Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамента цифровых, роботехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
дисциплины «Искусственный интеллект в профессиональной сфере»

Выполнила:
Гайчук Дарья Дмитриевна
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)

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

Ссылка на репозиторий: https://github.com/Ichizuchi/PR_AI_002

Порядок выполнения работы:

1. Создала новый репозиторий и клонировала его на свой компьютер.

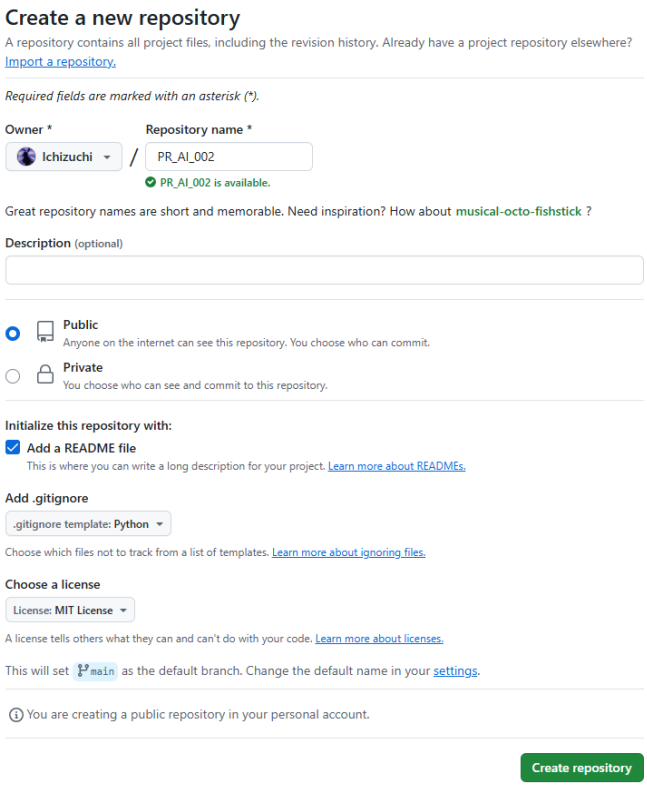

Рисунок 1. Создание репозитория

2. Клонировала репозиторий на свой компьютер.

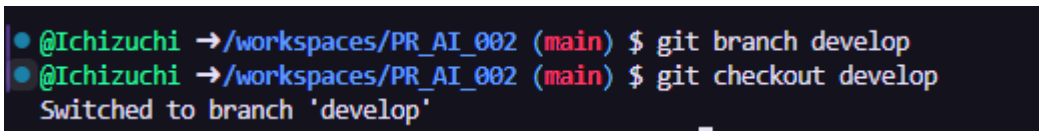

Рисунок 2. Модель ветвления git-flow

3. Создала виртуальное окружение Anaconda с именем репозитория.

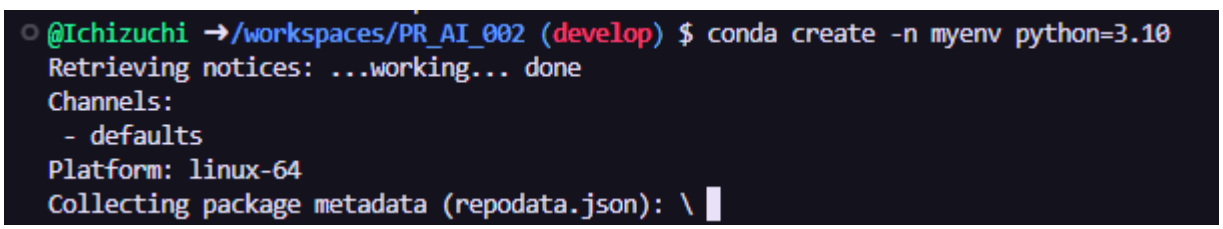

Рисунок 3. Создание виртуального окружения

Вариант 3

Задание №1. **(Подсчет островов)**: Создать матрицу и реализовать алгоритм подсчета островов.

```python
# Подсчет количества островов
def count_islands(matrix):
    def dfs(x, y):
        if x < 0 or y < 0 or x >= len(matrix) or y >= len(matrix[0]) or matrix[x][y] == 0:
            return
        matrix[x][y] = 0
        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (1, 1), (-1, 1), (1, -1)]:
            dfs(x + dx, y + dy)

    count = 0
    for i in range(len(matrix)):
        for j in range(len(matrix[0])):
            if matrix[i][j] == 1:
                count += 1
                dfs(i, j)
    return count

custom_matrix = [
    [1, 1, 0, 0],
    [0, 1, 0, 0],
    [0, 0, 1, 1],
    [0, 0, 0, 1]
]
print("Количество островов:", count_islands(custom_matrix))

```

Рисунок 6. Код программы

```
@Ichizuchi ➜/workspaces/PR_AI_002 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/PR_AI_002/Tasks/Task_1.py
Количество островов: 1
```

Рисунок 7. Вывод результата

Задание №2. **(Кратчайший путь в лабиринте)**: Создать лабиринт, задать начальные и конечные позиции, реализовать поиск кратчайшего пути.

asks > Task_2.py > ...

```python
# Поиск кратчайшего пути
from collections import deque

def shortest_path(maze, start, goal):
    rows, cols = len(maze), len(maze[0])
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    queue = deque([(start, 0)])
    visited = set()

    while queue:
        (x, y), distance = queue.popleft()
        if (x, y) == goal:
            return distance
        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:
                visited.add((nx, ny))
                queue.append(((nx, ny), distance + 1))
    return -1

# Пример использования
custom_maze = [
    [1, 1, 0, 0],
    [1, 1, 0, 1],
    [0, 1, 1, 1],
    [0, 0, 0, 1]
]
start_position = (0, 0)
goal_position = (3, 3)
print("Длина кратчайшего пути:", shortest_path(custom_maze, start_position, goal_position))
```

Рисунок 8. Код программы

```
@Ichizuchi ➜/workspaces/PR_AI_002 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/PR_AI_002/Tasks/Task_2.py
Длина кратчайшего пути: 6
```

Рисунок 9. Вывод задания №2

Задание №3. **Поиск в ширину для графов**: Использовать алгоритм из лабораторной работы 1 для поиска минимального расстояния.

```python
graph = {
    "Стамбул": [("Бурса", 155), ("Эскишехир", 480), ("Анкара", 450)],
    "Бурса": [("Стамбул", 155), ("Анкара", 480), ("Маниса", 70)],
    "Эскишехир": [("Стамбул", 480), ("Анкара", 230)],
    "Анкара": [("Эскишехир", 230), ("Стамбул", 450), ("Бурса", 480), ("Сивас", 320), ("Конья", 260)],
    "Маниса": [("Бурса", 70), ("Измир", 330)],
    "Измир": [("Маниса", 330), ("Анталья", 460)],
    "Конья": [("Анкара", 260), ("Анталья", 290), ("Мерсин", 460)],
    "Анталья": [("Измир", 460), ("Конья", 290)],
    "Мерсин": [("Конья", 460), ("Адана", 70)],
    "Адана": [("Мерсин", 70), ("Газантеп", 220)],
    "Газантеп": [("Адана", 220), ("Шанлыурфа", 150)],
    "Шанлыурфа": [("Газантеп", 150), ("Диярбакыр", 180)],
    "Диярбакыр": [("Шанлыурфа", 180), ("Ван", 320)],
    "Ван": [("Диярбакыр", 320), ("Эрзурум", 370)],
    "Эрзурум": [("Ван", 370), ("Трабзон", 250)],
    "Трабзон": [("Эрзурум", 250), ("Самсун", 330)],
    "Самсун": [("Трабзон", 330), ("Сивас", 410)],
    "Сивас": [("Самсун", 410), ("Анкара", 320), ("Кайсери", 240)],
    "Кайсери": [("Сивас", 240), ("Малатья", 240)],
    "Малатья": [("Кайсери", 240), ("Шанлыурфа", 90)]
}

def bfs_shortest_path(graph, start, goal):
    # Очередь для хранения путей и начальной точки
    queue = deque([(start, [start])])
    visited = set()

    while queue:
        (vertex, path) = queue.popleft()
        if vertex in visited:
            continue

        visited.add(vertex)

        if vertex == goal:
            return path

        for (neighbor, distance) in graph.get(vertex, []):
            if neighbor not in visited:
                queue.append((neighbor, path + [neighbor]))

    return None  # Путь не найден

start_city = "Стамбул"
goal_city = "Ван"
path = bfs_shortest_path(graph, start_city, goal_city)
print("Кратчайший путь из", start_city, "в", goal_city, ":", path)
```

Рисунок 10. Код программы

```
@Ichizuchi →/workspaces/PR_AI_002 (develop) $ /home/codespace/.python/current/bin/python3 /workspaces/PR_AI_002/Tasks/Task_3.py
Кратчайший путь из Стамбул в Ван : ['Стамбул', 'Анкара', 'Сивас', 'Самсун', 'Трабзон', 'Эрзурум', 'Ван']
```

Рисунок 11. Вывод задания №3

```
@Ichizuchi →/workspaces/PR_AI_002 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
@Ichizuchi →/workspaces/PR_AI_002 (main) $ git merge develop
Updating a71b217..f2ed92e
Fast-forward
 Tasks/Task_1.py | 24 ++++++++++++++++++++++
 Tasks/Task_2.py | 30 ++++++++++++++++++++++++++++++
 Tasks/Task_3.py | 52 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 3 files changed, 106 insertions(+)
 create mode 100644 Tasks/Task_1.py
 create mode 100644 Tasks/Task_2.py
 create mode 100644 Tasks/Task_3.py
@Ichizuchi →/workspaces/PR_AI_002 (main) $
```

Рисунок 12. Слияние веток

Ответы на контрольные вопросы

1. Какой тип очереди используется 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?

Чтобы обеспечить обработку узлов в порядке их добавления и сохранить свойства поиска в ширину, когда сначала обрабатываются узлы меньшей глубины.

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

Они обрабатываются первыми, так как очередь FIFO извлекает элементы в порядке их добавления.

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

Узел, который находится на одном уровне глубины с корнем и был добавлен в очередь первым.

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

Чтобы гарантировать нахождение оптимального решения (пути с минимальным количеством шагов) в случае равномерной стоимости шагов.

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

Временная сложность пропорциональна O(bd)O(b^d)O(bd), где bbb — коэффициент разветвления (среднее число дочерних узлов), а ddd — глубина решения.

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

Число узлов, которые хранятся в памяти одновременно, что также пропорционально O(bd)O(b^d)O(bd).

8. В каких случаях поиск в ширину считается полным?

Когда дерево поиска конечно или если пространство поиска бесконечно, но существует гарантированная глубина, на которой находится решение.

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

Потому что для хранения всех узлов на одном уровне требуется много памяти, особенно при большом коэффициенте разветвления и глубине решения.

10. В чем заключается оптимальность поиска в ширину?

Он гарантирует нахождение пути минимальной длины (если все шаги имеют одинаковую стоимость).

11. Какую задачу решает функция breadth_first_search?

Она реализует поиск в ширину для нахождения пути от начального состояния к целевому в пространстве состояний.

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

Объект, описывающий задачу, содержащий начальное состояние, целевое состояние, возможные действия и правила переходов.

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

Чтобы создать корневой узел дерева поиска, соответствующий начальному состоянию задачи.

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

Функция немедленно вернет этот узел как решение.

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

Используется очередь FIFO для обработки узлов в порядке их добавления, что обеспечивает поиск в ширину.

16. Какую роль выполняет множество reached?

Оно хранит состояния, которые уже были посещены, чтобы избежать повторной обработки.

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

Чтобы предотвратить зацикливание и дублирование узлов в очереди.

18. Какую функцию выполняет цикл while frontier?

Он последовательно извлекает узлы из очереди для обработки, пока не будет найдено решение или очередь не станет пустой.

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

Этот узел расширяется для генерации его потомков.

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

Сгенерировать дочерние узлы текущего узла, используя возможные действия, определенные задачей.

21. Как определяется, что состояние узла является целевым?

Сравнивается состояние узла с целевым состоянием задачи.

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

Узел добавляется в очередь frontier, а его состояние помечается как достигнутое в reached.

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

Ошибка в вопросе. Для очереди FIFO дочерние узлы добавляются в конец с помощью append(child).

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

Функция возвращает значение failure или None.

25. Каково значение узла и когда он возвращается?

Узел представляет собой конечное состояние и путь к нему; возвращается, когда это состояние является целевым.

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