

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:  
Говоров Егор Юрьевич  
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

Ссылка на гитхаб: [https://github.com/Artorias1469/Artificial\\_Intelligence\\_1\\_aboratory\\_work\\_2.git](https://github.com/Artorias1469/Artificial_Intelligence_1_aboratory_work_2.git)

Ход работы:

1. Выполнение примера из лабораторной работы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

# Рассмотрим реализацию алгоритма поиска в ширину на практике, в программном коде.

class Node:
    """usage new"""
    def __init__(self, state, parent=None):
        """new"""
        self.state = state
        self.parent = parent

class FIFOQueue:
    """usage new"""
    def __init__(self, initial=None):
        """new"""
        self.queue = initial or []

    def pop(self):
        """usage new"""
        return self.queue.pop(0) if self.queue else None

    def appendleft(self, item):
        """usage new"""
        self.queue.insert(0, item)

    def __bool__(self):
        """new"""
        return len(self.queue) > 0

def expand(problem, node):
    """usage new"""
    # Предполагается, что функция expand генерирует дочерние узлы для узла.
    # Это нужно реализовать в зависимости от вашего конкретного случая.
    pass

class Problem:
    """new"""
    def __init__(self, initial):
        """new"""
```

Рис 1. Выполнение примера

2. Задача расширенного подсчета количества островов в бинарной матрице

Условие: Вам дана бинарная матрица, где 0 представляет воду, а 1 представляет землю. Связанные единицы формируют остров. Необходимо подсчитать общее количество островов в данной матрице. Острова могут соединяться как по вертикали и горизонтали, так и по диагонали.

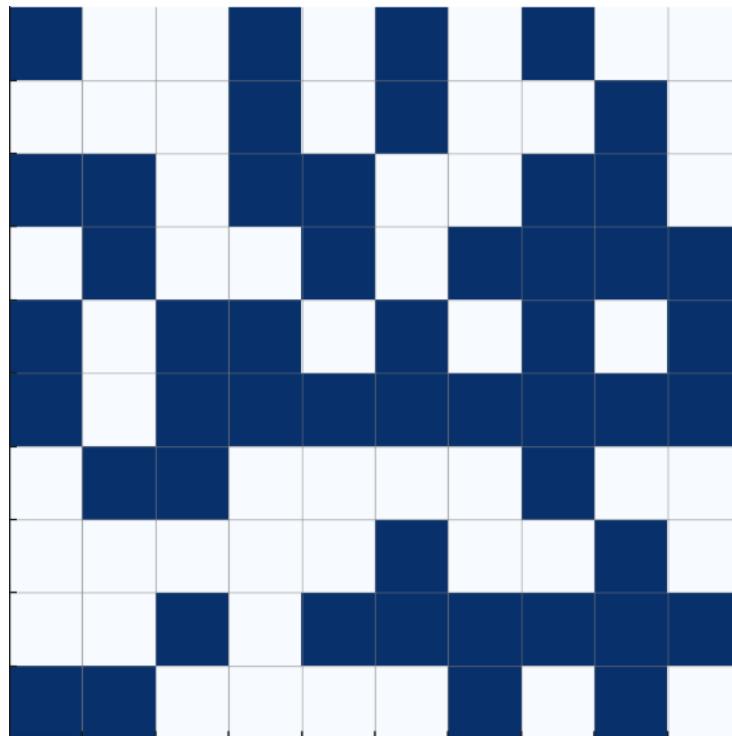


Рис 2. Матрица 10 на 10

```
Prog
  Ind_1.py
  Prim.py
  .gitignore
  LICENSE
  README.md
  External Libraries
  Scratches and Consoles

69 if __name__ == '__main__':
70     mat = [
71         [1, 0, 0, 1, 0, 1, 0, 1, 0, 0],
72         [0, 0, 0, 1, 0, 1, 0, 0, 1, 0],
73         [1, 1, 0, 1, 1, 0, 0, 1, 1, 0],
74         [0, 1, 0, 0, 1, 0, 1, 1, 1, 1],
75         [1, 0, 1, 1, 0, 1, 0, 1, 0, 1],
76         [1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
77         [0, 1, 1, 0, 0, 0, 0, 1, 0, 0],
78         [0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
79         [0, 0, 1, 0, 1, 1, 1, 1, 1, 1],
80         [1, 1, 0, 0, 0, 0, 1, 0, 1, 0]
```

Ind\_1 x

D:\Users\Admin\anaconda3\envs\Artificial\_Intelligence\_laboratory\_work\_2\python.exe "D:\Users\Admin\...

Общее количество островов: 3

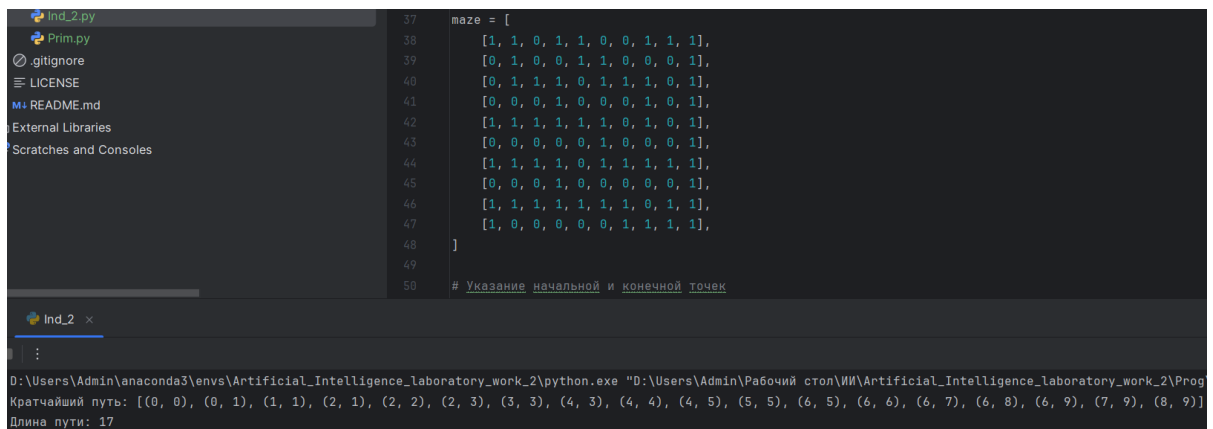
Рис 3. Посчитанное количество островов по данной матрице

### 3. Задача поиска кратчайшего пути в лабиринте

Вам предоставлен код 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. Необходимо модифицировать и дополнить код, чтобы реализовать полный функционал поиска пути.

```
[1, 1, 0, 1, 1, 0, 0, 1, 1, 1],  
[0, 1, 0, 0, 1, 1, 0, 0, 0, 1],  
[0, 1, 1, 1, 0, 1, 1, 1, 0, 1],  
[0, 0, 0, 1, 0, 0, 0, 1, 0, 1],  
[1, 1, 1, 1, 1, 1, 0, 1, 0, 1],  
[0, 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],
```

Рис 4. Матрица поиска



```
37 maze = [  
38     [1, 1, 0, 1, 1, 0, 0, 1, 1, 1],  
39     [0, 1, 0, 0, 1, 1, 0, 0, 0, 1],  
40     [0, 1, 1, 1, 0, 1, 1, 1, 0, 1],  
41     [0, 0, 0, 1, 0, 0, 0, 1, 0, 1],  
42     [1, 1, 1, 1, 1, 1, 0, 1, 0, 1],  
43     [0, 0, 0, 0, 0, 1, 0, 0, 0, 1],  
44     [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],  
45     [0, 0, 0, 1, 0, 0, 0, 0, 0, 1],  
46     [1, 1, 1, 1, 1, 1, 1, 0, 1, 1],  
47     [1, 0, 0, 0, 0, 0, 1, 1, 1, 1],  
48 ]  
49  
50 # Указание начальной и конечной точек
```

Кратчайший путь: [(0, 0), (0, 1), (1, 1), (2, 1), (2, 2), (2, 3), (3, 3), (4, 3), (4, 4), (4, 5), (5, 5), (6, 5), (6, 6), (6, 7), (6, 8), (6, 9), (7, 9), (8, 9)]  
Длина пути: 17

Рис 5. Найденный кратчайший путь из точки [0;0] в точку [8;9]

### 4. Задача нахождения минимального пути между городами с помощью поиска в ширину

Необходимо для графа из ЛР 1 написать программу, которая с помощью алгоритма поиска в ширину находит минимальное расстояние между начальным и конечным пунктами. Сравнить полученное решение с тем, что найдено вручную. Ручной поиск из города Париж в Ганновер равен 798.



Рис 8. Граф из ЛР 1

```
D:\Users\Admin\anaconda3\envs\Artificial_In
Кратчайший путь: [1, 2, 7, 11]
Длина пути: 798
```

Рис 9. Результат программы, который дает нам аналогичный результат

### 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 заикливание.

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 = frontier.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