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 №2

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 ширину.

Цель: приобретение навыков по работе с поиском в ширину 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_2.git
Cloning into 'Laba_ii_2'...
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_2
(base) PS C:\Users\evdak\Laba_ii_2> conda create -n ii_2 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_2

  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
```

Рисунок 2. Создание виртуального окружения

Задание 3. Создал проект PyCharm в папке репозитория. Приступил к работе с примером. Добавил новый файл primer.py.

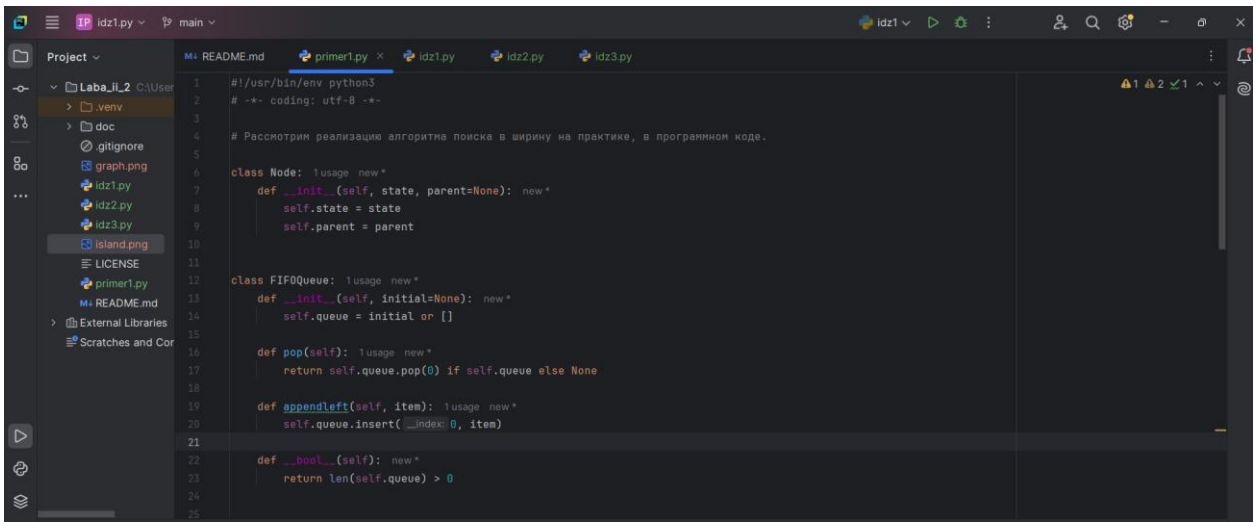

Рисунок 3. Выполнение примера

Задание 4. Необходимо для задачи "Расширенный подсчет количества островов в бинарной матрице" подготовить собственную матрицу, для которой с помощью разработанной в предыдущем пункте программы, подсчитаем количество островов. Разработаем матрицу:

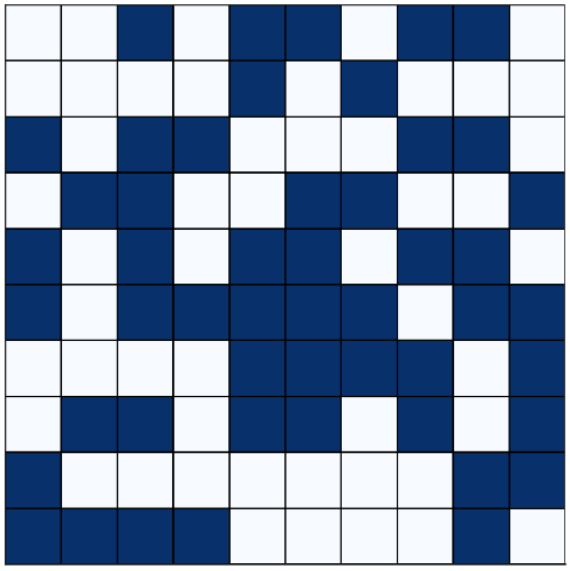

Рисунок 4. Используемая среда для создания матрицы

Из данной среды образуем бинарную матрицу, где белым цветом будет представлена вода, а синим представлена земля. Связанные единицы формируют остров. Необходимо подсчитать общее количество островов в данной матрице. Острова могут соединяться как по вертикали и горизонтали, так и по диагонали. Напишем программу:

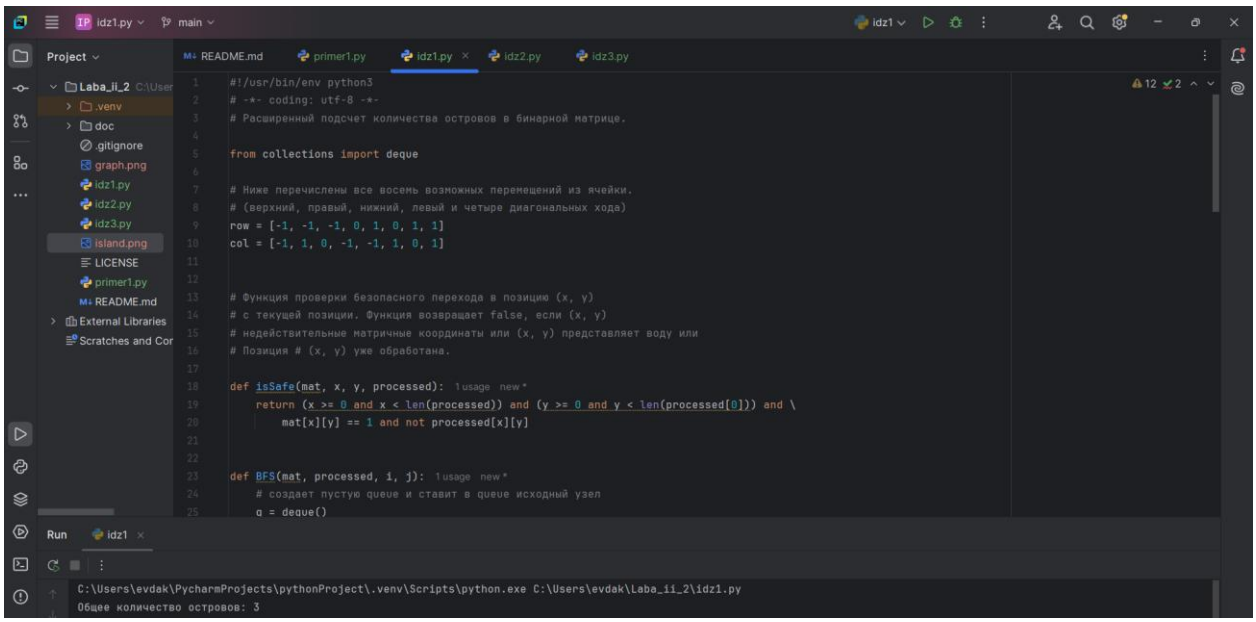

Рисунок 5. Результат программы для задачи "Расширенный подсчет количества островов в бинарной матрице"

Вывод программы показал результат 3, что соответствует выбранной нами матрице с островами.

Задание 5. Необходимо для задачи "Поиск кратчайшего пути в лабиринте" подготовить собственную схему лабиринта, а также определить начальную и конечную позиции в лабиринте. Для данных найти минимальный путь в лабиринте от начальной к конечной позиции. Заполним матрицу:

```
maze = [
    [1, 0, 1, 1, 1, 0, 1, 1, 1, 1],
    [1, 0, 1, 0, 1, 0, 0, 0, 0, 1],
    [1, 1, 1, 0, 1, 1, 1, 1, 0, 1],
    [0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
    [1, 1, 1, 1, 1, 1, 0, 1, 1, 1],
    [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
    [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 0, 1, 1],
    [1, 0, 0, 0, 0, 0, 1, 1, 1, 1],
]
```

Рисунок 6. Матрица в виде лабиринта

Напишем программу для поиска кратчайшего пути через лабиринт, используя алгоритм поиска в ширину (BFS). Лабиринт представлен в виде бинарной матрицы, где 1 обозначает проход, а 0 — стену.

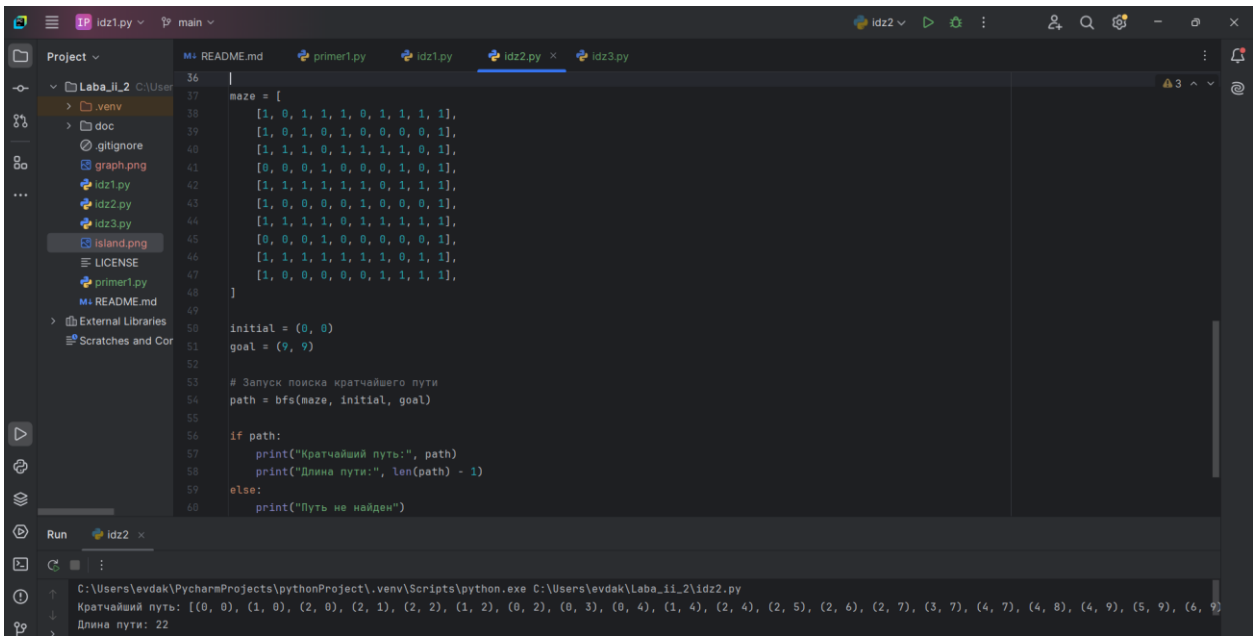

Рисунок 7. Выполнение программы для поиска кратчайшего пути из лабиринта

Задание 6. Необходимо для построенного графа лабораторной работы 1 написать программу на языке программирования Python, которая с помощью алгоритма поиска в ширину находит минимальное расстояние между начальным и конечным пунктами. Сравним найденное решение с решением, полученным вручную. Найдем минимальное расстояние между городами Гамбург и Альтенграбов.

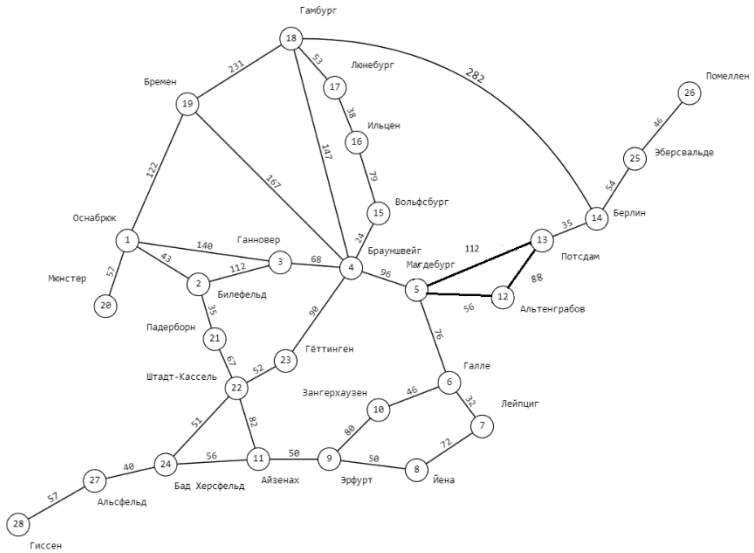

Рисунок 8. Граф из лабораторной работы 1

Если считать вручную, то минимальное расстояние составляет 299 км. Далее составим программу и проверим:

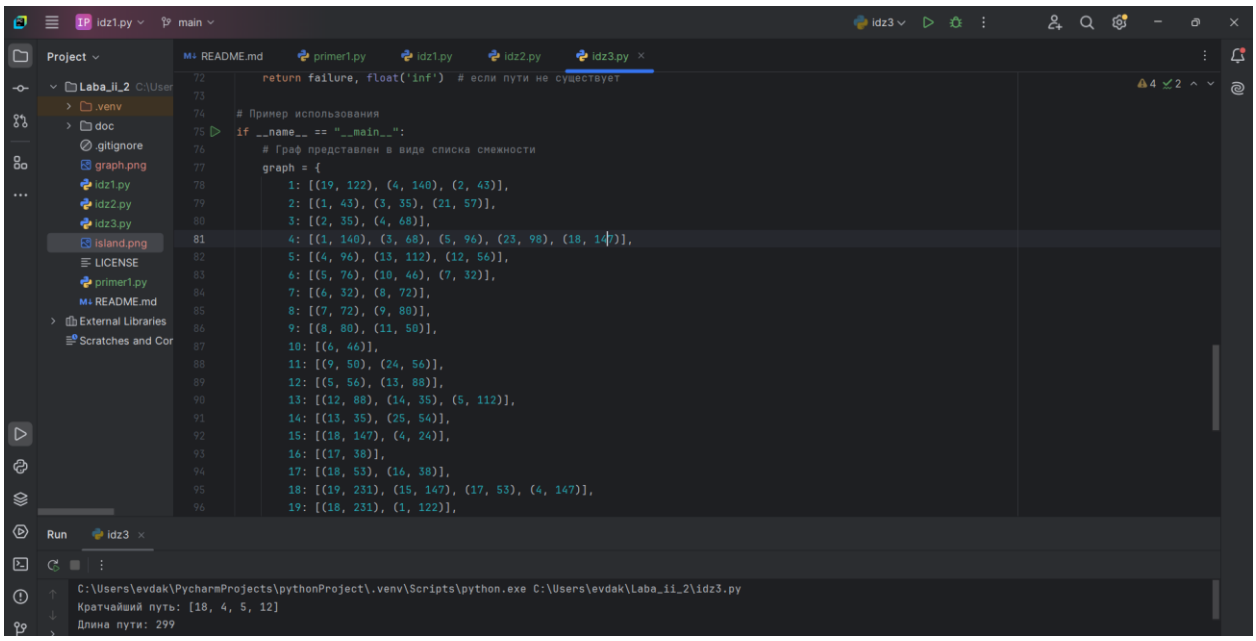

Рисунок 9. Выполнение программы

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 envirement.yml и деактивировал виртуальное окружение.

```
(ii_1) PS C:\Users\evdak\Laba_ii_1> conda env export > environment
(ii_1) PS C:\Users\evdak\Laba_ii_1> conda deactivate
```

Рисунок 10. Деактивация ВО

Ссылка: https://github.com/EvgenyEvdakov/Laba_ii_2

Ответы на контрольные вопросы:

1. **Какой тип очереди используется в стратегии поиска в ширину?**

В поиске в ширину используется очередь FIFO (First In, First Out), где узлы извлекаются в том порядке, в котором были добавлены.

2. **Почему новые узлы в стратегии поиска в ширину добавляются в конец очереди?**

Это позволяет гарантировать, что узлы будут расширяться в порядке их глубины, т.е., сначала обрабатываются более близкие к корню узлы, затем более удаленные. Это является основной стратегией поиска в ширину.

3. Что происходит с узлами, которые дольше всего находятся в очереди в стратегии поиска в ширину?

Узлы, которые дольше находятся в очереди, будут извлекаться и расширяться первыми, так как очередь FIFO гарантирует, что первым выходит узел, который был добавлен раньше всех.

4. Какой узел будет расширен следующим после корневого узла, если используются правила поиска в ширину?

Следующими будут расширены узлы, которые непосредственно связаны с корневым узлом, то есть узлы на глубине 1.

5. Почему важно расширять узлы с наименьшей глубиной в поиске в ширину?

Это гарантирует, что первое найденное решение является оптимальным (самым коротким путём) в терминах количества шагов от корня до цели.

6. Как временная сложность алгоритма поиска в ширину зависит от коэффициента разветвления и глубины?

Временная сложность поиска в ширину зависит от двух факторов: коэффициента разветвления (то есть количества потомков у каждого узла) и глубины целевого узла (то есть минимального числа шагов до цели). Поиск в ширину проходит все узлы уровня за уровнем, начиная с корня, поэтому на каждом новом уровне количество узлов для обработки резко возрастает. Чем больше потомков у каждого узла и чем глубже находится целевое состояние, тем больше узлов нужно обработать. Это приводит к экспоненциальному росту времени выполнения при увеличении этих двух параметров.

7. Каков основной фактор, определяющий пространственную сложность алгоритма поиска в ширину?

Основной фактор, влияющий на объем памяти, который требует поиск в ширину, — это количество узлов, которые нужно сохранить на самом нижнем уровне поиска. Так как алгоритм должен хранить в памяти все узлы на каждом уровне, пока они не будут обработаны, наибольшее количество узлов накапливается на последнем уровне. Чем больше у узлов потомков и чем

глубже находится целевое состояние, тем больше узлов нужно хранить одновременно, и это сильно увеличивает потребность в памяти.

8. В каких случаях поиск в ширину считается полным?

Поиск в ширину считается полным, если пространство состояний конечно или если решение существует на конечной глубине, т.е. если есть гарантии достижения цели.

9. Объясните, почему поиск в ширину может быть неэффективен с точки зрения памяти.

Поскольку поиск в ширину хранит в памяти все узлы на каждом уровне, он требует много памяти, особенно при высоком коэффициенте разветвления и большой глубине .

10. В чем заключается оптимальность поиска в ширину?

Поиск в ширину является оптимальным по количеству шагов, если все шаги имеют одинаковую длину, так как он первым находит кратчайший путь от начального состояния к целевому.

11. Какую задачу решает функция breadth_first_search?

Breadth_first_search решает задачу поиска пути от начального состояния к целевому состоянию, используя алгоритм поиска в ширину.

12. Что представляет собой объект problem, который передается в функцию?

Problem представляет собой объект задачи, который содержит начальное состояние, целевое состояние, а также методы для определения допустимых действий и проверки достижения цели.

13. Для чего используется узел Node(problem.initial) в начале функции?

Node(problem.initial) создаёт корневой узел дерева поиска, представляющий начальное состояние задачи, с которого начинается процесс поиска.

14. Что произойдет, если начальное состояние задачи уже является целевым?

Если начальное состояние уже является целевым, функция breadth_first_search немедленно вернет этот узел, завершая поиск.

15. Какую структуру данных использует frontier и почему выбрана именно очередь FIFO?

Frontier использует очередь FIFO для обеспечения расширения узлов в порядке их глубины, что соответствует стратегии поиска в ширину.

16. Какую роль выполняет множество reached?

Множество reached хранит состояния, которые уже были достигнуты, чтобы избежать повторного расширения одного и того же состояния и предотвратить зацикливание.

17. Почему важно проверять, находится ли состояние в множестве reached?

Это предотвращает повторное расширение одного и того же состояния, экономя время и память.

18. Какую функцию выполняет цикл while frontier?

Цикл while frontier продолжает процесс поиска, пока остаются узлы для расширения. Он завершится, когда либо будет найдено решение, либо будут исчерпаны все узлы.

19. Что происходит с узлом, который извлекается из очереди в строке node = rontier.pop()?

Узел извлекается из очереди для дальнейшего расширения, и его дочерние узлы (возможные новые состояния) будут добавлены в очередь.

20. Какова цель функции expand(problem, node)?

Функция expand генерирует дочерние узлы для данного узла, используя допустимые действия и правила перехода в задаче problem.

21. Как определяется, что состояние узла является целевым?

Целевое состояние определяется с помощью метода is_goal объекта problem, который проверяет, соответствует ли текущее состояние целевому.

22. Что происходит, если состояние узла не является целевым, но также не было ранее достигнуто?

Если состояние узла не является целевым и не было достигнуто ранее, оно добавляется в множество reached и очередь frontier для дальнейшего расширения.

23. Почему дочерний узел добавляется в начало очереди с помощью appendleft(child)?

В алгоритме поиска в ширину дочерний узел добавляется в конец очереди, а не в начало, чтобы соблюсти принцип FIFO (очередь с извлечением элементов в порядке их поступления). Это гарантирует, что узлы будут обрабатываться по мере их добавления в очередь, начиная с узлов, расположенных ближе к корневому, и заканчивая узлами на более глубоких уровнях. Использование метода appendleft(child) применимо, скорее, для алгоритма поиска в глубину, который следует стратегии LIFO (стек), где узлы обрабатываются в порядке последнего добавления.

24. Что возвращает функция breadth_first_search , если решение не найдено?

Если решение не найдено, функция возвращает специальное значение failure, показывающее, что достижение цели невозможно.

25. Каково значение узла failure и когда он возвращается?

Узел failure обычно имеет состояние None или «неудача» и длина пути бесконечность. Он возвращается, если поиск завершился, но не было найдено решения.

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