такие крупные компании, как Uber, Microsoft, Discord, а продукты на Flutter уже разрабатывали в Яндекс и SpaceX.

Основными преимуществами кроссплатформенных решений являются высокая скорость и низкая стоимость разработки, так как для Android и iOS версий разрабатывается и используется единая кодовая база. [3]

Но в использовании данного подхода также есть свои недостатки и риски. Так как кроссплатформенные технологии являются более молодыми Open Source решениями, нужно обратить внимание на то, что в них более часто могут возникать ошибки, выбор проверенных библиотек не так велик по сравнению с нативными решениями, и более высок риск, что поддержка как используемых библиотек, так и самих фреймворков в целом может прекратиться. Но, изучив результаты опроса разработчиков «Stack Overflow Developer Survey 2021», в

котором Flutter вошел в топ наиболее популярных фреймворков, можно сделать вывод, что данная технология активно развивается, и минусы, описанные выше, будут все реже проявлять себя [4].

Еще одним специфическим недостатком разработки на кроссплатформе является возвращение к написанию нативного кода в случае, если требуется разработать два разных дизайна для Android и iOS версий или написать модуль, использующий уникальные функции конкретной платформы. Однако даже в этом случае дублировать бизнес-логику не надо, что дает преимущество.

Также при проведении работ над приложением в сфере туризма необходимо изучить наличие актуальных поддерживаемых библиотек, предоставляющих возможность реализовать нестандартный запланированный функционал. В данном случае было важно наличие хороших решений для работы с геолокацией, картами и навигацией. Как нативные, так и кросспатформенные решения обладают всем необходимым функционалом для работы в данной сфере. В частности, фреймворк Flutter позволяет работать с геолокацией, картами и навигацией с помощью библиотек geolocator, flutter_map и flutter_mapbox_navigation соответственно.

Кроме того, была изучена информация о довольно крупных проектах на Flutter, задачи которых также были тесно связаны с картами.

Так, перед компанией Яндекс еще в 2020 году встала задача разработать приложение Таксометр на платформу iOS, с помощью которого водители Яндекс-такси могли бы принимать заказы. Данную задачу они успешно выполнили с использованием Flutter, который оказался более выгоден в сравнение с React Native и нативным iOS [5]. Еще один пример - это приложение для сети аптек Ригла, которое включает в себя функции заказа и доставки товара по разным сценариям [6].

Данные примеры показывают, что работа с геолокацией, навигацией и картами в кроссплатформенном фреймворке Flutter уже проверена и готова к использованию.

Изучив сильные и слабые стороны нативного и кроссплатформенного подхода в разработке мобильных приложений, можно сделать вывод, что нет универсального и идеального решения. Каждый подход эффективен в определенных условиях и ситуациях. Для разработки приложения в сфере туризма с единым интерфейсом для Android и iOS в сжатые сроки более рационально использовать кроссплатформенный фреймворк Flutter.

Список литературы

1) Мобильная разработка: Cross-platform или Native. - URL: https://habr.com/ru/company/agima/blog/586092/ (дата обращения 10.03.2022).

2) Сравнение React Native и Flutter с точки зрения их применения в реальных проектах - URL: https://habr.com/ru/company/ruvds/blog/478322/ (дата обращения 10.03.2022).

3) Кроссплатформенная разработка мобильных приложений на основе набора средств разработки Flutter. - Сибирский государственный университет науки и технологий имени академика М.Ф. Решетнева, 2020. - 30 с.

4) Stack Overflow Developer Survey 2021. - URL: https://insights.stackoverflow.com/survey/2021#overview (дата обращения 10.03.2022).

5) Опыт выбора кроссплатформенной технологии для разработки приложения. Доклад Яндекса. - URI: https://habr.com/ru/company/yandex/blog/528322/ (дата обращения 10.03.2022).

6) Разработка приложения для аптеки Ригла. - URL: https://surf.ru/cases/rigla/ (дата обращения 10.03.2022).

ПЕРСПЕКТИВЫ РАЗВИТИЯ ЭКСПЕРТНЫХ СИСТЕМ

Трунов Павел Романович, группа ПИ-18-1

Грунау Герман Юрьевич, группа АИ-18-1

Аннотация: в современных информационных технологиях большое место занимает искусственный интеллект. В этой статье рассматривается одно из ключевых направлений этой отрасли – экспертные системы, а также его понятие, применение и перспективы развития на сегодняшний день.

Ключевые слова: экспертные системы, искусственный интеллект

Современное информационное общество не стоит на месте и развивается головокружительными темпами в различных областях науки. Одним из перспективных направлений является искусственный интеллект. Не секрет, что ученые достигли небывалых высот в робототехнике, генерации и распознавании речи, разработке интерфейсов на естественном языке, экспертных системах и многих других областях. В этой статье мы более подробно рассмотрим экспертные системы.

Экспертная система - система, созданная с помощью информационных технологий и баз данных, целью которой является замена экспертов в различных сферах деятельности. Такая система добивается более высокой производительности за счет перебора огромного количества вариантов при выборе решения, основываясь на высококачественном опыте профессионалов этой сферы. Основным преимуществом экспертной системы является накопление и сохранение знаний. Это позволяет анализировать воздействие большого объема ранее неизвестных факторов, оценивая их при построении логики решения, добавляя возможности прогноза.

Экспертные системы появились в начале 60-х годов и использовались активно только учёными при разработках и различного рода исследованиях. Однако спустя 20 с лишним лет они начали находить себе место во многих коммерческих проектах. Многие методы, которые были разработаны ранее, нашли свое применение именно в этих системах. Примером таких методов можно считать: логический вывод, эвристический поиск, распознавание предложений на естественном языке [1].

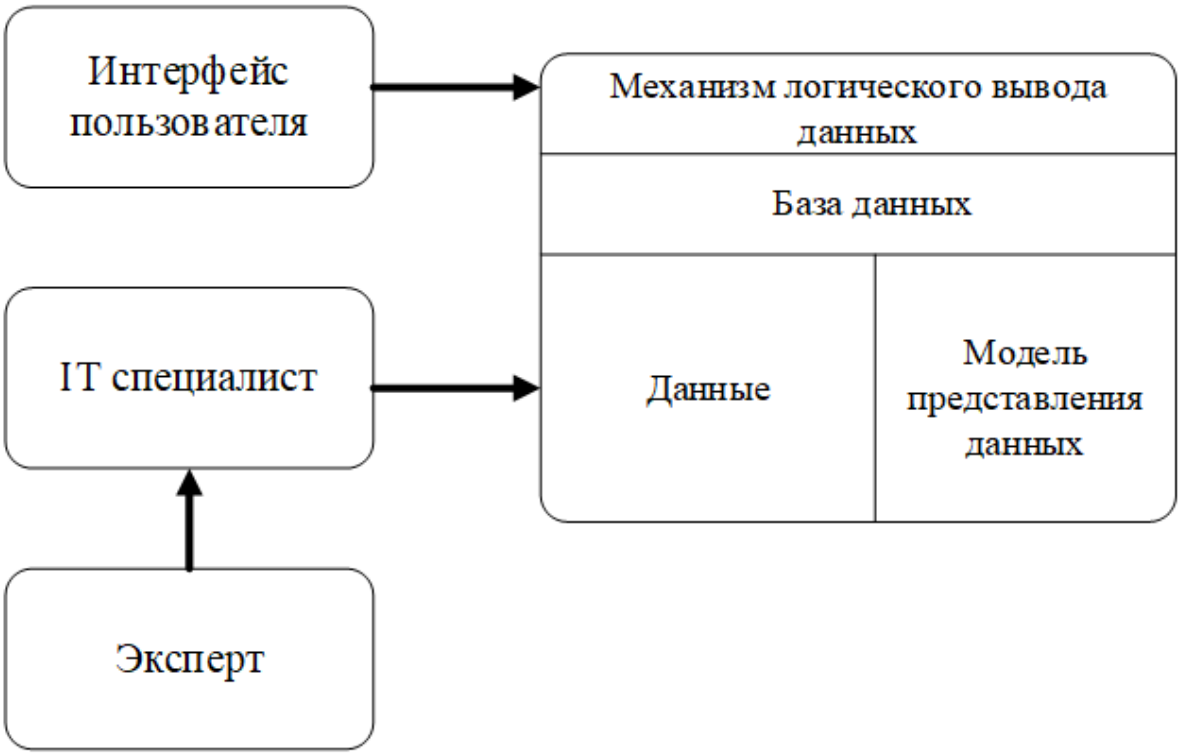

Рисунок 1 – Структура ЭС

С современном мире информационных технологий огромной популярностью пользуются системы искусственного интеллекта, в особенности нейронные сети, имитирующие естественный интеллект и применяемые в задачах дискриминантного анализа и прогнозирования. Внутреннее представление при решении задач с использованием нейросетей поддаётся анализу с большим трудом или не поддаётся вообще. Это сравнимо с процедурными знаниями человека, находящимися вне области осознания, например, умение плавать. Здесь важна быстрота реакции и результат. Мозг отдаёт команды телу двигаться в воде так, чтобы не утонуть, и эти команды обходят наше сознание. Но существуют задачи, требуемые последовательного объяснения алгоритма и логического вывода, например, решение математических уравнений. Для таких задач подходят экспертные системы засчёт их умения «объяснить» полученный результат.

Количество экспертных систем в наше время растёт с каждым днём и уже сейчас приравнивается к десяткам тысяч. Они находят широкое применение в множестве сфер: медицины, авиации, географических систем, машиностроении и так далее. Экспертные системы позволяют более эффективно проводить обработку и анализ данных на компьютере лицам, не имеющим достаточного опыта в данной сфере [2].

Также большое количество компаний занимается внедрением ЭС в различные сферы жизнедеятельности. Как было ранее сказано в статье, существуют задачи, требующие различного рода подходы для их решения и задействования разных отделов мозга. Одним из перспективных направлений

развития экспертных систем является их совмещение с нейронными сетями для более полной эмуляции человеческого интеллекта. Разработка таких систем, в которых нейронные сети и экспертные системы будут взаимно дополнять друг друга [3], станет большим научным достижением в IT сфере и может стать ступенью на пути к достижению цели парадигмы Software 2.0, которую сформулировал руководитель отдела ИИ компании Tesla Андрей Карпатый. Идея «нового программного обеспечения» заключается в том, чтобы встроить в компьютер обучаемую модель и сделать его обучаемым: модель будет сама формировать исполняемый код, а человеку остаётся лишь роль наблюдателя. Важнейшим компонентом такого подхода является перманентное тестирование и управление человеко-машинным процессом разработки. В таком случае человек может сосредоточиться на сути задачи, а не на написании программного кода. От него будут требоваться только опыт и знания в предметной области, что схоже с подходом в экспертных системах.

На сегодняшний день реализация нового подхода к разработке ПО на основе модулей искусственного интеллекта является только гипотезой, а экспертные системы пока что – лишь задатки на пути к её воплощению.

Список литературы

1.Пырнова О.А. Зарипова Р.С Перспективы развития экспертных систем//Информационные технологии в строительных, социальных и экономических системах. 2020. №3 (21). С 55-57.

2. Александровская Л.А. Забуруннова А.Е. Практика применения экспертных систем в землеустроительных системах автоматизированного проектирования. // Материалы XVII Всероссийской научно-практической конференции. Новочеркасск 2020. С 3-5.

3. Шумков Е.А. Фреймовые экспертные системы с использованием нейронных сетей// Политематический сетевой электронный научный журнал кубанского государственного аграрного университета 2019 №154. С 226-232.

PROSPECTS FOR THE DEVELOPMENT OF EXPERT SYSTEMS

TRUNOV PAVEL ROMANOVICH
student of group PI-18-1
GRUNAU GERMAN YURIEVICH
student of group AI-18-1
Lipetsk State Technical University

Summary: artificial intelligence occupies a large place in modern information technologies. This article discusses one of the key areas of this industry – expert systems, as well as its concept, application and prospects for development today.

Keywords: expert system, artificial intelligence