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»

Выполнила:
Михеева Елена Александровна
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Проверил:
Воронкин Р.А.-доцент департамента цифровых, роботехнических 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

Порядок выполнения работы:

Репозиторий: https://github.com/helendddd/AI_2.git

Расширенный подсчет количества островов в бинарной матрице.

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.

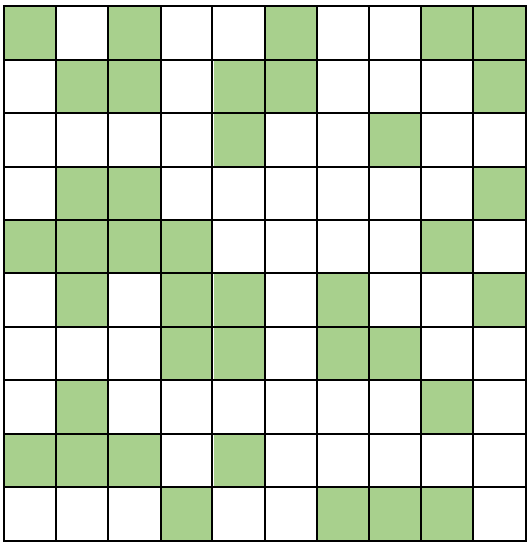
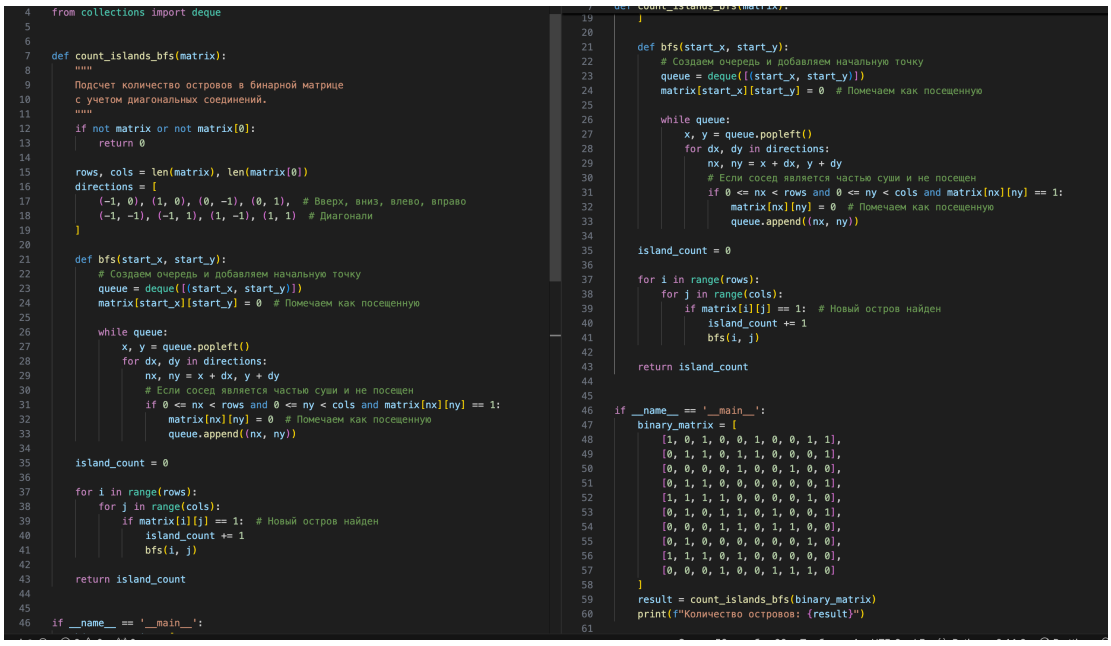


Рисунок 1. Наглядное представление островов в матрице

Или в бинарном представлении:

1	0	1	0	0	1	0	0	1	1
0	1	1	0	1	1	0	0	0	1
0	0	0	0	1	0	0	1	0	0
0	1	1	0	0	0	0	0	0	1
1	1	1	1	0	0	0	0	1	0
0	1	0	1	1	0	1	0	0	1
0	0	0	1	1	0	1	1	0	0
0	1	0	0	0	0	0	0	1	0
1	1	1	0	1	0	0	0	0	0
0	0	0	1	0	0	1	1	1	0



```python
from collections import deque


def count_islands_bfs(matrix):
    """
    Подсчет количество островов в бинарной матрице
    с учетом диагональных соединений.
    """
    if not matrix or not matrix[0]:
        return 0

    rows, cols = len(matrix), len(matrix[0])
    directions = [
        (-1, 0), (1, 0), (0, -1), (0, 1),  # Вверх, вниз, влево, вправо
        (-1, -1), (-1, 1), (1, -1), (1, 1)  # Диагонали
    ]

    def bfs(start_x, start_y):
        # Создаем очередь и добавляем начальную точку
        queue = deque([(start_x, start_y)])
        matrix[start_x][start_y] = 0  # Помечаем как посещенную

        while queue:
            x, y = queue.popleft()
            for dx, dy in directions:
                nx, ny = x + dx, y + dy
                # Если сосед является частью суши и не посещен
                if 0 <= nx < rows and 0 <= ny < cols and matrix[nx][ny] == 1:
                    matrix[nx][ny] = 0  # Помечаем как посещенную
                    queue.append((nx, ny))

    island_count = 0

    for i in range(rows):
        for j in range(cols):
            if matrix[i][j] == 1:  # Новый остров найден
                island_count += 1
                bfs(i, j)

    return island_count


if __name__ == '__main__':
    binary_matrix = [
        [1, 0, 1, 0, 0, 1, 0, 0, 1, 1],
        [0, 1, 1, 0, 1, 1, 0, 0, 0, 1],
        [0, 0, 0, 0, 1, 0, 0, 1, 0, 0],
        [0, 1, 1, 0, 0, 0, 0, 0, 0, 1],
        [1, 1, 1, 1, 0, 0, 0, 0, 1, 0],
        [0, 1, 0, 1, 1, 0, 1, 0, 0, 1],
        [0, 0, 0, 1, 1, 0, 1, 1, 0, 0],
        [0, 1, 0, 0, 0, 0, 0, 0, 1, 0],
        [1, 1, 1, 0, 1, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0, 0, 1, 1, 1, 0]
    ]
    result = count_islands_bfs(binary_matrix)
    print(f"Количество островов: {result}")
```

Рисунок 2. Код программы

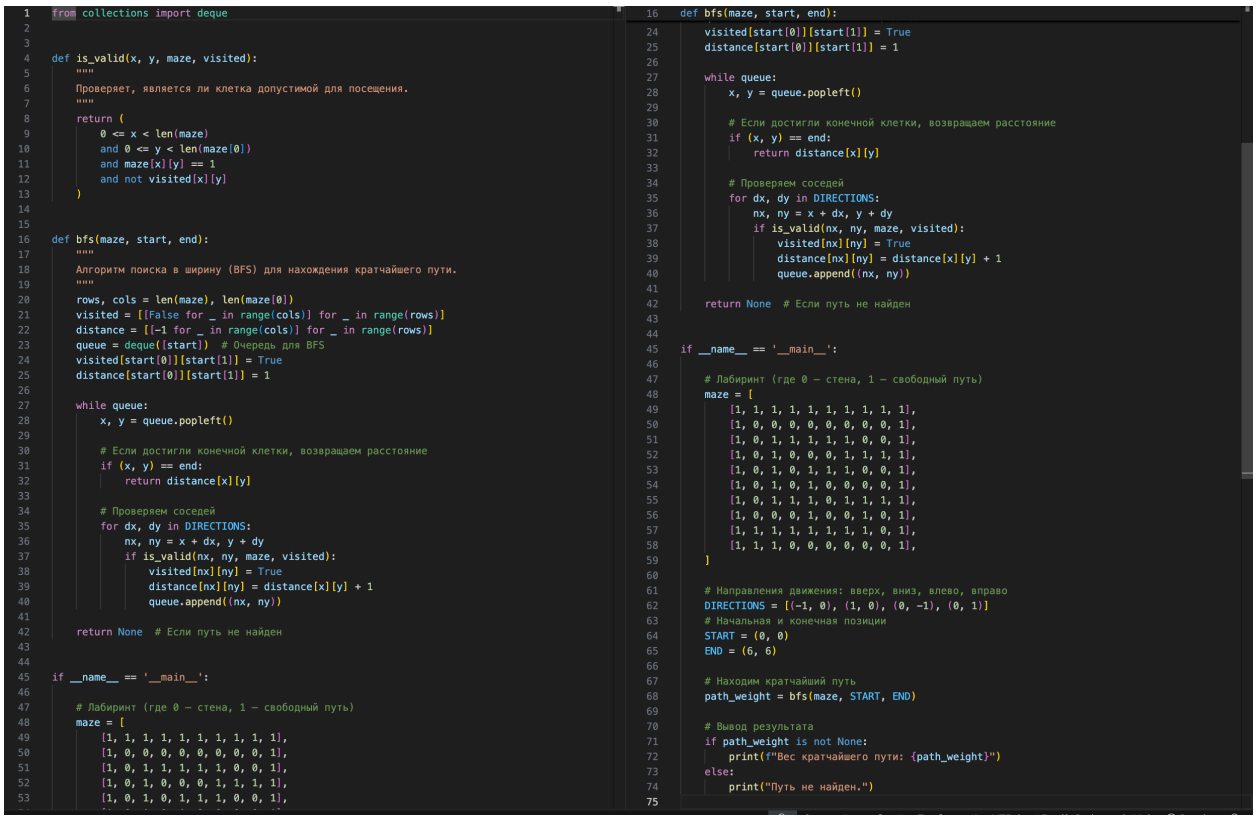
```
(base) elenamiheeva@MacBook-Pro-Elena AI_2 % python3 program/count_island.py
Количество островов: 9
(base) elenamiheeva@MacBook-Pro-Elena AI_2 %
```

Рисунок 3. Результат работы программы

Поиск кратчайшего пути в лабиринте.

Необходимо подготовить схему лабиринта, а также определить начальную и конечную позиции в лабиринте. Лабиринт представлен в виде бинарной матрицы, где 1 обозначает проход, а 0 — стену. На вход программа принимает лабиринт, координаты начальной и конечной точки. В результате работы программа выводит длину полученного пути. В качестве начальной позиции выбрана точка с координатами (1, 1), а для конечной – (7, 7).

1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
1	0	0	0	0	0	0	0	0	**1**
1	0	**1**	**1**	**1**	**1**	**1**	0	0	**1**
1	0	**1**	0	0	0	**1**	**1**	**1**	**1**
1	0	**1**	0	**1**	**1**	**1**	0	0	**1**
1	0	**1**	0	**1**	0	0	0	0	**1**
1	0	**1**	**1**	**1**	0	**1**	**1**	**1**	**1**
1	0	0	0	**1**	0	0	**1**	0	**1**
1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	0	**1**
1	**1**	**1**	0	0	0	0	0	0	**1**



```python
from collections import deque


def is_valid(x, y, maze, visited):
    """
    Проверяет, является ли клетка допустимой для посещения.
    """
    return (
        0 <= x < len(maze)
        and 0 <= y < len(maze[0])
        and maze[x][y] == 1
        and not visited[x][y]
    )


def bfs(maze, start, end):
    """
    Алгоритм поиска в ширину (BFS) для нахождения кратчайшего пути.
    """
    rows, cols = len(maze), len(maze[0])
    visited = [[False for _ in range(cols)] for _ in range(rows)]
    distance = [[-1 for _ in range(cols)] for _ in range(rows)]
    queue = deque([start])  # Очередь для BFS
    visited[start[0]][start[1]] = True
    distance[start[0]][start[1]] = 1

    while queue:
        x, y = queue.popleft()

        # Если достигли конечной клетки, возвращаем расстояние
        if (x, y) == end:
            return distance[x][y]

        # Проверяем соседей
        for dx, dy in DIRECTIONS:
            nx, ny = x + dx, y + dy
            if is_valid(nx, ny, maze, visited):
                visited[nx][ny] = True
                distance[nx][ny] = distance[x][y] + 1
                queue.append((nx, ny))

    return None  # Если путь не найден


if __name__ == '__main__':

    # Лабиринт (где 0 — стена, 1 — свободный путь)
    maze = [
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
        [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
        [1, 0, 1, 1, 1, 1, 1, 0, 0, 1],
        [1, 0, 1, 0, 0, 0, 1, 1, 1, 1],
        [1, 0, 1, 0, 1, 1, 1, 0, 0, 1],
        [1, 0, 1, 0, 1, 0, 0, 0, 0, 1],
        [1, 0, 1, 1, 1, 0, 1, 1, 1, 1],
        [1, 0, 0, 0, 1, 0, 0, 1, 0, 1],
        [1, 1, 1, 1, 1, 1, 1, 1, 0, 1],
        [1, 1, 1, 0, 0, 0, 0, 0, 0, 1],
    ]

    # Направления движения: вверх, вниз, влево, вправо
    DIRECTIONS = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    # Начальная и конечная позиции
    START = (0, 0)
    END = (6, 6)

    # Находим кратчайший путь
    path_weight = bfs(maze, START, END)

    # Вывод результата
    if path_weight is not None:
        print(f"Вес кратчайшего пути: {path_weight}")
    else:
        print("Путь не найден.")
```

Рисунок 4. Код программы

```
(base) elenamiheeva@MacBook-Pro-Elena AI_2 % python3 program/labirinth.py
Вес кратчайшего пути: 19
```

Рисунок 5. Результат работы программы

Нахождение минимального расстояния между начальными и конечными пунктами с использованием алгоритма поиска в ширину.

Был построен граф из 20 населенных пунктов Италии. Узлы данного графа представляют населённые пункты, а рёбра — дороги, соединяющие их. Вес каждого ребра соответствует расстоянию между этими пунктами. В качестве начального населенного пункта выбран Кунео, а конечного – Верона.

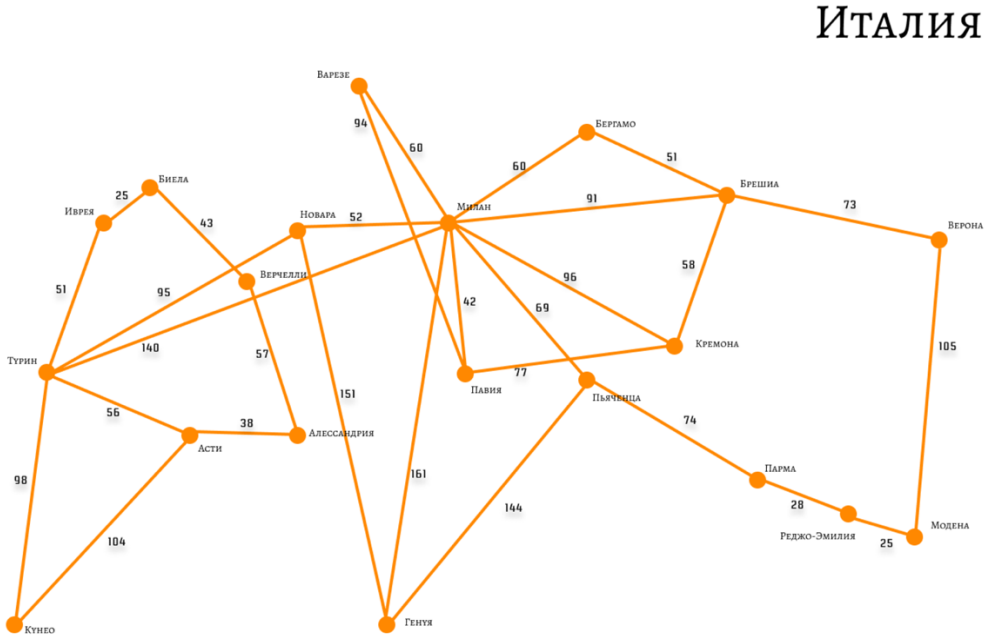


Рисунок 6. Граф из населенных пунктов

```python
from collections import deque


class Node:
    def __init__(self, state, parent=None, cost=0):
        self.state = state  # состояние узла
        self.parent = parent  # родитель узла
        self.cost = cost  # стоимость пути до узла


class Problem:
    def __init__(self, initial, goal, graph):
        self.initial = initial  # начальное состояние
        self.goal = goal  # конечное состояние
        self.graph = graph  # граф с соседями для каждого состояния

    def is_goal(self, state):
        return state == self.goal

    def expand(self, node):
        """Возвращает список соседей для текущего узла."""
        neighbors = []
        for neighbor, cost in self.graph[node.state]:
            neighbors.append(Node(neighbor, node, node.cost + cost))
        return neighbors

def breadth_first_search(problem):
    node = Node(problem.initial)

    if problem.is_goal(problem.initial):
        return node

    frontier = deque([node])  # Очередь для хранения узлов
    reached = {problem.initial}  # Множество посещенных узлов

    while frontier:
        node = frontier.popleft()  # Удаление узла из очереди

        # Проходим по соседям узла
        for child in problem.expand(node):
            s = child.state

            # Если цель найдена, возвращаем путь
            if problem.is_goal(s):
                return child

            if s not in reached:
                reached.add(s)
                frontier.append(child)

    return None  # Возвращаем None, если путь не найден

def breadth_first_search(problem):

    return None  # Возвращаем None, если путь не найден


# Восстановление пути до узла
def reconstruct_path(node):
    path = []
    while node:
        path.append(node.state)
        node = node.parent
    return path[::-1]  # Возвращаем путь в обратном порядке


if __name__ == '__main__':
    graph = {
        'Турин': [('Иврея', 51), ('Новара', 95), ('Милан', 140),
                  ('Асти', 56), ('Кунео', 98)],
        'Иврея': [('Турин', 51), ('Биелла', 25)],
        'Биелла': [('Иврея', 25), ('Верчелли', 43)],
        'Новара': [('Турин', 95), ('Милан', 52),
                   ('Генуя', 151), ('Павия', 42)],
        'Верчелли': [('Биелла', 43), ('Алессандрия', 57)],
        'Варезе': [('Милан', 60), ('Павия', 94)],
        'Милан': [('Турин', 140), ('Новара', 52), ('Варезе', 60),
                  ('Бергамо', 60), ('Брешиа', 91), ('Кремона', 91),
                  ('Пьянценца', 69), ('Генуя', 52), ('Павия', 42)],
        'Бергамо': [('Милан', 60), ('Брешиа', 51)],
        'Брешиа': [('Милан', 91), ('Верона', 73),
                   ('Кремона', 58), ('Бергамо', 51)],
        'Верона': [('Брешиа', 73), ('Модена', 105)],
        'Кремона': [('Брешиа', 58), ('Милан', 96), ('Павия', 77)],
        'Пьяченца': [('Милан', 69), ('Парма', 74), ('Генуя', 144)],
        'Парма': [('Пьяченца', 74), ('Реджо-Эмилия', 28)],
        'Реджо-Эмилия': [('Парма', 28), ('Модена', 25)],
        'Модена': [('Верона', 105), ('Реджо-Эмилия', 25)],
        'Генуя': [('Новара', 151), ('Пьячнца', 144), ('Милан', 161)],
        'Алессандрия': [('Верчелли', 57), ('Асти', 38)],
        'Асти': [('Турин', 56), ('Алессандрия', 38), ('Кунео', 104)],
        'Кунео': [('Турин', 98), ('Асти', 104)],
        'Павия': [('Варезе', 94), ('Милан', 42), ('Кремона', 77)]
    }

    problem = Problem('Кунео', 'Верона', graph)
    result = breadth_first_search(problem)

    if result:
        path = reconstruct_path(result)
        print("Минимальный путь:", path)
        print("Минимальное расстояние:", result.cost)
    else:
        print("Путь не найден")
```

Рисунок 7. Код программы для задания

```
(base) elenamiheeva@MacBook-Pro-Elena AI_2 % python3 program/breadth-first_search.py
Минимальный путь: ['Кунео', 'Турин', 'Милан', 'Брешиа', 'Верона']
Минимальное расстояние: 402
(base) elenamiheeva@MacBook-Pro-Elena AI_2 %
```

Рисунок 8. Результат работы программы

Ответы на контрольные вопросы:

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

В стратегии поиска в ширину используется очередь (FIFO - First In, First Out).

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

Это делается для того, чтобы гарантировать, что узлы будут обрабатываться в порядке их появления, что обеспечивает уровень за уровнем (глубину) расширения узлов.

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

Узлы, которые находятся в очереди дольше, будут расширены ранее, чем более новые узлы, если они находятся на одном уровне (глубине) поиска.

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

Следующим будет узел, который был добавлен в очередь сразу после корневого узла, то есть первый дочерний узел корневого узла.

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

Это позволяет находить кратчайшие пути к целевым узлам, так как узлы на меньшей глубине исследуются первыми, гарантируя, что цель будет достигнута с минимальным количеством шагов.

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

Временная сложность составляет O(b^d), где b - коэффициент разветвления (среднее количество детей) и d - глубина решения. Это объясняется тем, что каждый уровень добавляет b новых узлов.

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

Основной фактор - это количество узлов, которые находятся на текущем уровне поиска, т.е. O(b^d), особенно когда d является глубиной решения.

8. В каких случаях поиск в ширину считается полным?

Поиск в ширину считается полным, если есть конечная глубина, на которой можно достигнуть целевое состояние, и если пространство состояний не бесконечно.

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

Поиск в ширину хранит все узлы на текущем уровне, что может привести к большим требованиям к памяти, особенно в разветвленных графах с большой глубиной.

10. В чем заключается оптимальность поиска в ширину?

Поиск в ширину гарантирует нахождение кратчайшего пути к целевому узлу, если все переходы имеют одинаковую стоимость.

11. Какую задачу решает функция breadthfirstsearch?

Функция breadth_first_search решает задачу поиска целевого состояния в пространстве состояний, начиная с начального состояния.

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

Объект problem представляет собой описание проблемы (задачи), включая начальное состояние, функцию перехода и целевое состояние.

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

Он инициализирует корневой узел поиска, который представляет начальное состояние задачи.

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

Функция сразу вернет начальный узел как решение, так как нет необходимости в дальнейших поисках.

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

frontier использует очередь FIFO, чтобы обеспечивать порядок обработки узлов по уровням глубины, что является ключевым в стратегии поиска в ширину.

16. Какую роль выполняет множество reached?

Множество reached отслеживает все достигнутые состояния, чтобы избежать повторного рассмотрения уже исследованных узлов.

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

Это важно для предотвращения цикла и избыточной работы, а также для уменьшения использования памяти.

18. Какую функцию выполняет цикл while frontier?

Цикл while frontier отвечает за повторное извлечение и обработку узлов из очереди до тех пор, пока в ней есть узлы.

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

Извлеченный узел становится активным узлом для обработки, и с ним выполняются действия по его расширению.

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

Функция expand генерирует и возвращает всех дочерних узлов активного узла, основываясь на заданных правилах задачи.

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

Условие целевого состояния проверяется с использованием заранее определённой функции проверки, связанной с объектом проблемы. Если состояние узла не является целевым, то узел будет расширен, и его дочерние узлы будут добавлены в очередь для дальнейшей обработки.

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

Это может быть специфично для других стратегий поиска (например, для поиска в глубину), однако для поиска в ширину дочерние узлы добавляются в конец очереди.

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

Если решение не найдено, функция обычно возвращает None или аналогичный объект, который указывает на отсутствие решения.

24. Каково значение узла failure и когда он возвращается?

Узел failure может использоваться для сигнализации о том, что алгоритм не нашел решение, и обычно возвращается в конце выполнения, если ни один узел не удовлетворяет условиям.

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