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
По дисциплине
«Искусственный интеллект в профессиональной сфере»

Выполнил:
Арзютов Иван Владиславович

3 курс, группа ИТС-б-о-22-1,

11.03.02 «Инфокоммуникационные технологии и системы связи, очная форма обучения

_____
(подпись)

Проверил:

Воронкин Р.А., канд. тех. наук, доцент, доцент кафедры инфокоммуникаций

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Исследование поиска в ширину

Ссылка на репозиторий: https://github.com/Ivanuschka/2-LB

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

Порядок выполнения работы

Задание 1

Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT.

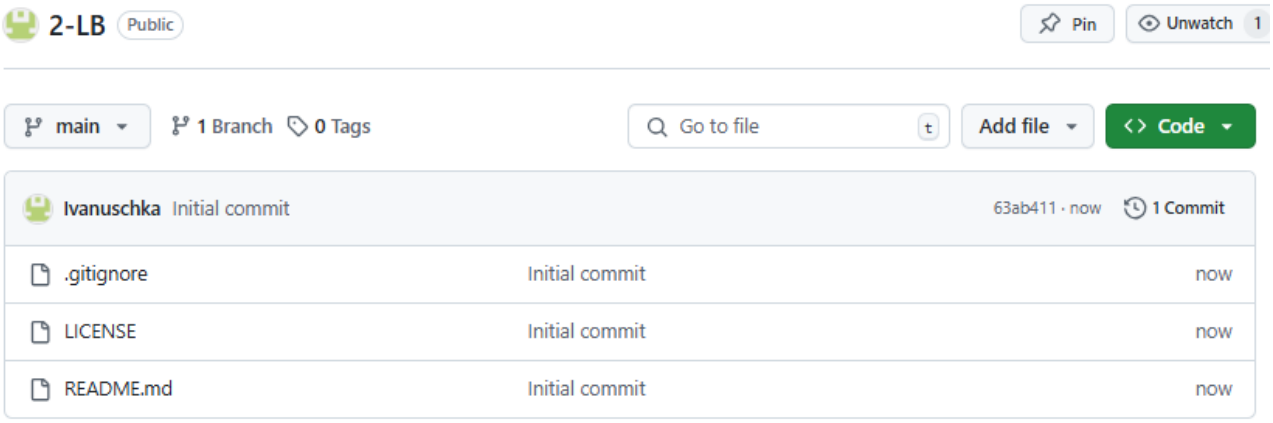


Рисунок 1. Создан новый репозиторий

Задание 2

Клонирование репозитория на свой компьютер.

```
C:\Users\ivana\OneDrive\Рабочий стол\ИИ\1 Лабораторная работа>git clone https://github.com/Ivanuschka/2-LB.git
Cloning into '2-LB'...
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.

C:\Users\ivana\OneDrive\Рабочий стол\ИИ\1 Лабораторная работа>
```

Рисунок 2. Клонирование репозитория

Задание 3

Проработка примера лабораторной работы

Пример.py > ...

```
1   def breadth_first_search(problem):
2       node = Node(problem.initial)
3       if problem.is_goal(problem.initial):
4           return node
5       frontier = FIFOQueue([node])
6       reached = {problem.initial}
7       while frontier:
8           node = frontier.pop()
9           for child in expand(problem, node):
10              s = child.state
11              if problem.is_goal(s):
12                  return child
13              if s not in reached:
14                  reached.add(s)
15                  frontier.appendleft(child)
16      return failure
```

Рисунок 3. Выполнение примера

Задание 4

Создание модулей для примера

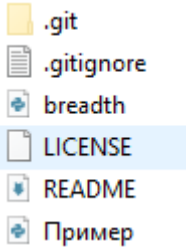
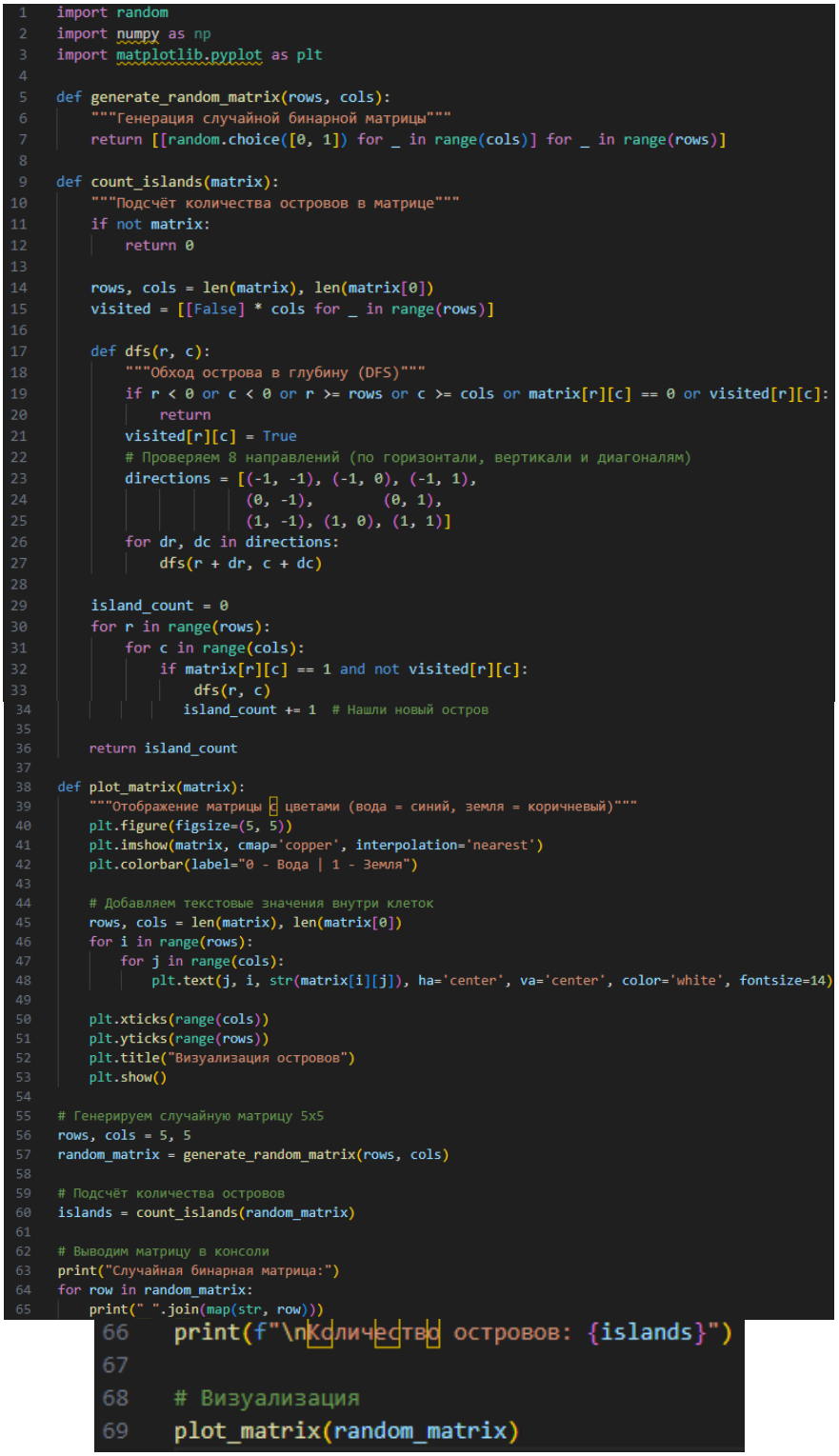


Рисунок 4. Модули в папке репозитория

Задание 5

Произвести расширенный подсчет количества островов в бинарной матрице.



```python
import random
import numpy as np
import matplotlib.pyplot as plt

def generate_random_matrix(rows, cols):
    """Генерация случайной бинарной матрицы"""
    return [[random.choice([0, 1]) for _ in range(cols)] for _ in range(rows)]

def count_islands(matrix):
    """Подсчёт количества островов в матрице"""
    if not matrix:
        return 0

    rows, cols = len(matrix), len(matrix[0])
    visited = [[False] * cols for _ in range(rows)]

    def dfs(r, c):
        """Обход острова в глубину (DFS)"""
        if r < 0 or c < 0 or r >= rows or c >= cols or matrix[r][c] == 0 or visited[r][c]:
            return
        visited[r][c] = True
        # Проверяем 8 направлений (по горизонтали, вертикали и диагоналям)
        directions = [(-1, -1), (-1, 0), (-1, 1),
                      (0, -1),         (0, 1),
                      (1, -1), (1, 0), (1, 1)]
        for dr, dc in directions:
            dfs(r + dr, c + dc)

    island_count = 0
    for r in range(rows):
        for c in range(cols):
            if matrix[r][c] == 1 and not visited[r][c]:
                dfs(r, c)
                island_count += 1  # Нашли новый остров

    return island_count

def plot_matrix(matrix):
    """Отображение матрицы с цветами (вода = синий, земля = коричневый)"""
    plt.figure(figsize=(5, 5))
    plt.imshow(matrix, cmap='copper', interpolation='nearest')
    plt.colorbar(label="0 - Вода | 1 - Земля")

    # Добавляем текстовые значения внутри клеток
    rows, cols = len(matrix), len(matrix[0])
    for i in range(rows):
        for j in range(cols):
            plt.text(j, i, str(matrix[i][j]), ha='center', va='center', color='white', fontsize=14)

    plt.xticks(range(cols))
    plt.yticks(range(rows))
    plt.title("Визуализация островов")
    plt.show()

# Генерируем случайную матрицу 5x5
rows, cols = 5, 5
random_matrix = generate_random_matrix(rows, cols)

# Подсчёт количества островов
islands = count_islands(random_matrix)

# Выводим матрицу в консоли
print("Случайная бинарная матрица:")
for row in random_matrix:
    print(" ".join(map(str, row)))
print(f"\nКоличество островов: {islands}")

# Визуализация
plot_matrix(random_matrix)
```

Рисунок 4. Код для определения матрицы и подсчёта островов

Задание 6.

```python
from collections import deque

def shortest_path(maze, start, end):
    rows, cols = len(maze), len(maze[0])
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]  # Вверх, вниз, влево, вправо
    queue = deque([(start[0], start[1], 0)])  # (x, y, шаги)
    visited = set()
    visited.add(start)

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

        if (x, y) == end:
            return steps  # Возвращаем количество шагов до выхода

        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < rows and 0 <= ny < cols and maze[nx][ny] == 1 and (nx, ny) not in visited:
                queue.append((nx, ny, steps + 1))
                visited.add((nx, ny))

    return -1  # Если путь не найден

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

start = (6, 0)
end = (4, 6)

result = shortest_path(maze, start, end)
print("Кратчайший путь от S к E:", result)
```

Рисунок 5. Код для определения кратчайшего пути для выхода

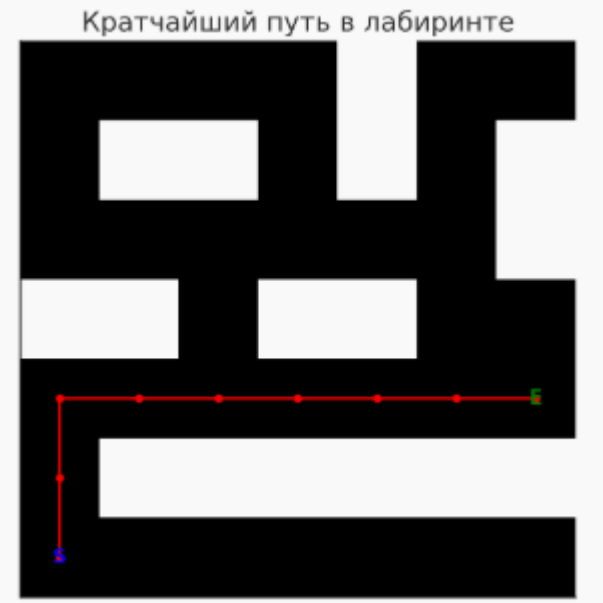


Рисунок 6. Графическая визуализация пути

Задание 6.

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.

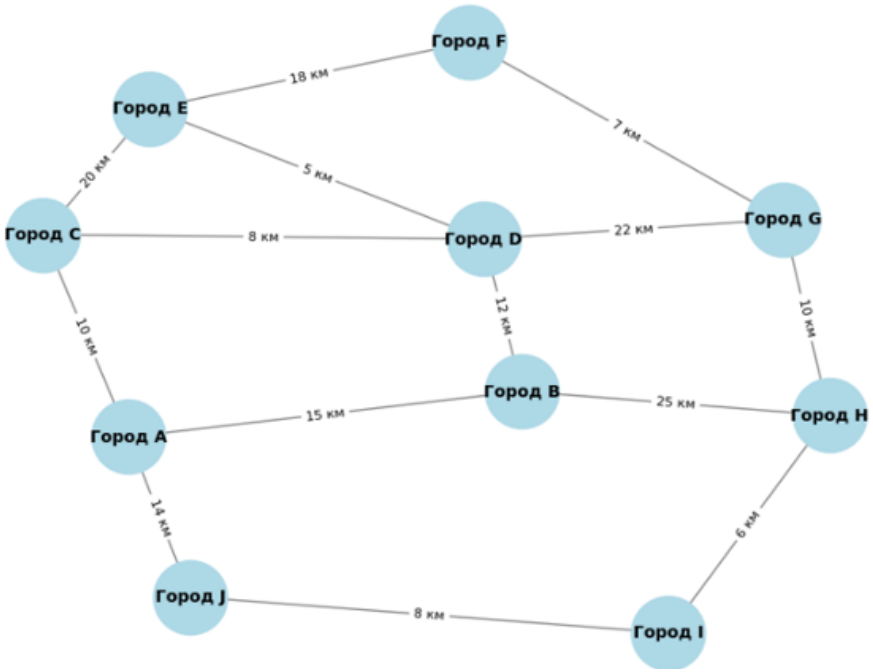


Рисунок 7. Граф городов

```python
from collections import deque

def bfs_shortest_path(graph, start, end):
    queue = deque([(start, 0)])  # (город, расстояние)
    visited = set()
    visited.add(start)

    while queue:
        node, distance = queue.popleft()
        if node == end:
            return distance

        for neighbor, weight in graph.get(node, []):
            if neighbor not in visited:
                queue.append((neighbor, distance + weight))
                visited.add(neighbor)

    return float('inf')  # Если путь не найден

graph = {
    'A': [('C', 10), ('J', 14), ('B', 15)],
    'B': [('A', 15), ('D', 12), ('H', 25)],
    'C': [('A', 10), ('D', 8), ('E', 20)],
    'D': [('C', 8), ('B', 12), ('E', 5), ('F', 18), ('G', 22)],
    'E': [('C', 20), ('D', 5), ('F', 18)],
    'F': [('D', 18), ('E', 18), ('G', 7)],
    'G': [('D', 22), ('F', 7), ('H', 10)],
    'H': [('G', 10), ('B', 25), ('I', 6)],
    'I': [('H', 6), ('J', 8)],
    'J': [('I', 8), ('A', 14)]
}
start = 'E'
end = 'I'
result = bfs_shortest_path(graph, start, end)
print(f"Кратчайший путь от {start} до {end}: {result} км")
```

Рисунок 8. Код программы для поиска в ширину

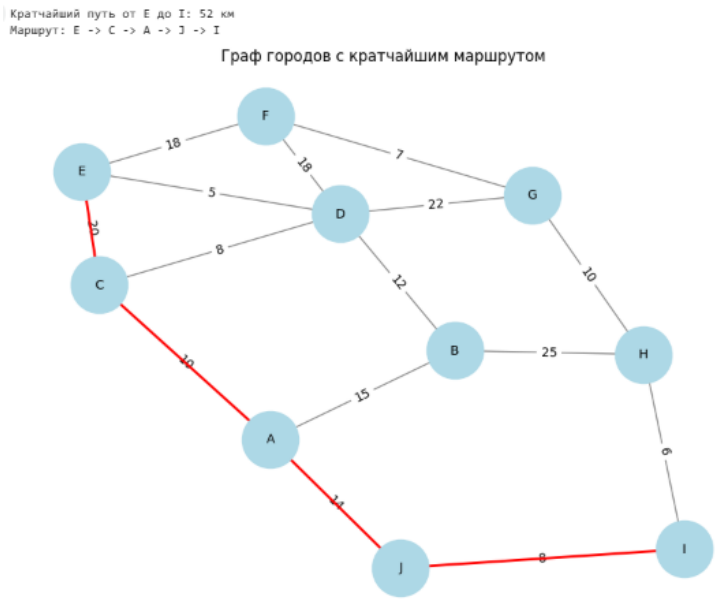


Рисунок 9. Визуализация кратчайшего пути при поиске в ширину

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

Ответы на контрольные вопросы

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

В стратегии поиска в ширину используется очередь (обычно реализованная как FIFO - First In, First Out). Это означает, что узлы обрабатываются в том порядке, в котором они были добавлены в очередь.

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

Новые узлы добавляются в конец очереди, чтобы гарантировать, что узлы, которые были добавлены в очередь раньше, будут обработаны первыми. Это соответствует принципу FIFO и позволяет исследовать узлы на каждом уровне глубины перед тем, как перейти к следующему уровню.

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

Узлы, которые дольше всего находятся в очереди, будут обработаны первыми, когда очередь будет извлечена. Это происходит потому, что BFS обрабатывает узлы в порядке их добавления, и по мере извлечения узлов из очереди части графа, ближайшие к корневому узлу, будут исследованы первыми.

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

Следующий узел, который будет расширен, будет первым узлом, добавленным в очередь после корневого. Это зависит от порядка, в котором дочерние узлы были добавлены в очередь.

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

Важно расширять узлы с наименьшей глубиной, чтобы гарантировать, что когда будет найдено решение, оно будет оптимальным, и вы получите кратчайший путь к целевому узлу.

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

Временная сложность BFS составляет O(b^d), где b — коэффициент разветвления (максимальное количество дочерних узлов для одного узла), а d

— глубина решения. Это означает, что время выполнения алгоритма растет экспоненциально с увеличением количества дочерних узлов и глубины решения.

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

Основной фактор, определяющий пространственную сложность алгоритма BFS, — это максимальное количество узлов, которые могут находиться в очереди одновременно. Это количество связано с коэффициентом разветвления и глубиной. Пространственная сложность также составляет O(b^d).

8. В каких случаях поиск в ширину считается полным?

Поиск в ширину считается полным, если он гарантирует нахождение решения, если оно существует. BFS полный, если граф конечен или если у нас есть возможность обнаруживать циклы и избегать их.

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

Поиск в ширину неэффективен с точки зрения памяти, потому что он хранит все узлы на текущем уровне и все узлы на следующем уровне в памяти, что может быстро потреблять большое количество памяти, особенно при большом коэффициенте разветвления.

10. В чем заключается оптимальность поиска в ширину?

Поиск в ширину оптимален в том смысле, что если существует решение, то он найдет его в кратчайшем возможном пути, потому что он исследует все узлы на одном уровне перед переходом на следующий уровень.

11. Какую задачу решает функция `breadth_first_search`?

Функция `breadth_first_search` решает задачу поиска решения в графе или дереве, начиная с корневого узла и исследуя узлы по уровням, чтобы найти целевой узел.

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

Объект `problem` представляет собой задачу, включающую в себя начальное состояние, цель и, возможно, функцию перехода, которая описывает, как переходить от одного состояния к другому.

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

функции?

Узел `Node(problem.initial)` инициализируется с начальным состоянием задачи и представляет собой стартовый узел для поиска. Он добавляется в очередь BFS и является базовой точкой для дальнейшего расширения поиска.

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

Если начальное состояние уже является целевым, алгоритм немедленно найдет решение, и будет возвращен этот узел без дальнейших расширений.

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

`frontier` использует структуру данных очереди (FIFO), поскольку это позволяет расширять узлы в порядке их добавления, что важно для стратегии поиска в ширину, где необходимо исследовать узлы по уровням их глубины.

16. Какую роль выполняет множество `reached`?

- Множество `reached` используется для отслеживания узлов, которые были уже обработаны или посещены, чтобы избежать повторных обработок и циклов.

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

Важно проверять, находится ли узел в множестве `reached`, чтобы избежать бесконечного цикла и повторной обработки узлов, что может привести к излишнему потреблению памяти и времени.

18. Какую функцию выполняет цикл `while frontier`?

Цикл `while frontier` выполняет основную работу алгоритма BFS, периодически извлекая узел из очереди, расширяя его и добавляя его дочерние узлы в очередь, пока не будет найдено решение или очередь не станет пустой.

19. Что происходит с узлом, который извлекается из очереди в строке

`node = frontier.pop()`?

Когда узел извлекается из очереди, он становится текущим узлом, который будет расширен. Это означает, что будут проверены все его дочерние узлы и добавлены в очередь для дальнейшего исследования.

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

Функция `expand(problem, node)` возвращает дочерние узлы для данного узла, раскрывая возможности перехода к новым состояниям.

21. Как определяется, что состояние узла является целевым?

Состояние узла определяется как целевое, если оно совпадает с состоянием, описанным в целевой функции задачи, указанной в объекте `problem`.

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

Если состояние узла не является целевым и ранее не было достигнуто, то его дочерние узлы будут добавлены в `frontier`, и узел будет добавлен в множество `reached` для последующего отслеживания.

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

Использование `appendleft` имеет смысл только в тех случаях, когда очередь реализована как двусторонняя. В типичном подходе BFS дочерние узлы добавляются в конец. Если вы хотите организовывать хүүхакы (на данный момент) в другой порядок, следует использовать аналогичные возможности, предоставляемые данными структурами.

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

Если решение не найдено, функция `breadth_first_search` обычно возвращает `None`, `failure`, или аналогичные значения, специфичные для реализации, указывающие на то, что целевой узел не был найден.

25. Каково значение узла `failure` и когда он возвращается?

Узел `failure` представляет собой специальный флаг или значение, указывающее на то, что алгоритм не нашёл целевой узел. Он возвращается в

случае, если существуют все возможные узлы и все они были исследованы, но целевого узла не оказалось.