Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ

По лабораторной работе №3

Дисциплины «Искусственный интеллект в профессиональной сфере»

Выполнил:

Евдаков Евгений Владимирович

3 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____

(подпись)

Руководитель практики:

Воронкин Р. А., доцент департамента цифровых и робототехнических систем и электроники института перспективной инженерии

_____

(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: исследование поиска в глубину.

Цель: приобретение навыков по работе с поиском в глубину с помощью языка программирования Python версии 3.x

Ход работы:

Задание 1. Создал общедоступный репозиторий на GitHub, в котором использована лицензий MIT и язык программирования Python, также добавил файл .gitignore с необходимыми правилами. Клонировал свой репозиторий на свой компьютер. Организовал свой репозиторий в соответствие с моделью ветвления git-flow, появилась новая ветка develop в которой буду выполнять дальнейшие задачи.

```
C:\Users\evdak>git clone https://github.com/EvgenyEvdakov/Laba_ii_3.git
Cloning into 'Laba_ii_3'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (5/5), done.
```

Рисунок 1. Клонирование репозитория

Задание 2. Создал виртуальное окружение conda и активировал его, также установил необходимые пакеты isort, black, flake8.

```
(base) PS C:\Users\evdak> cd C:\Users\evdak\Laba_ii_3
(base) PS C:\Users\evdak\Laba_ii_3> conda create -n ii_3 python=3.10
Retrieving notices: ...working... done
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\evdak\.conda\envs\ii_3

  added / updated specs:
    - python=3.10


The following NEW packages will be INSTALLED:

  bzip2              pkgs/main/win-64::bzip2-1.0.8-h2bbff1b_6
  ca-certificates    pkgs/main/win-64::ca-certificates-2024.9.24-haa95532_0
  libffi             pkgs/main/win-64::libffi-3.4.4-hd77b12b_1
  openssl            pkgs/main/win-64::openssl-3.0.15-h827c3e9_0
  pip                pkgs/main/win-64::pip-24.2-py310haa95532_0
  python             pkgs/main/win-64::python-3.10.15-h4607a30_1
  setuptools         pkgs/main/win-64::setuptools-75.1.0-py310haa95532_0
  sqlite             pkgs/main/win-64::sqlite-3.45.3-h2bbff1b_0
  tk                 pkgs/main/win-64::tk-8.6.14-h0416ee5_0
  tzdata             pkgs/main/noarch::tzdata-2024b-h04d1e81_0
  vc                 pkgs/main/win-64::vc-14.40-h2eaa2aa_1
  vs2015_runtime     pkgs/main/win-64::vs2015_runtime-14.40.33807-h98bb1dd_
  wheel              pkgs/main/win-64::wheel-0.44.0-py310haa95532_0
  xz                 pkgs/main/win-64::xz-5.4.6-h8cc25b3_1
  zlib               pkgs/main/win-64::zlib-1.2.13-h8cc25b3_1
```

Рисунок 2. Создание виртуального окружения

Задание 3. Создал проект PyCharm в папке репозитория. Приступил к работе с примером. Добавил новый файл primer.py. Рассмотрим реализацию алгоритма поиска в глубину на практике, в программном коде:

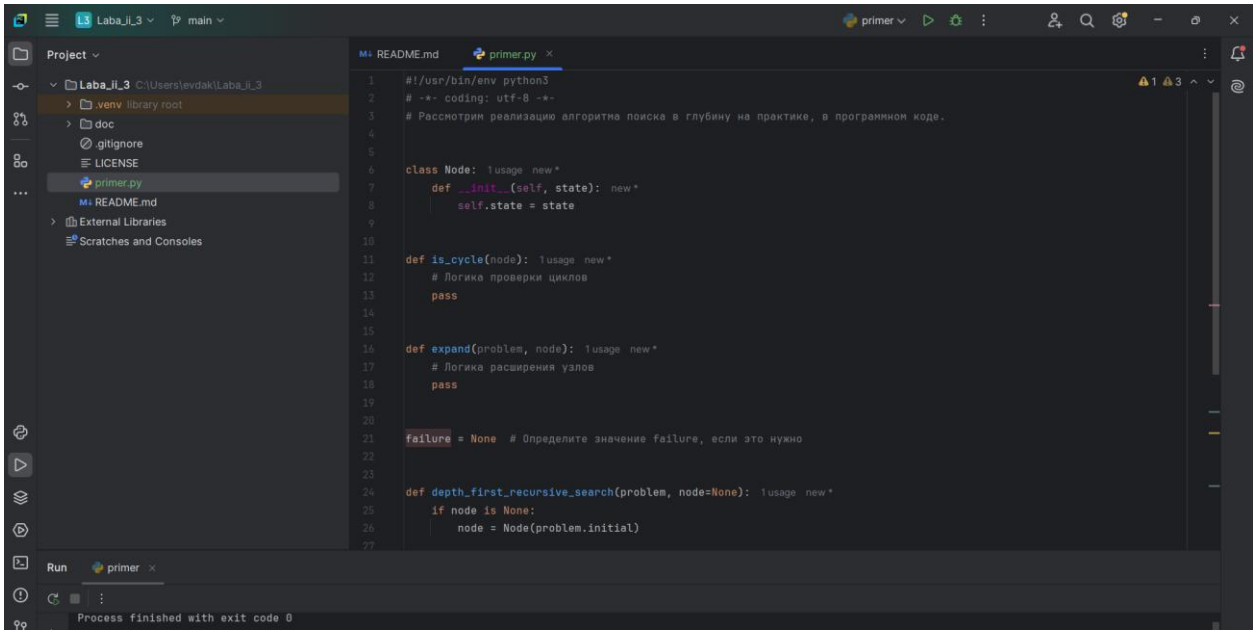

Рисунок 3. Выполнение примера

Задание 4. Необходимо для задачи "Поиск самого длинного пути в матрице" подготовить собственную матрицу, для которой с помощью разработанной в предыдущем пункте программы, подсчитать самый длинный путь. Разработаем матрицу:

```
new_matrix = [
    ["A", "B", "C", "D", "E", "F", "T"],
    ["Z", "Y", "X", "W", "V", "G", "H"],
    ["I", "J", "K", "L", "M", "T", "S"],
    ["R", "Q", "P", "O", "N", "U", "T"],
    ["S", "T", "U", "V", "W", "X", "Y"],
    ["Z", "A", "B", "C", "D", "E", "F"],
    ["G", "H", "I", "J", "K", "L", "M"]
]
```

Рисунок 4. Матрица для первого задания

Дана матрица символов размером M×N. Необходимо найти длину самого длинного пути в матрице, начиная с заданного символа. Каждый следующий символ в пути должен алфавитно следовать за предыдущим без пропусков. Разработать функцию поиска самого длинного пути в матрице

символов, начиная с заданного символа. Символы в пути должны следовать в алфавитном порядке и быть последовательными. Поиск возможен во всех восьми направлениях. Напишем программу:

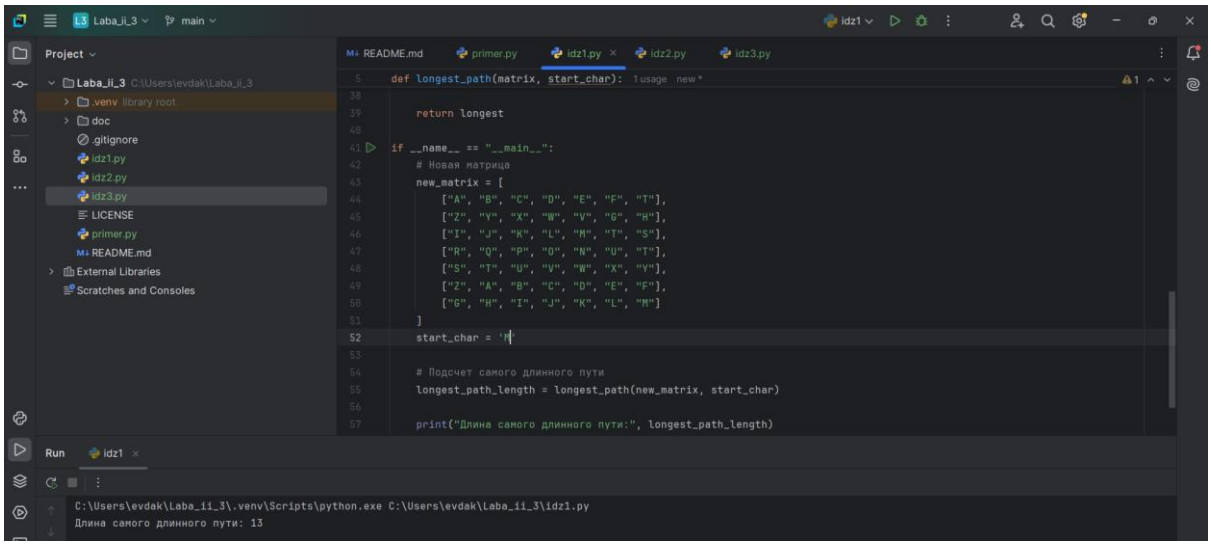

Рисунок 5. Выполнение программы для поиска самого длинного пути в матрице

Задание 5. Необходимо для задачи "Генерирование списка возможных слов из матрицы символов" подготовить собственную матрицу для генерирования списка возможных слов с помощью разработанной программы.

```
board = [
    ['P', 'A', 'C'],
    ['T', 'E', 'Л'],
    ['И', 'O', 'H']
]
```

Рисунок 6. Матрица для второго задания

Наша задача — найти и вывести список всех возможных слов, которые могут быть сформированы из последовательности соседних символов в этой матрице. При этом слово может формироваться во всех восьми возможных направлениях (север, юг, восток, запад, северо-восток, северо-запад, юго-восток, юго-запад), и каждая клетка может быть использована в слове только один раз.

Напишем программу:

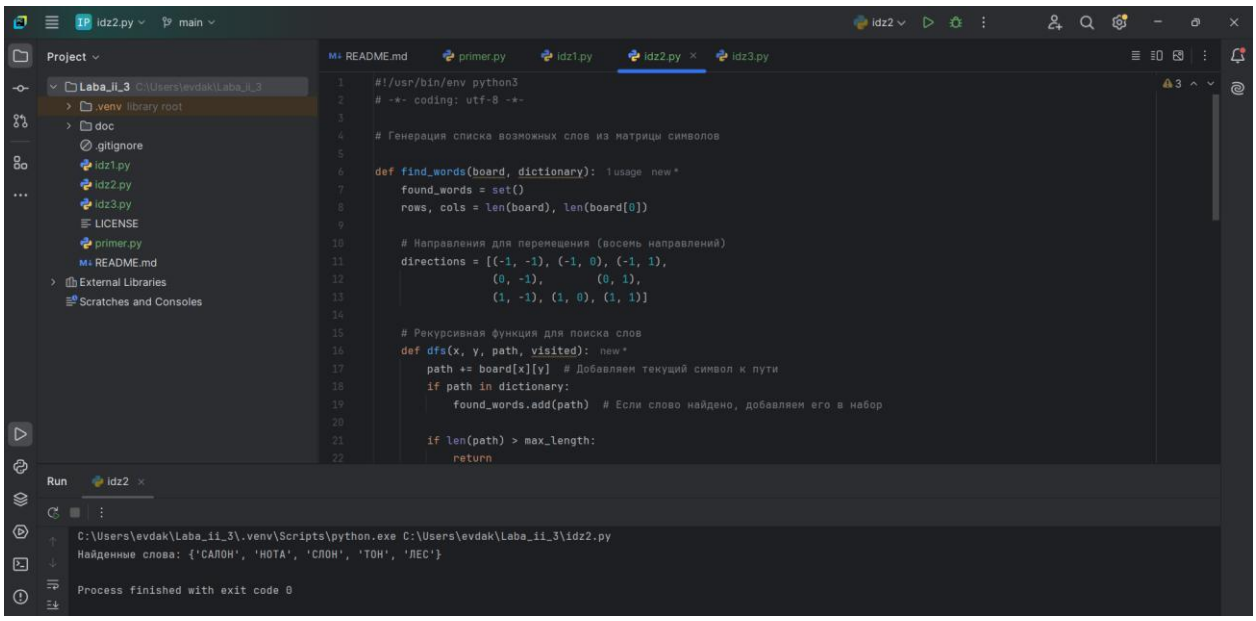

Рисунок 7. Выполнение программы для генерации списка слов из матрицы символов

Задание 6. Необходимо для построенного графа лабораторной работы 1 написать программу на языке программирования Python, которая с помощью алгоритма поиска в глубину находит минимальное расстояние между начальным и конечным пунктами. И так сравним найденное решение с решением, полученным вручную.

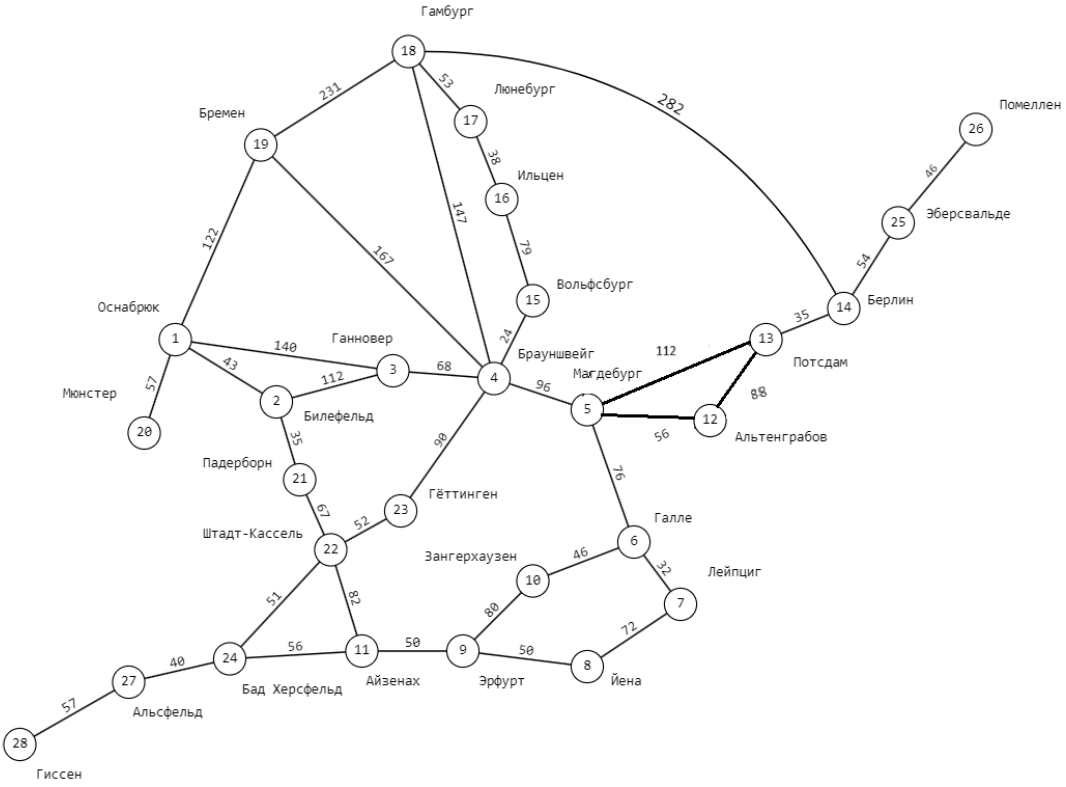

Рисунок 8. Построение графа

Найдем минимальное расстояние между городами Гамбург и Альтенграбов. Если считать вручную, то минимальное расстояние составляет 299 км. Далее составим программу и проверим:

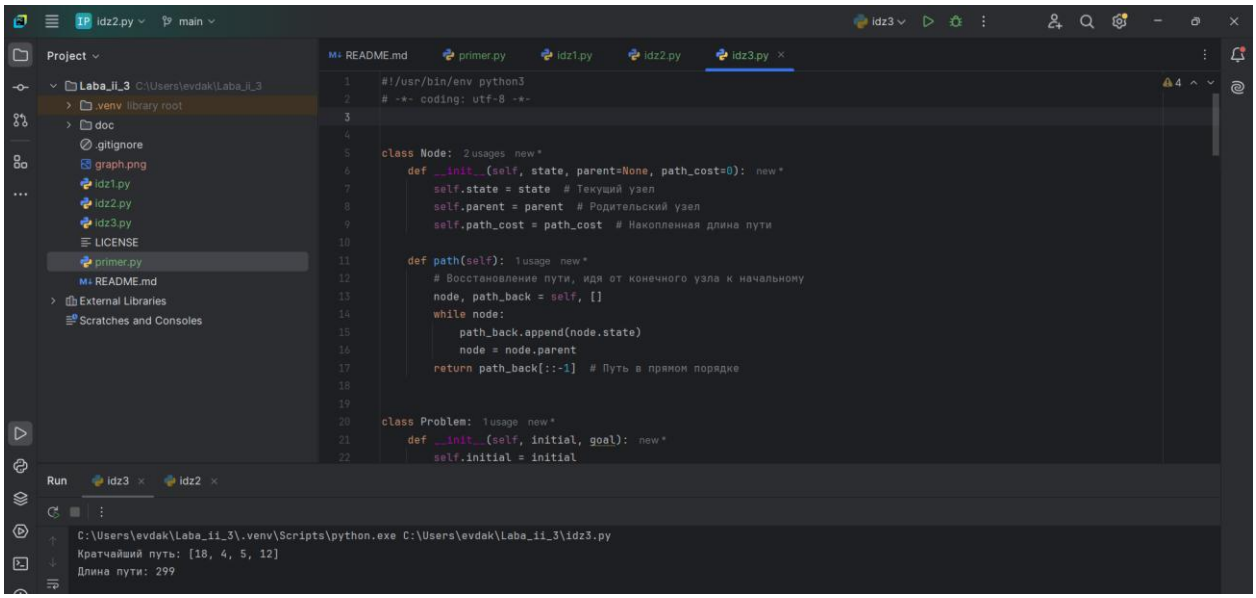

Рисунок 9. Выполнение программы

Результат программы вывел так же 299 км.

Задание 7.

После выполнения работы на ветке develop, слил ее с веткой main и отправил изменения на удаленный сервер. Создал файл environment.yml и деактивировал виртуальное окружение.

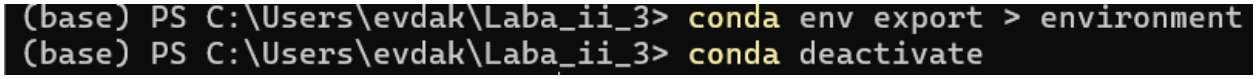

Рисунок 10. Деактивация BO

Ссылка: https://github.com/EvgenyEvdakov/Laba_ii_3

Ответы на контрольные вопросы:

1. **В чем ключевое отличие поиска в глубину от поиска в ширину?**

Ключевое отличие состоит в стратегии обхода. Поиск в глубину (DFS) исследует одну ветвь дерева/графа до самого глубокого уровня, прежде чем перейти к следующей ветви. Поиск в ширину (BFS) исследует все узлы на одном уровне глубины перед переходом к следующему уровню.

2. **Какие четыре критерия качества поиска обсуждаются в тексте для оценки алгоритмов?**

- Полнота — гарантирует ли алгоритм нахождение решения, если оно существует.
- Оптимальность — находит ли алгоритм лучшее (минимальное по стоимости) решение.
- Временная сложность — сколько времени требуется для нахождения решения.
- Пространственная сложность — сколько памяти использует алгоритм.

3. Что происходит при расширении узла в поиске в глубину?

При расширении узла генерируются его дочерние узлы. Это включает создание новых узлов на основе соседей текущего узла, которые затем добавляются в стек или передаются в рекурсию.

4. Почему поиск в глубину использует очередь типа "последним пришел — первым ушел" (LIFO)?

Очередь LIFO (стек) обеспечивает приоритетное исследование недавно добавленных узлов, что позволяет алгоритму "углубляться" в ветви графа или дерева.

5. Как поиск в глубину справляется с удалением узлов из памяти, и почему это преимущество перед поискм в ширину?

Поиск в глубину удаляет узлы из памяти, как только они обработаны и нет необходимости возвращаться к ним. Это снижает объем памяти по сравнению с поиском в ширину, который должен хранить все узлы на текущем уровне.

6. Какие узлы остаются в памяти после того, как достигнута максимальная глубина дерева?

В памяти остаются узлы текущего пути от корня до текущего узла и узлы, которые еще не были исследованы.

7. В каких случаях поиск в глубину может "застрять" и не найти решение?

- Если граф или дерево бесконечно глубокие.

- Если существует цикл, и алгоритм не имеет проверки на циклы.

8. Как временная сложность поиска в глубину зависит от максимальной глубины дерева?

Временная сложность DFS составляет $O(b^m)$, где b — фактор ветвления, а m — максимальная глубина дерева. Она растет экспоненциально с глубиной дерева.

9. Почему поиск в глубину не гарантирует нахождение оптимального решения?

DFS не рассматривает все пути одновременно, поэтому может найти не самый короткий путь, если решение обнаружено до исследования более выгодного варианта.

10. В каких ситуациях предпочтительно использовать поиск в глубину, несмотря на его недостатки?

- Когда пространство поиска ограничено и важно минимизировать потребление памяти.
- Если известна приблизительная глубина решения.
- Когда нужно найти любое решение быстро, а не обязательно оптимальное.

11. Что делает функция depth_first_recursive_search, и какие параметры она принимает?

Функция выполняет рекурсивный поиск в глубину для нахождения решения. Она принимает:

- problem — задачу, содержащую начальный узел и цель.
- graph — граф для обхода.
- node (опционально) — текущий узел.

Дополнительно может передаваться текущая минимальная длина пути и путь.

12. Какую задачу решает проверка if node is None?

Она задает начальный узел, если он не был передан в качестве параметра.

13. В каком случае функция возвращает узел как решение задачи?

Когда состояние узла совпадает с целевым состоянием задачи (problem.is_goal(node.state)).

14. Почему важна проверка на циклы в алгоритме рекурсивного поиска в глубину?

Проверка на циклы предотвращает бесконечный возврат к ранее посещенным узлам, особенно в графах с циклическими структурами.

15. Что возвращает функция при обнаружении цикла?

Она возвращает None (или failure), указывая, что цикл был обнаружен и продолжение поиска по этому пути невозможно.

16. Как функция обрабатывает дочерние узлы текущего узла?

Функция генерирует дочерние узлы через expand и рекурсивно вызывает саму себя для каждого из них.

17. Какой механизм используется для обхода дерева поиска в этой реализации?

Используется рекурсия для перехода между узлами, а стек вызовов автоматически сохраняет текущий путь.

18. Что произойдет, если не будет найдено решение в ходе рекурсии?

Функция вернет failure, что указывает на отсутствие пути к цели.

19. Почему функция рекурсивно вызывает саму себя внутри цикла?

Это позволяет исследовать все ветви графа/дерева, начиная с текущего узла.

20. Как функция expand(problem, node) взаимодействует с текущим узлом?

Она генерирует список дочерних узлов текущего узла на основе графа и его соседей.

21. Какова роль функции is_cycle(node) в этом алгоритме?

Она проверяет, встречался ли текущий узел ранее в пути, предотвращая зацикливание.

22. Почему проверка if result в рекурсивном вызове важна для корректной работы алгоритма?

Эта проверка определяет, было ли найдено решение по данному пути, и завершает дальнейший поиск, если оно найдено.

23. В каких ситуациях алгоритм может вернуть failure?

- Если узел не может быть расширен (нет дочерних узлов).
- Если все пути исследованы, но цель не достигнута.

24. Как рекурсивная реализация отличается от итеративного поиска в глубину?

В рекурсивной реализации используется стек вызовов, управляемый автоматически, в то время как в итеративной используется явный стек для хранения состояния узлов.

25. Какие потенциальные проблемы могут возникнуть при использовании этого алгоритма для поиска в бесконечных деревьях?

- Бесконечная рекурсия при отсутствии проверки на глубину или циклы.
- Переполнение стека вызовов, что приведет к ошибке сегментации (stack overflow).

Вывод: приобрел навыки по работе с поиском в глубину с помощью языка программирования Python версии 3.x