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:
Быковская Стефания Станиславовна
3 курс, группа ИТС-б-о-22-1,
11.03.02 «Инфокоммуникационные технологии и системы связи»

_____
(подпись)

Проверил: доцент департамента цифровых, робототехнических систем и электроники института перспективной инженерии
Воронкин Роман Александрович

_____
(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:

Решите задания лабораторной работы с помощью языка программирования Python и элементов программного кода лабораторной работы 1 Проверьте правильность решения каждой задачи на приведенных тестовых примерах.

```python
class Problem:
    def __init__(self, matrix):
        self.matrix = matrix
        self.rows = len(matrix)
        self.cols = len(matrix[0]) if self.rows > 0 else 0

    def actions(self, state):
        # Возвращает возможные действия (соседние клетки)
        row, col = state
        directions = [
            (-1, -1), (-1, 0), (-1, 1),
            (0, -1),           (0, 1),
            (1, -1), (1, 0), (1, 1)
        ]
        for dr, dc in directions:
            new_row, new_col = row + dr, col + dc
            if 0 <= new_row < self.rows and 0 <= new_col < self.cols:
                if self.matrix[new_row][new_col] == 1:
                    yield (new_row, new_col)

    def result(self, state, action):
        return action

    def i_goal(self, state):
        return False  # В данной задаче не требуется проверка на цель
```

Рисунок 1. Класс Problem для поиска в ширину

```python
class Node:
    def __init__(self, state, parent=None):
        self.state = state
        self.parent = parent

    def __repr__(self):
        return f'<Node(state={self.state})>'
    def __lt__(self, other): return self.state < other.state

class PriorityQueue:
    def __init__(self, items=(), key=lambda x: x) -> None:
        self.key = key
        self.items = []
        for item in items:
            self.add(item)

    def add(self, item):
        pair = (self.key(item), item)
        heapq.heappush(self.items, pair)

    def pop(self):
        return heapq.heappop(self.items)[1]

    def top(self): return self.items[0][1]

    def __len__(self): return len(self.items)
```

Рисунок 2. Очередь и узел для поиска в ширину

Для задачи «Расширенный подсчет количества островов в бинарной матрице» подготовить собственную матрицу, для которой с помощью разработанной в предыдущем пункте программы, подсчитать количество островов.

```
def breadth_first_search(problem):
    """Perform a breadth-first search on the given problem."""
    island_count = 0
    visited = set()

    for r in range(problem.rows):
        for c in range(problem.cols):
            if problem.matrix[r][c] == 1 and (r, c) not in visited:
                # Запускаем BFS для новой единицы
                node = Node((r, c))
                queue = PriorityQueue()  # Используем PriorityQueue
                queue.add(node)  # Добавляем узел в очередь
                visited.add(node.state)
                island_count += 1

                while len(queue) > 0:
                    current_node = queue.pop()  # Извлекаем узел из очереди
                    for action in problem.actions(current_node.state):
                        if action not in visited:
                            visited.add(action)
                            queue.add(Node(action))  # Добавляем новый узел в очередь

    return island_count
```

Рисунок 3. Подсчёт количества островов

```
84
85  if __name__ == '__main__':
86      # Пример использования
87      matrix = [
88          [1, 1, 0, 1, 0],
89          [0, 1, 0, 0, 0],
90          [1, 0, 0, 1, 1],
91          [0, 0, 0, 0, 0],
92          [1, 0, 1, 0, 1]
93      ]
94
95      problem = Problem(matrix)
96      print(breadth_first_search(problem))
97
```

Рисунок 4. Результат работы

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.

```python
def breadth_first_search(problem):
    start_node = Node(problem.start)
    queue = FIFOQueue()
    queue.add(start_node)
    visited = set()
    visited.add(start_node.state)
    parent_map = {start_node.state: None}  # Для отслеживания пути

    while not queue.is_empty():
        current_node = queue.pop()

        # Проверка, достигли ли мы цели
        if current_node.state == problem.goal:
            return reconstruct_path(parent_map, current_node.state)

        for action in problem.actions(current_node.state):
            if action not in visited:
                visited.add(action)
                queue.add(Node(action, current_node))  # Добавляем узел с ссылкой на родителя
                parent_map[action] = current_node.state  # Сохраняем родителя

    return None  # Если путь не найден
```

Рисунок 5. Поиск пути в лабиринте

```python
if __name__ == '__main__':
    # Пример использования
    matrix = [
        [1, 1, 1, 1, 1, 0, 0, 1, 1, 1],
        [0, 1, 1, 1, 1, 1, 0, 1, 0, 1],
        [0, 0, 1, 0, 1, 1, 1, 0, 0, 1],
        [1, 0, 1, 1, 1, 0, 1, 1, 0, 1],
        [0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
        [1, 0, 1, 1, 1, 0, 0, 1, 1, 0],
        [0, 0, 0, 0, 1, 0, 0, 1, 0, 1],
        [0, 1, 1, 1, 1, 1, 1, 1, 0, 0],
        [1, 1, 1, 1, 1, 0, 0, 1, 1, 1],
        [0, 0, 1, 0, 0, 1, 1, 0, 0, 1],
    ]

    start = (0, 0)  # Начальная точка
    goal = (7, 5)   # Конечная точка

    problem = Problem(matrix, start, goal)
    path = breadth_first_search(problem)

    if path:
        print("Кратчайший путь:", path)  # Вывод: кратчайший путь
        print("Длинна: ", len(path) - 1)
    else:
        print("Путь не найден.")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

C:\Users\Лео\Desktop\ИИ\ii-2>python .\prog\3.py
Кратчайший путь: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (3, 3), (4, 3), (5, 3), (5, 4), (6, 4), (7, 4), (7, 5)]
Длинна:  12

C:\Users\Лео\Desktop\ИИ\ii-2>
```

Рисунок 6. Результат работы

Для построенного графа лабораторной работы 1 напишите программу на языке программирования Python, которая с помощью алгоритма поиска в

ширину находит минимальное расстояние между начальным и конечным пунктами. Сравните найденное решение с решением, полученным вручную.

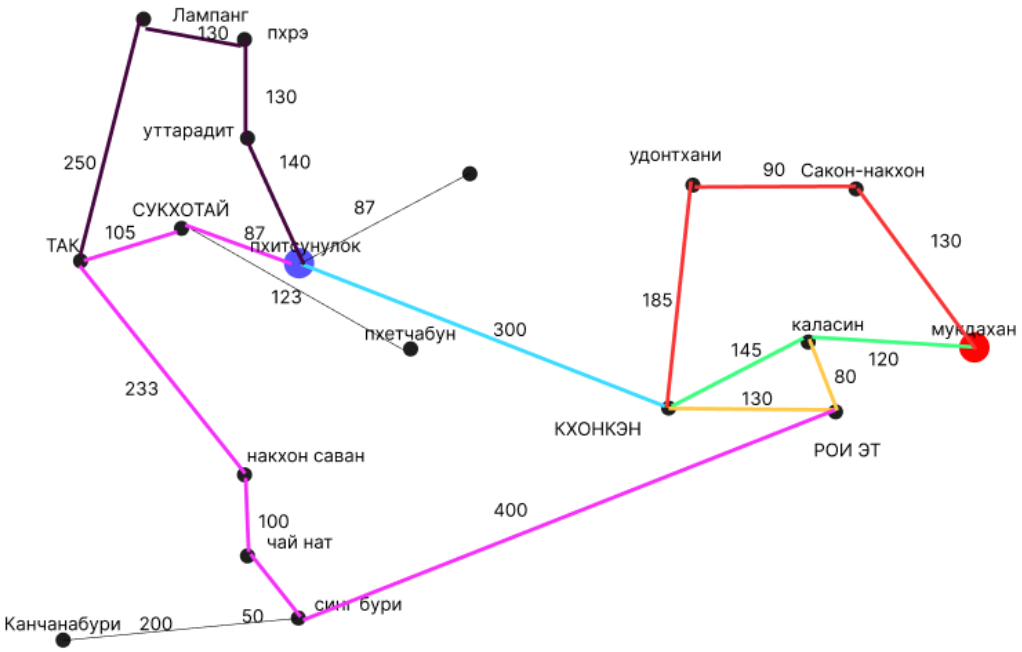

Рисунок 7. Возможные пути

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

from collections import deque
import math

class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, u, v, weight):
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        self.graph[u].append((v, weight))
        self.graph[v].append((u, weight))  # Для неориентированного графа

    def bfs(self, start, goal):
        queue = deque([(start, 0)])  # (вершина, текущая сумма весов)
        visited = set()
        visited.add(start)

        while queue:
            current_node, current_distance = queue.popleft()

            # Проверка, достигли ли мы цели
            if current_node == goal:
                return current_distance

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

        return -1  # Если путь не найден
```

Рисунок 8. Класс поиска оптимального пути по графу

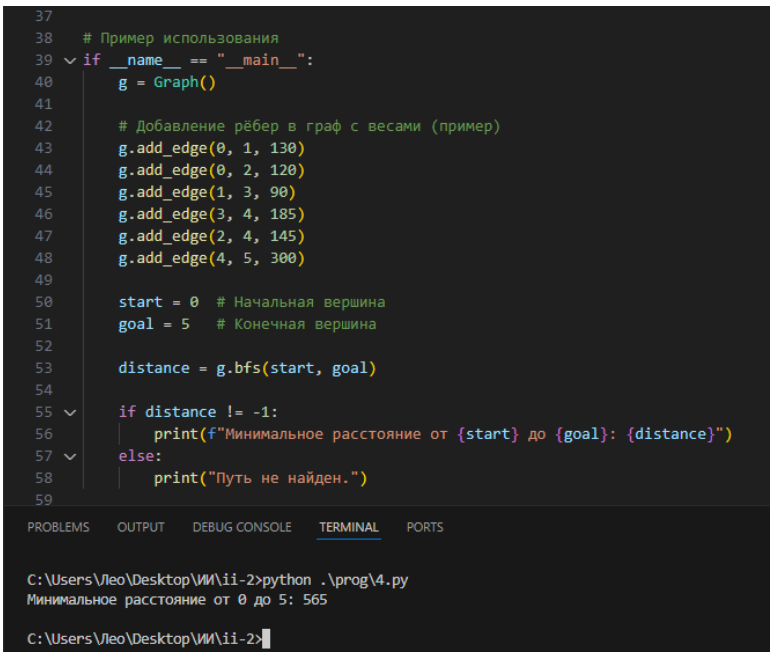

Рисунок 8. Поиск пути

Контрольные вопросы:

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

BFS использует очередь FIFO (First-In-First-Out). Это обеспечивает расширение узлов в порядке их обнаружения, что критично для обхода узлов уровень за уровнем.

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

Новые узлы добавляются в конец очереди для поддержания порядка FIFO. Это обеспечивает расширение узлов в порядке их обнаружения, позволяя BFS исследовать все узлы на текущем уровне, прежде чем перейти к узлам следующего уровня.

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

Узлы, которые находятся в очереди дольше всего, расширяются первыми. Это происходит потому, что BFS обрабатывает узлы в порядке их добавления в очередь, что позволяет исследовать узлы на текущем уровне полностью, прежде чем перейти к следующему уровню.

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

Следующим узлом, который будет расширен после корневого узла, станет один из его непосредственных детей. BFS расширяет узлы уровень за уровнем, поэтому он исследует всех детей корневого узла, прежде чем углубляться в дерево.

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

Расширение узлов с наименьшей глубиной гарантирует, что BFS находит кратчайший путь в неориентированном графе. Исследуя все узлы на текущем уровне перед тем, как перейти к более глубоким, BFS удостоверяется, что при первом достижении узла цели найден самый короткий путь.

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 (среднее количество детей у узла), а d — глубина самого поверхностного решения. Эта сложность возникает из-за того, что BFS исследует все узлы на каждом уровне глубины.

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

Пространственная сложность BFS в первую очередь определяется количеством узлов, хранящихся в очереди, что может быть так же велико, как количество узлов на глубочайшем уровне, т.е. $O(b^d)$.

8 В каких случаях поиск в ширину считается полным?

BFS считается полным, если коэффициент разветвления конечен и решение существует. Он в конечном итоге исследует все узлы на каждом уровне, обеспечивая нахождение решения, если оно существует.

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

BFS может быть неэффективен по памяти, поскольку он хранит все узлы на текущем уровне в очереди. Для больших графов с высоким коэффициентом разветвления это может привести к значительному потреблению памяти.

10 В чем заключается оптимальность поиска в ширину?

BFS оптимален для нахождения кратчайшего пути в неориентированных графах, потому что он исследует все узлы на текущем уровне перед тем, как углубиться, что гарантирует, что первый раз, когда он достигает узла цели, он нашел кратчайший путь.

11 Какую задачу решает функция breadth_first_search?

Функция breadth_first_search предназначена для нахождения кратчайшего пути от начального узла до узла цели в графе, предполагая, что все рёбра имеют равный вес.

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

Объект problem обычно представляет граф или пространство поиска, включая начальное состояние, целевое состояние и методы для генерации потомков и проверки условий достижения цели.

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

Node(problem.initial) инициализирует поиск с начальным состоянием задачи, инкапсулируя его в структуре узла, который может быть расширен и отслежен.

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

Если начальное состояние является целью, функция может немедленно вернуть начальный узел, так как кратчайший путь тривиально равен нулю.

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

frontier использует очередь FIFO, чтобы гарантировать, что узлы расширяются в порядке их обнаружения, что важно для BFS, чтобы исследовать узлы уровень за уровнем.

16 Какую роль выполняет множество reached ?

Множество reached отслеживает все узлы, которые были обнаружены и добавлены в очередь, предотвращая алгоритм от повторного посещения узлов и выхода в бесконечный цикл.

17 Почему важно проверять, находится ли состояние в множестве reached

Проверка, находится ли состояние в множестве reached, предотвращает многократную обработку одного и того же узла, что оптимизирует производительность и избегает циклов.

18 Какую функцию выполняет цикл while frontier?

Цикл while frontier продолжает процесс поиска, расширяя узлы из очереди до тех пор, пока не будет найдено решение или очередь не опустеет, что указывает на отсутствие решения

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

Узел удаляется из очереди и становится текущим узлом, который будет расширен. Его потомки будут сгенерированы и потенциально добавлены в очередь.

20 Какова цель функции expand(problem, node)?

Функция expand генерирует потомков текущего узла, создавая новые узлы для каждого возможного действия из текущего состояния.

21 Как определяется, что состояние узла является целевым?

Состояние узла считается целевым, если оно удовлетворяет условию цели, определенному в задаче, обычно проверяемому с помощью метода, такого как problem.i_goal(state).

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, оно добавляется в очередь для будущего расширения, а его состояние помечается как достигнутое.

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

В типичной реализации BFS узлы добавляются в конец очереди. Если используется appendleft, это подразумевает поведение LIFO, что не является стандартным для BFS. Обычно используется append.

24 Что возвращает функция breadth_first_search , если решение не найдено?

Если решение не найдено, функция обычно возвращает специальное значение, указывающее на неудачу, такое как None или предопределённый узел failure.

25 Каково значение узла failure и когда он возвращается?

Узел failure — это заполнитель, указывающий на то, что решение не найдено. Он возвращается, когда поиск исчерпывает все возможности без достижения цели.

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