Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б10-мм

Разработка инфраструктуры для автоматизации проверки задач по курсу «Теория формальных языков»

ПОНОМАРЕВ Николай Алексеевич

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
доцент кафедры системного программирования, к. ф.-м. н., Григорьев С. В.

Санкт-Петербург
2024

Оглавление

Введение

Сложно представить курс по информатике без практических заданий: они позволяют лучше усвоить и закрепить теоретический материал. К сожалению, на проверку заданий у преподавателя может уходить довольно много времени, а контрольные цифры приема на специальности, связанные с информационными технологиями, увеличиваются каждый год [20]. Некоторые преподаватели решают данную проблему, привлекая других проверяющих, например старшекурсников, но данный подход не является идеальным: по наблюдениям кафедры системного программирования на каждые 4–5 человек нужен один проверяющий. Поиск такого количества людей является непростой задачей.

Курс по теории формальных языков является важной частью обучения IT-специалиста. Формальные языки находят своё применение как в теоретических областях, таких как математическая логика [15, 3], так и в прикладных: при написании парсеров [9, 19], в графовых базах данных [12, 13], в биоинформатике [4, 5].

На математико-механическом факультете СПбГУ так же читается курс по теории формальных языков. В нём имеется 12 практических заданий. В большинстве задач требуется реализовать нетривиальный алгоритм: это вызывает сложности как у студентов, так как требуется аккуратность при выполнении, так и у проверяющих, ведь для проверки необходимо разобраться в коде студента. Ранее при сдаче задания студенты обязаны были предоставлять рукописные тесты к каждой задаче. Тем не менее в данной системе со временем нашелся серьёзный недостаток: тестовое покрытие не всегда описывало все крайние случаи, поэтому иногда студентам удавалось сдать не корректные решения.

При более тщательном изучении задач, появилась идея о том, что алгоритмические задачи в курсе хорошо поддадутся property-based тестированию. Таким образом было принято решение автоматизировать проверку задач, используя property-based тесты. Кроме того, в курсе использовались старые версии библиотек, а также стандартный менеджер пакетов Python. Данные проблемы тоже было решено исправить.

1. Постановка задачи

Целью работы является разработка инфраструктуры для автоматизации проверки задач по курсу «Теория формальных языков». Для её выполнения были поставлены следующие задачи.

1. Изучить способы автоматического тестирования учебных задач (раздел 2).
2. Создать базовую инфраструктуру для автоматического тестирования (раздел 4.1).
3. Произвести проверку автоматических тестов, путём реализации решений (раздел 4.2).
4. Исправить накопленный технический долг (раздел 4.3).
   (a) Сменить систему управления зависимостями.
   (b) Обновить зависимости.

2. Обзор способов тестирования заданий

Автоматические тесты можно разделить по нескольким критериям: по видимости, по способу реализации, по способу сдачи решений.

2.1. Классификация по видимости

По видимости тесты можно разделить на открытые и закрытые.

Открытые тесты явно доступны студентам. Инфраструктура для такого способа проста, так как не требуются сложные системы для выдачи тестов. Тем не менее в такой системе есть шанс появления частичных решений — таких решений, которые полностью проходят имеющиеся тесты, но не решают задачу полностью.

Закрытые тесты каким-либо образом скрыты от студента. Такие тесты сложнее с точки зрения инфраструктуры, но в такой системе сложнее «подгонять» решение к тестам.

2.2. Классификация по способу реализации

По способу реализации тесты бывают ручные и property-based.

Ручные тесты проще для реализации. Хотя имеют и минусы: их нужно написать в достаточном объеме, чтобы покрыть различные случаи. Кроме того, они так же подвержены частичному решению.

Property-based тесты проверяют известные свойства программы. Тестовые данные генерируются при каждом запуске, тем самым обеспечивая широкое покрытие случаев. Но здесь кроется проблема: если одна тестируемая функция тестируется другой тестируемой (не эталонной) функцией, то можно получить работающие тесты при некорректном поведении.

В качестве универсального решения можно совмещать оба похода. Тогда за базовую корректность отвечают ручные тесты, а за полноту реализации — property-based тесты.

2.3. Классификация по способу сдачи решений

Организация автоматических тестов сильно зависит от способа сдачи решений.

Одним из классических способов реализации учебных курсов является система управления обучением (Learning Management System). Классическими примерами являются системы Blackboard[1] и Moodle[2]. Данные системы обладают достаточной гибкостью, чтобы поддерживать различные способы сдачи решений: начиная от сдачи архива с исходным кодом, заканчивая использованием специальных плагинов для сдачи заданий в формате контеста[3]. Описать все возможные способы в данном случае довольно трудно.

Для предметов, связанных с программированием, часто используется подход сдачи заданий с помощью Pull Request (далее — PR) на таких сервисах, как GitHub или GitLab. Процесс работы студента выглядит следующим образом: студент должен создать fork (или новый репозиторий) и пригласить туда проверяющего; далее, на каждую задачу студент создает ветку; после выполнения задания студент открывает PR; проверяющий проверяет решение, прохождение тестов, и либо засчитывает задание, либо просит исправить замечания.

Сервис GitHub Classroom можно рассматривать как объединение идей LMS и сдачи с помощью PR. Он позволяет упростить процесс создания репозиториев и приглашения туда проверяющего; добавить возможность автоматического выставления оценок с помощью тестов; кроме того, сервис автоматические создает PR для удобного отслеживания хода работы и комментирования кода.

[1] https://www.blackboard.com/ — официальный сайт Blackboard (дата доступа: 28 мая 2024 г.).

[2] https://moodle.org — официальный сайт Moodle (дата доступа: 28 мая 2024 г.).

[3] https://coderunner.org.nz/ — официальный сайт плагина Moodle CodeRunner (дата доступа: 28 мая 2024 г.).

3. Обзор используемых технологий

Данный курс читается не первый год, поэтому технологии во многом были зафиксированы.

- Python[4] — основной язык программирования.
- Pytest[5] — библиотека для тестирования.
- CFPQ_Data[6] — библиотека-датасет графов и грамматик.
- Pyformlang [17] — библиотека для работы с грамматиками, регулярными выражениями и различными конечными автоматами.
- NetworkX [10] — библиотека для работы с графами.
- SciPy [18] — библиотека для работы с разреженными матрицами.
- ANTLR4 [16] — инструмент для генерации парсеров.

В процессе работы появилась необходимость генерации слов из языка, описываемого по грамматикой для ANTLR4. Для этого использовалась библиотека Grammarinator. Задача генерации слов из языка встречается не часто, поэтому Grammarinator является единственным существующим решением [7].

[4] https://www.python.org/ — официальный сайт Python (дата доступа: 27 мая 2024 г.).

[5] https://pytest.org/ — официальный сайт библиотеки Pytest (дата доступа: 27 мая 2024 г.).

[6] https://formallanguageconstrainedpathquerying.github.io/CFPQ_Data/ — официальный сайт библиотеки CFPQ_Data (дата доступа: 27 мая 2024 г.).

4. Реализация

После обсуждения результатов обзора были зафиксированы следующие требования к тестам:

- Тесты должны быть открытыми.
- Тесты должны быть property-based.
- Задачи должны сдаваться с помощью PR.
- Имена и сигнатуры требуемых функций и классов известны заранее.
- Архитектуру проекта и связь между модулями студенты определяют сами.

4.1. Инфраструктура тестирования

Основная проблема при написании тестов к курсу — необходимость запуска тестов только для реализованных студентом заданий.

Одним из самых простых решений является создание отдельного репозитория для каждой задачи. Такой подход крайне неудобен для нашего курса, так как создавать 12 репозиториев для заданий, которые зависят друг от друга, а затем добавлять в каждый проверяющего, будет слишком неудобно.

В качестве альтернативного решения можно запускать только необходимые тесты, например указывая их как параметр для фреймворка

Листинг 1: Код отвечающий за запуск или пропуск теста, в зависимости от наличия или отсутствия решения задачи

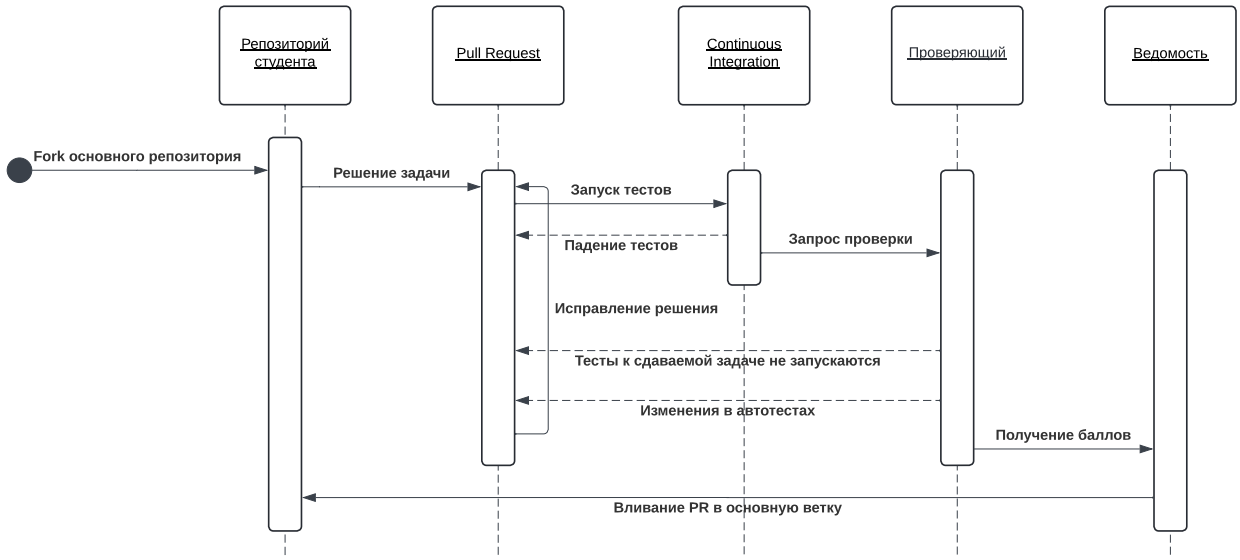
```
try:
    from project.taskN import some_func
except ImportError:
    pytestmark = pytest.mark.skip("Task N is not ready to test!")
```

тестирования. Здесь не требуется создавать много репозиториев, и после выполнения каждой задачи студенту достаточно будет включить необходимые тесты. Но кроме этого потребуется исправить пути импорта тестируемых модулей.

Наш подход основывается на том, что пути импорта исправлять нужно всегда и это может служить минимальным требованием для запуска тестов. Python — интерпретируемый язык, поэтому поиск необходимых модулей происходит во время исполнения. Это позволяет прямо во время исполнения узнать существует ли решение нужной задачи, и в зависимости от этого запустить или пропустить тесты. Достаточно попробовать импортировать нужные объекты, и, в случае получения `ImportError`, выставить флаг пропуска всех тестов в модуле (см. листинг 1).

Вместе с требованиями и способом запуска текстов, процесс сдачи заданий иллюстрирует рисунок 1.

Рис. 1: Процесс сдачи заданий студентом



4.2. Проверка автоматических тестов

Написанием самих тестов занимался мой товарищ, Ефим Кубышкин. Моей задачей была проверка тестов, путём реализации задач из курса.

Задачи в курсе разделены на три крупных блока.

1. Регулярные языки.

2. Контекстно-свободные запросы.

3. Синтаксис и семантика.

И обычно предполагают либо создание обертки над библиотечными функциями, либо самостоятельную реализацию алгоритма.

В первом блоке предлагается реализовать два алгоритма регулярных запросов к графу: тензорный алгоритм и алгоритм на основе поиска в ширину. Во втором блоке необходимо реализовать четыре алгоритма исполнения контекстно-свободных запросов: алгоритм Хеллингса, матричный алгоритм Рустама Азимова, тензорный алгоритм, алгоритм на основе GLL. Третий блок предполагает работу с модельным языком запросов к графам: написание парсера с использованием ANTLR4, реализацию вывода типов, а затем интерпретатора.

Задача №1 вводная, в ней необходимо создать fork, написать две функции и тесты к ним для знакомства с библиотеками, а также настроить CI для запуска тестов. Автотесты к ней не были реализованы.

В задаче №2 необходимо научиться преобразовывать регулярное выражение в ДКА и граф в формате NetworkX в НКА, используя библиотеку Pyformlang. Обе функции по сути являются обертками на цепочками функций из библиотеки.

В задаче №3 требовалось реализовать класс, описывающий конечный автомат в формате разреженной матрицы из SciPy, а затем тензорный алгоритм выполнения регулярных запросов к графам [14].

В задаче №4 необходимо было реализовать алгоритм выполнения регулярных запросов к графу на основе обхода в ширину [8].

Задача №5 подразумевает постановку экспериментов, поэтому не тестировалась.

В задаче №6 необходимо было реализовать функцию преобразования контекстно-свободной грамматики в ослабленную нормальную форму Хомского (ОНФХ) и алгоритм исполнения контекстно-свободных запросов Хеллингса [11].

В задаче №7 требовалось реализовать матричный алгоритм исполнения контекстно-свободных запросов Рустама Азимова [1].

Листинг 2: Пример объявления грамматики, задающей язык $a^n b^n$, в модельном языке

```
let a = ("a" . b) | "a" ^ [0..0]
let b = a . "b"
```

В задаче №8 необходимо было сначала научиться работать с Recursive State Machine (RSM) из Pyformlang путём реализации функций конвертации различных представлений КС грамматик в RSM. А затем реализовать тензорный алгоритм исполнения контекстно-свободных запросов [2, 14].

В задаче №9 требовалось реализовать алгоритм исполнения контекстно-свободных запросов на основе Generalized LL [6].

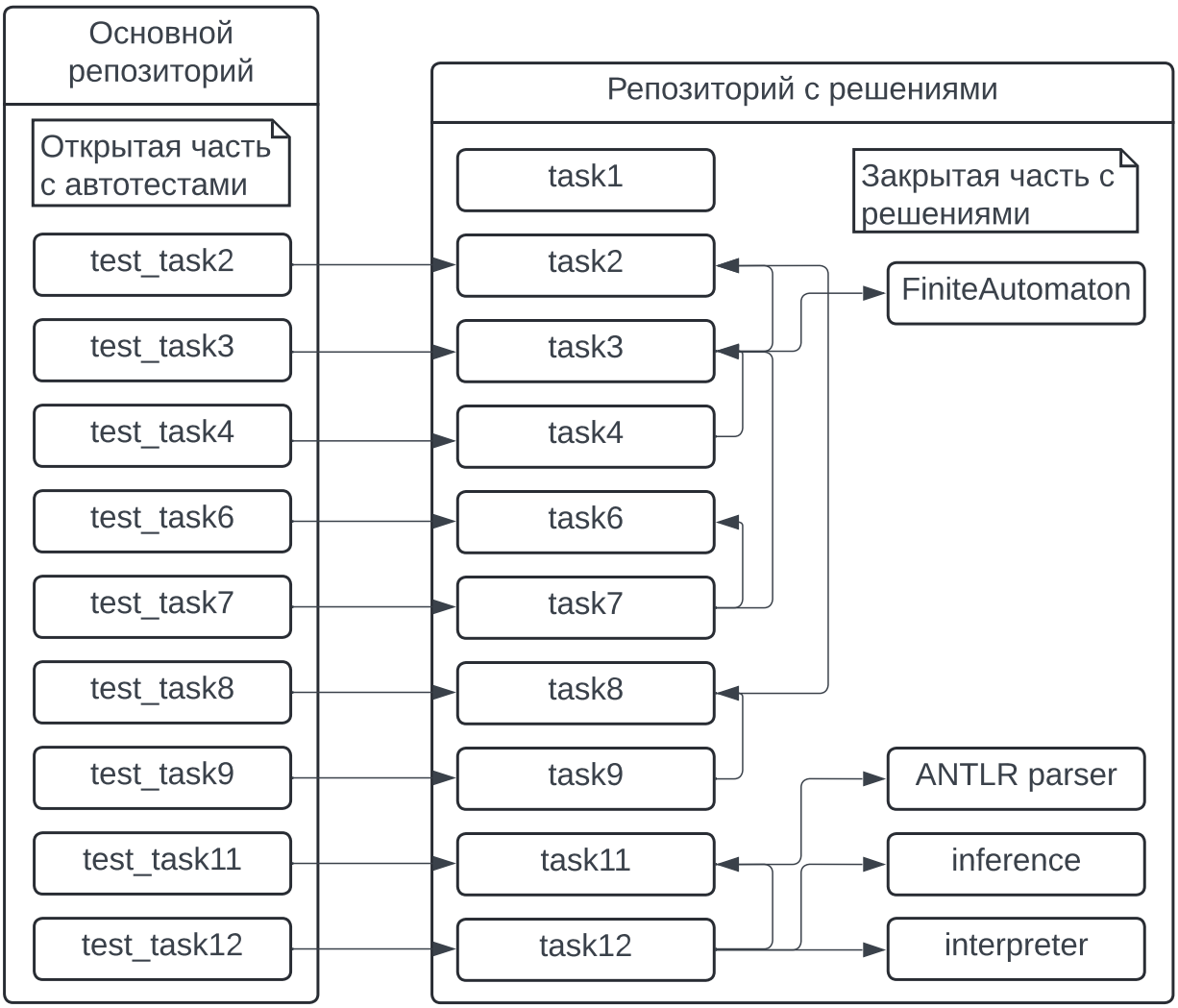
Задача №10 подразумевает постановку экспериментов, поэтому не тестировалась.

В задаче №11 необходимо было написать парсер модельного языка запросов к графам с использованием ANTLR4, а так же функцию подсчёта количества узлов и функцию преобразования дерева разбора в текст программы.

В задаче №12 требовалось реализовать механизм вывода типов и интерпретатор для языка из задания №11. Эта задача достаточно сильно отличается от предыдущих, так как алгоритм вывода типов требуется придумать самому. В модельном ЯП не присутствует полиморфизм, что облегчает задачу. Однако присутствует возможность ссылаться на ещё не объявленные переменные, так как большая часть объявлений — правила КС грамматики в регулярной форме (см. листинг 2). В таком случае необходимо отдельно строить граф зависимостей объявлений друг от друга.

Реализованные задачи выкладывались в приватный репозиторий. В нём был настроен CI, который запускал автотесты из `main` ветки основного репозитория на эталонных решениях. Общая структура решения продемонстрирована на рисунке 2.

Рис. 2: Структура решения



4.3. Обновление используемых инструментов

При реализации одного из заданий появилась необходимость в использовании транзитивного замыкания графа. Однако при запуске выяснилось, что установленная версия NetworkX не поддерживает транзитивное замыкание для класса `MultiDiGraph`. Дальнейшее исследование показало, что не смотря на указание более свежей версии NetworkX, pip — стандартный пакетный менеджер для Python — выбирал более старую версию, так как её требовал пакет CFPQ_Data.

В течение семестра для решения этой проблемы был создан Pull Request в библиотеку CFPQ_Data с обновлением всех её зависимостей. Поддержкой библиотеки занимается старший преподаватель кафедры

ИАС Азимов Рустам Шухратуллович, поэтому Pull Request был принят быстро, после чего была опубликована новая версия библиотеки.

Дабы избежать подобной проблемы в будущем было решено перейти на использование Poetry в качестве менеджера пакетов. Poetry использует собственный решатель зависимостей, который сообщит об ошибке в подобной ситуации. С данным изменением был оформлен Pull Request, который будет влит в основную ветку после окончания семестра.

5. Апробация

> Случайные тесты как коробка шоколадных конфет — никогда не знаешь, какой упадёт.
>
> Барсуков Илья, магистр ИТМО

В весеннем семестре 2024 года данный курс читался первокурсникам магистратуры ИТМО. В течение семестра были выявлены следующие проблемы.

- Отсутствие ручных тестов.
- Необходимость тестирования всех требуемых функций.
- Необходимость предупреждать студентов о том, что решения нужно оптимизировать.

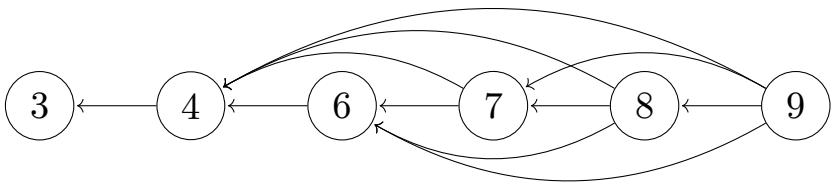
5.1. Отсутствие ручных тестов

При постановке задачи планировалось использовать property-based тесты. Тем не менее данный подход в одиночку оказался недостаточен.

В большинстве задач тесты основываются на сравнении ответов различных алгоритмов (см. рисунок 3). Так алгоритм из задачи №3 тестируется только в тестах задачи №4, когда появляется возможность сравнить результаты. А затем алгоритмом из задачи №4 тестируются задачи №№6–9.

К сожалению, в таком случае есть вероятность, что задачи №№3 и 4 решены одинаково неправильно, из-за чего тесты проходят, и проверя-

Рис. 3: Зависимость задач друг от друга



ющий засчитывает некорректное решение. А при выполнении задания №6 не проходят тесты по непонятной студенту причине.

В такую ловушку попало как минимум два студента: их решения не обрабатывали граничный случай — пустую строку. Из этого был сделан вывод о том, что для каждой задачи стоит иметь небольшой набор ручных тестов, проверяющих в том числе крайние случаи.

5.2. Тестирование всех функций

Для некоторых заданий требуются вспомогательные функции, которые затем могут использоваться в тестах. Тем не менее так получается не всегда, и в таком случае стоит иметь хотя бы тесты, проверяющие самое наличие функции, иначе про необходимость реализации забывают как студенты, так и проверяющие.

Основная алгоритмическая функция из задания №3 тестировалась только в задаче №4 и многие студенты забыли её реализовать, и нескольким студентам задача была зачтена без неё. Эти студенты реализовали её при выполнении задачи №4.

5.3. Оптимизация решений

В мае был выполнен рефакторинг кода автоматических тестов, в следствие чего количество тестов резко увеличилось. Время исполнения тестов наших решений увеличилось немного, а вот тесты у некоторых студентов стали проходить по 4 часа.

В процессе исследования выяснилось, что студенты склонны выполнять задания «в лоб», так, чтобы проходили тесты. А нами при реализации эталонных решений применялись доступные способы оптимизации.

В результате, был сделан вывод о том, что необходимо каким-либо образом мотивировать студентов на оптимизацию собственных решений.

Заключение

В рамках данной учебной практики были достигнуты следующие результаты.

1. Изучены способы автоматического тестирования учебных задач.
2. Создана базовая инфраструктура для автоматического тестирования.
3. Произведена проверка автоматических тестов, путём реализации решений.
4. Исправлен накопленный технический долг.

Для исправления выявленных недостатков в будущем планируется:

1. добавить ручные тесты;
2. добавить тесты на наличие вспомогательных функций;
3. добавить ограничение на скорость исполнения алгоритмов.

Исходный код тестов и инфраструктуры доступен по адресу: https://github.com/FormalLanguageConstrainedPathQuerying/formal-lang-course. Имя пользователя: `WoWaster`.

Исходный код решений находится в приватном репозитории.

Pull Request в проект CFPQ_Data доступен по следующей ссылке: https://github.com/FormalLanguageConstrainedPathQuerying/CFPQ_Data/pull/92

Список литературы

[1] Azimov Rustam, Grigorev Semyon. Context-Free Path Querying by Matrix Multiplication. — P. 1–10.

[2] Context-Free Path Querying by Kronecker Product / Egor Orachev, Ilya Epelbaum, Rustam Azimov, Semyon Grigorev. — P. 49–59.

[3] Draghici Andrei, Haase Christoph, Manea Florin. Sem\"enov Arithmetic, Affine VASS, and String Constraints. — arxiv : cs/2306.14593.

[4] Estimating Probabilistic Context-Free Grammars for Proteins Using Contact Map Constraints / Witold Dyrka, Mateusz Pyzik, François Coste, Hugo Talibart. — Vol. 7. — P. e6559. — pmid : 30918754.

[5] Evolving Stochastic Context–Free Grammars for RNA Secondary Structure Prediction / James WJ Anderson, Paula Tataru, Joe Staines et al. — Vol. 13. — P. 78. — pmid : 22559985.

[6] Abzalov Vadim, Pogozhelskaya Vlada, Kutuev Vladimir, Grigorev Semyon. GLL-based Context-Free Path Querying for Neo4j. — arxiv : cs/2312.11925.

[7] Generating Grammar-Conformant Texts from ANTLR Grammars (or Other Way)? — URL: https://groups.google.com/g/antlr-discussion/c/6aatRq-IH1U?pli=1 (дата обращения: 2024-05-26).

[8] A GraphBLAS Solution to the SIGMOD 2014 Programming Contest Using Multi-Source BFS / Marton Elekes, Attila Nagy, David Sandor et al. // 2020 IEEE High Performance Extreme Computing Conference (HPEC). — IEEE. — P. 1–7. — URL: https://ieeexplore.ieee.org/document/9286186/ (дата обращения: 2024-05-31).

[9] Grune Dick, Jacobs Ceriel J. H. Parsing Techniques. Monographs in Computer Science. — Springer. — ISBN: 978-0-387-20248-8 978-0-387-68954-8. — URL: http://link.springer.com/10.1007/978-0-387-68954-8 (дата обращения: 2024-05-27).

[10] Hagberg Aric A., Schult Daniel A., Swart Pieter J. Exploring Network Structure, Dynamics, and Function Using NetworkX // Proceedings of the 7th Python in Science Conference / Ed. by Gaël Varoquaux, Travis Vaught, Jarrod Millman. — P. 11–15.

[11] Hellings Jelle. Conjunctive Context-Free Path Queries. — URL: https://openproceedings.org/ICDT/2014/paper_34.pdf (дата обращения: 2024-05-31).

[12] Hellings Jelle. Querying for Paths in Graphs Using Context-Free Path Queries. — arxiv : cs/1502.02242.

[13] Nolé Maurizio, Sartiani Carlo. Regular Path Queries on Massive Graphs // Proceedings of the 28th International Conference on Scientific and Statistical Database Management. — SSDBM '16. — Association for Computing Machinery. — P. 1–12. — URL: https://doi.org/10.1145/2949689.2949711 (дата обращения: 2024-05-27).

[14] Shemetova Ekaterina, Azimov Rustam, Orachev Egor et al. One Algorithm to Evaluate Them All: Unified Linear Algebra Based Approach to Evaluate Both Regular and Context-Free Path Queries. — arxiv : cs/2103.14688.

[15] Guha Shibashis, Jecker Ismaël, Lehtinen Karoliina, Zimmermann Martin. Parikh Automata over Infinite Words. — arxiv : cs/2207.07694.

[16] Parr Terence. The Definitive ANTLR 4 Reference. — 2 edition. — Pragmatic Bookshelf. — ISBN: 978-1-934356-99-9.

[17] Romero Julien. Pyformlang: An Educational Library for Formal Language Manipulation // Proceedings of the 52nd ACM Technical Symposium on Computer Science Education. — SIGCSE '21. — Association for Computing Machinery. — P. 576–582. — URL: https://dl.acm.org/doi/10.1145/3408877.3432464 (дата обращения: 2024-05-27).

[18] SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python / Pauli Virtanen, Ralf Gommers, Travis E. Oliphant et al. — Vol. 17, no. 3. — P. 261–272. — URL: https://www.nature.com/articles/s41592-019-0686-2 (дата обращения: 2024-05-27).

[19] Scott Elizabeth, Johnstone Adrian. GLL Parsing. — Vol. 253, no. 7. — P. 177–189. — URL: https://www.sciencedirect.com/science/article/pii/S1571066110001209 (дата обращения: 2024-05-29).

[20] В 2024 году количество бюджетных мест в российских вузах будет увеличено. — URL: https://minobrnauki.gov.ru/press-center/news/novosti-ministerstva/74773/ (дата обращения: 2024-05-27).