

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:  
Магомедов Имран Борисович  
3 курс, группа «Программная инженерия»,  
направленность (профиль) «Разработка  
и сопровождение программного  
обеспечения», очная форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р.А., доцент департамента  
цифровых, робототехнических систем и  
электроники

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

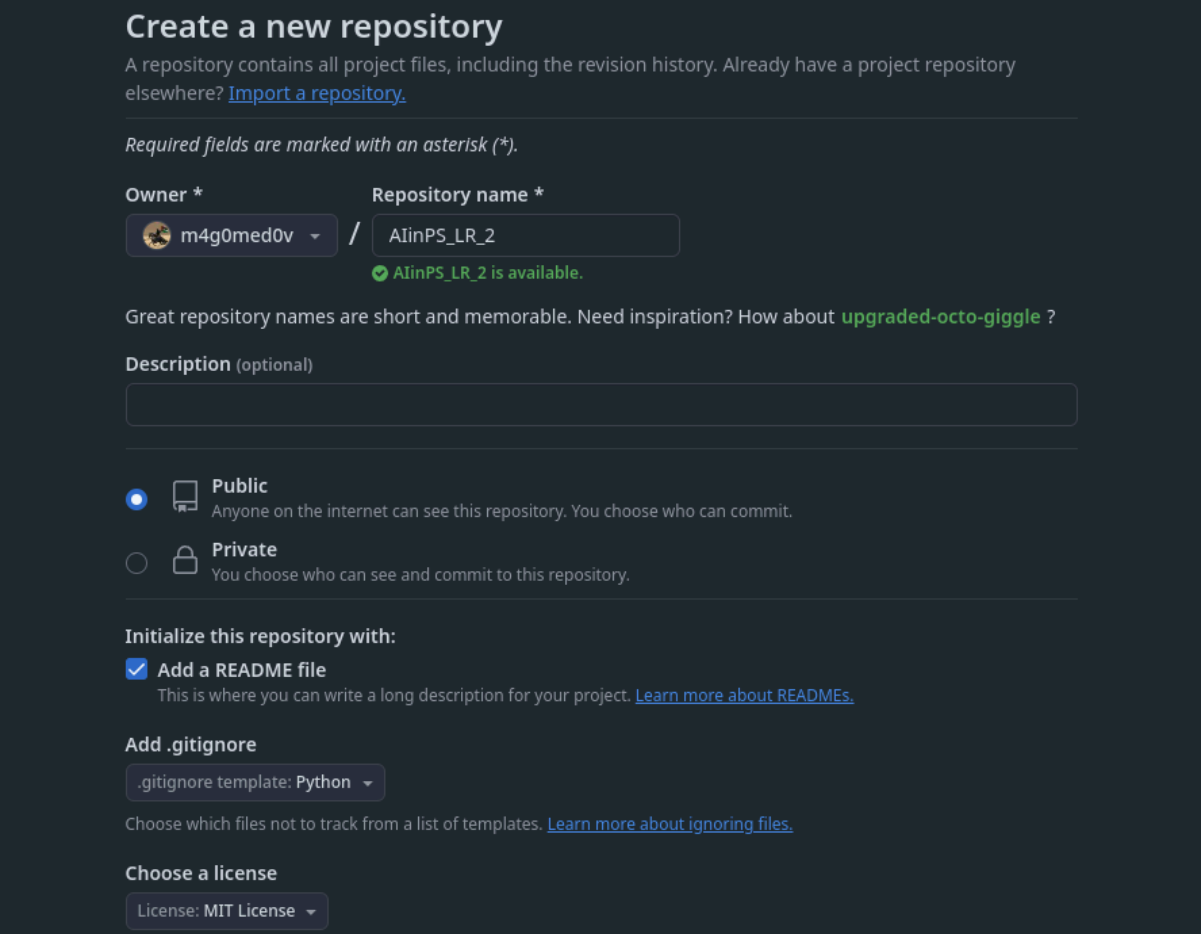
Ставрополь, 2024 г.

**Тема:** Исследование поиска в ширину.

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

### Методика выполнения работы

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.



The screenshot shows the GitHub 'Create a new repository' page. At the top, it says 'Create a new repository' and explains that a repository contains all project files, including the revision history. Below this, it states 'Required fields are marked with an asterisk (\*)'. The 'Owner' field is set to 'm4g0med0v' and the 'Repository name' field is set to 'AInPS\_LR\_2'. A green checkmark indicates that 'AInPS\_LR\_2 is available'. Below the name fields, there is a suggestion for repository names: 'Great repository names are short and memorable. Need inspiration? How about **upgraded-octo-giggle** ?'. The 'Description' field is optional and currently empty. The 'Visibility' section has two options: 'Public' (selected) and 'Private'. The 'Initialize this repository with:' section has a checked box for 'Add a README file'. The 'Add .gitignore' section has a dropdown menu set to '.gitignore template: Python'. The 'Choose a license' section has a dropdown menu set to 'License: MIT License'.

3. Выполните клонирование созданного репозитория.

```
~/ncfu git clone https://github.com/m4g0med0v/AIinPS_LR_2.git
Клонирование в «AIinPS_LR_2»...
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)
Получение объектов: 100% (5/5), готово.
```

4. Проработайте примеры лабораторной работы. Создайте для каждого примера отдельный модуль языка Python. Зафиксируйте изменения в репозитории.

```
breadth_first_search.py U X
1 from dependencies.handlers import expand
2
3 from .enums import failure
4 from .node import Node
5 from .queue import FIFOQueue
6
7
8 def breadth_first_search(problem):
9     """
10     Выполняет поиск в ширину (Breadth-First Search, BFS) для решения
11     заданной задачи.
12
13     Алгоритм работает следующим образом:
14     1. Создаётся начальный узел из начального состояния задачи.
15     2. Если начальное состояние уже является целевым, алгоритм завершает
16        работу и возвращает этот узел.
17     3. Используется очередь FIFO для хранения узлов на границе (frontier),
18        которые нужно исследовать.
19     4. Используется множество reached для отслеживания посещённых состояний и
20        предотвращения заикливания.
21     5. В цикле для каждого узла:
22        - Проверяется, является ли его состояние целевым.
23        - Если состояние узла ещё не было исследовано, оно добавляется в
24          множество reached и очередь frontier.
25     6. Если целевое состояние не найдено, возвращается значение failure.
26
27     :param problem: Экземпляр задачи, содержащий:
28        - initial: начальное состояние,
29        - is_goal(state): метод проверки целевого состояния.
30     :return: Узел, представляющий целевое состояние, либо константа failure,
31             если решение не найдено.
32     """
33     # Инициализация начального узла с состоянием начальной задачи.
```

```

~ /ncfu/AIinPS_LR_2 main +8 !3 git add .

~ /ncfu/AIinPS_LR_2 main +8 git commit -m "added dependencies"
check yaml.....Passed
fix end of files.....Passed
trim trailing whitespace.....Passed
ruff.....Passed
ruff-format.....Passed
[main 4eb58d2] added dependencies
8 files changed, 310 insertions(+)
create mode 100644 .pre-commit-config.yaml
create mode 100644 dependencies/__init__.py
create mode 100644 dependencies/breadth_first_search.py
create mode 100644 dependencies/enums.py
create mode 100644 dependencies/handlers.py
create mode 100644 dependencies/node.py
create mode 100644 dependencies/problem.py
create mode 100644 dependencies/queue.py

```

5. Решите задания лабораторной работы с помощью языка программирования Python и элементов программного кода лабораторной работы 1 (имя файла начинается с PR.AI.001.). Проверьте правильность решения каждой задачи на приведенных тестовых примерах.

```

60 # Создадим конкретную задачу, наследуя Problem
61 class SimpleGraphProblem(Problem):
62     def __init__(self, initial, goal, graph, **kws):
63         # Инициализация задачи с начальным и целевым состоянием, а также графом
64         super().__init__(initial, goal, **kws)
65         self.graph = graph
66
67     def actions(self, state):
68         """Возвращает список всех доступных переходов для данного состояния"""
69         # Возвращаем все соседние города
70         return self.graph.get(state, {})
71
72     def result(self, state, action):
73         """Возвращает новое состояние при применении действия"""
74         # Переход в указанный соседний город
75         return action
76
77     def action_cost(self, s, a, s1):
78         """Стоимость действия - в этом случае всегда 1"""
79         # Стоимость перехода (расстояние между городами)
80         return 1

```

```

83 if __name__ == "__main__":
84     # Начальное и целевое состояние задачи
85     initial_state = "Знаменское"
86     goal_state = "Леваши"
87
88     # Создаем объект задачи
89     problem = SimpleGraphProblem(
90         initial=initial_state, goal=goal_state, graph=graph
91     )
92
93     # Используем поиск в ширину для нахождения решения
94     solution_node = breadth_first_search(problem)
95
96     if solution_node != failure:
97         path = path_states(solution_node)
98         actions = path_actions(solution_node)
99         print(f"Путь до цели: {path}")
100        print(f"Действия: {actions}")
101    else:
102        print("Решение не найдено.")
103

```

```

~\ncfu\AIinPS_LR_2 main !7 ?2 python src/task_1.py
Путь до цели: ['Гудермес', 'Хасавюрт', 'Кизилюрт', 'Буйнакск', 'Леваши']
Действия: ['Хасавюрт', 'Кизилюрт', 'Буйнакск', 'Леваши']

```

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

```

3 # Матрица 10x10
4 grid = [
5     [1, 1, 0, 0, 0, 1, 0, 0, 1, 1],
6     [1, 1, 0, 0, 0, 1, 0, 1, 1, 0],
7     [0, 0, 0, 1, 1, 1, 1, 0, 1, 0],
8     [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
9     [1, 0, 0, 0, 1, 0, 1, 1, 0, 1],
10    [1, 0, 1, 1, 1, 0, 0, 0, 0, 1],
11    [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
12    [0, 0, 1, 0, 1, 0, 0, 1, 1, 0],
13    [1, 1, 0, 0, 0, 1, 0, 1, 0, 0],
14    [0, 0, 0, 0, 0, 0, 1, 0, 1, 1],
15 ]

```

```

18 class IslandCountingProblem:
19     def __init__(self, grid):
20         self.grid = grid
21         self.rows = len(grid)
22         self.cols = len(grid[0])
23
24     def actions(self, state):
25         """Возвращает возможные соседние клетки (включая диагонали)"""
26         row, col = state
27         directions = [
28             (-1, 0),
29             (1, 0),
30             (0, -1),
31             (0, 1), # горизонтальные и вертикальные
32             (-1, -1),
33             (-1, 1),
34             (1, -1),
35             (1, 1), # диагонали
36         ]
37         valid_actions = []
38         for dr, dc in directions:
39             new_row, new_col = row + dr, col + dc
40             if 0 ≤ new_row < self.rows and 0 ≤ new_col < self.cols:
41                 if self.grid[new_row][new_col] == 1: # Соседняя клетка земля
42                     valid_actions.append((new_row, new_col))
43         return valid_actions
44
45     def result(self, state, action):
46         """Возвращает новое состояние (соседнюю клетку)"""
47         return action

```

```

49     def is_goal(self, state):
50         """Для задачи подсчета островов не требуется использовать цель"""
51         return False
52
53     def action_cost(self, s, a, s1):
54         """Все переходы стоят одинаково"""
55         return 1
56
57     def h(self, node):
58         """Оценка не требуется, так как задача не ориентирована на поиск пути"""
59         return 0
60

```

```

62 def count_islands(grid):
63     problem = IslandCountingProblem(grid)
64     visited = set()
65     island_count = 0
66
67     # Проходим по всем клеткам
68     for row in range(problem.rows):
69         for col in range(problem.cols):
70             if grid[row][col] == 1 and (row, col) not in visited:
71                 # Запускаем BFS, чтобы посетить все клетки острова
72                 island_count += 1
73                 print(
74                     f"Новый остров найден, начинаем с клетки: ({row}, {col})"
75                 ) # Отладочный вывод
76                 # Запуск BFS для каждого нового острова
77                 bfs(problem, (row, col), visited)
78
79     return island_count

```

```

82 def bfs(problem, start, visited):
83     queue = deque([start])
84     visited.add(start)
85
86     while queue:
87         current = queue.popleft()
88         print(f"Посещаем клетку: {current}") # Отладочный вывод
89         for neighbor in problem.actions(current):
90             if neighbor not in visited:
91                 visited.add(neighbor)
92                 queue.append(neighbor)
93
94
95 if __name__ == "__main__":
96     # Подсчитываем количество островов
97     island_count = count_islands(grid)
98     print(f"Количество островов: {island_count}")

```

```

~/n/AIinPS_LR_2 main !7 ?2 python src/task_2.py
Новый остров найден, начинаем с клетки: (0, 0)
Новый остров найден, начинаем с клетки: (0, 5)
Новый остров найден, начинаем с клетки: (4, 6)
Новый остров найден, начинаем с клетки: (4, 9)
Количество островов: 4

```

7. Для задачи "Поиск кратчайшего пути в лабиринте" подготовить собственную схему лабиринта, а также определить начальную и конечную позиции в лабиринте. Для данных данных найти минимальный путь в лабиринте от начальной к конечной позиции.

```

1 from dependencies import Problem, breadth_first_search, failure, path_states
2
3 # Направления для движения (вверх, вниз, влево, вправо)
4 DIRECTIONS = [(-1, 0), (1, 0), (0, -1), (0, 1)]
5
6 class MazeProblem(Problem):
7     def __init__(self, maze, initial, goal, **kwargs):
8         self.maze = maze
9         self.rows = len(maze)
10        self.cols = len(maze[0])
11        super().__init__(initial=initial, goal=goal, **kwargs)
12
13    def actions(self, state):
14        """Возвращает доступные действия из текущего состояния"""
15        x, y = state
16        actions = []
17        for dx, dy in DIRECTIONS:
18            nx, ny = x + dx, y + dy
19            if (
20                0 ≤ nx < self.rows
21                and 0 ≤ ny < self.cols
22                and self.maze[nx][ny] == 1
23            ):
24                actions.append((nx, ny)) # Если клетка проходима (1)
25        return actions
26
27    def result(self, state, action):
28        """Возвращает новое состояние после применения действия"""
29        return action
30
31    def is_goal(self, state):
32        """Проверка, достигли ли мы цели"""
33        return state == self.goal
34
35    def action_cost(self, s, a, s1):
36        """Стоимость перехода всегда 1"""
37        return 1

```

```

40 def find_shortest_path(maze, start, goal):
41     # Создаем задачу с лабиринтом
42     problem = MazeProblem(maze, initial=start, goal=goal)
43     solution_node = breadth_first_search(problem)
44
45     if solution_node ≠ failure:
46         # Если решение найдено, возвращаем путь
47         path = path_states(solution_node)
48         return path
49     else:
50         # Если решения нет, возвращаем None
51         return None
52

```



```

54 if __name__ == "__main__":
55     # Пример лабиринта (0 – стена, 1 – путь)
56     maze = [
57         [1, 1, 0, 0, 0, 0, 0, 0],
58         [1, 1, 0, 1, 1, 0, 0, 0],
59         [0, 1, 0, 0, 1, 0, 1, 0],
60         [0, 1, 0, 1, 1, 0, 1, 0],
61         [0, 1, 0, 0, 0, 0, 1, 0],
62         [0, 1, 1, 1, 0, 1, 1, 0],
63         [0, 0, 0, 1, 1, 0, 0, 1],
64         [0, 0, 0, 1, 1, 1, 0, 1],
65     ]
66
67     start = (0, 0) # Начальная точка
68     goal = (7, 5) # Конечная точка
69
70     # Ищем кратчайший путь
71     path = find_shortest_path(maze, start, goal)
72
73     # Выводим результат
74     if path:
75         print(f"Кратчайший путь: {path}")
76         print(f"Длина пути: {len(path) - 1}")
77     else:
78         print("Путь не найден.")
79

```

```

начинается с PR.AI.001.) напишите программу на языке программирования
~/n/AIinPS_LR_2 main !7 ?2 python src/task_3.py
Кратчайший путь: [(0, 0), (1, 0), (1, 1), (2, 1), (3, 1), (4, 1), (5, 1), (5, 2), (5, 3), (6, 3), (7, 3), (7, 4), (7, 5)]
Длина пути: 12
минимальное расстояние между начальным и конечным пунктами.

```

8. Для построенного графа лабораторной работы 1 (имя файла начинается с PR.AI.001.) 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.

```

9 graph = {
10     "Гудермес": {"Шали": 35, "Курчалой": 19, "Хасавюрт": 45, "Буйнакск": 80},
11     "Шали": {"Гудермес": 35, "Курчалой": 19, "Буйнакск": 70, "Избербаш": 60},
12     "Курчалой": {"Шали": 19, "Гудермес": 19, "Ведено": 39, "Буйнакск": 55,
13                 "Тарумовка": 50
14     },
15     "Ведено": {"Курчалой": 39, "Хасавюрт": 90, "Буйнакск": 100},
16     "Хасавюрт": {"Гудермес": 45, "Ведено": 90, "Буйнакск": 60, "Леваши": 120,
17                 "Кизилюрт": 40, "Тарумовка": 75, "Сулак": 85
18     },
19     "Буйнакск": {"Хасавюрт": 60, "Леваши": 64, "Шали": 70, "Курчалой": 55,
20                 "Ведено": 100, "Избербаш": 85, "Каспийск": 90, "Кизилюрт": 50,
21                 "Тарумовка": 45, "Сулак": 78, "Гудермес": 80
22     },
23     "Леваши": {"Буйнакск": 64, "Хасавюрт": 120, "Избербаш": 75,
24                 "Каспийск": 55
25     },
26     "Избербаш": {"Шали": 60, "Буйнакск": 85, "Леваши": 75, "Каспийск": 30},
27     "Каспийск": {"Избербаш": 30, "Буйнакск": 90, "Леваши": 55, "Кизилюрт": 60},
28     "Кизилюрт": {"Хасавюрт": 40, "Буйнакск": 50, "Каспийск": 60},
29     "Тарумовка": {"Хасавюрт": 75, "Буйнакск": 45, "Курчалой": 50},
30     "Сулак": {"Буйнакск": 78, "Кизилюрт": 63, "Хасавюрт": 85},
31 }

```

```

60 # Создадим конкретную задачу, наследуя Problem
61 class SimpleGraphProblem(Problem):
62     def __init__(self, initial, goal, graph, **kwargs):
63         # Инициализация задачи с начальным и целевым состоянием, а также графом
64         super().__init__(initial, goal, **kwargs)
65         self.graph = graph
66
67     def actions(self, state):
68         """Возвращает список всех доступных переходов для данного состояния"""
69         # Возвращаем все соседние города
70         return self.graph.get(state, {})
71
72     def result(self, state, action):
73         """Возвращает новое состояние при применении действия"""
74         # Переход в указанный соседний город
75         return action
76
77     def action_cost(self, s, a, s1):
78         """Стоимость действия - в этом случае всегда 1"""
79         # Стоимость перехода (расстояние между городами)
80         return 1

```

```

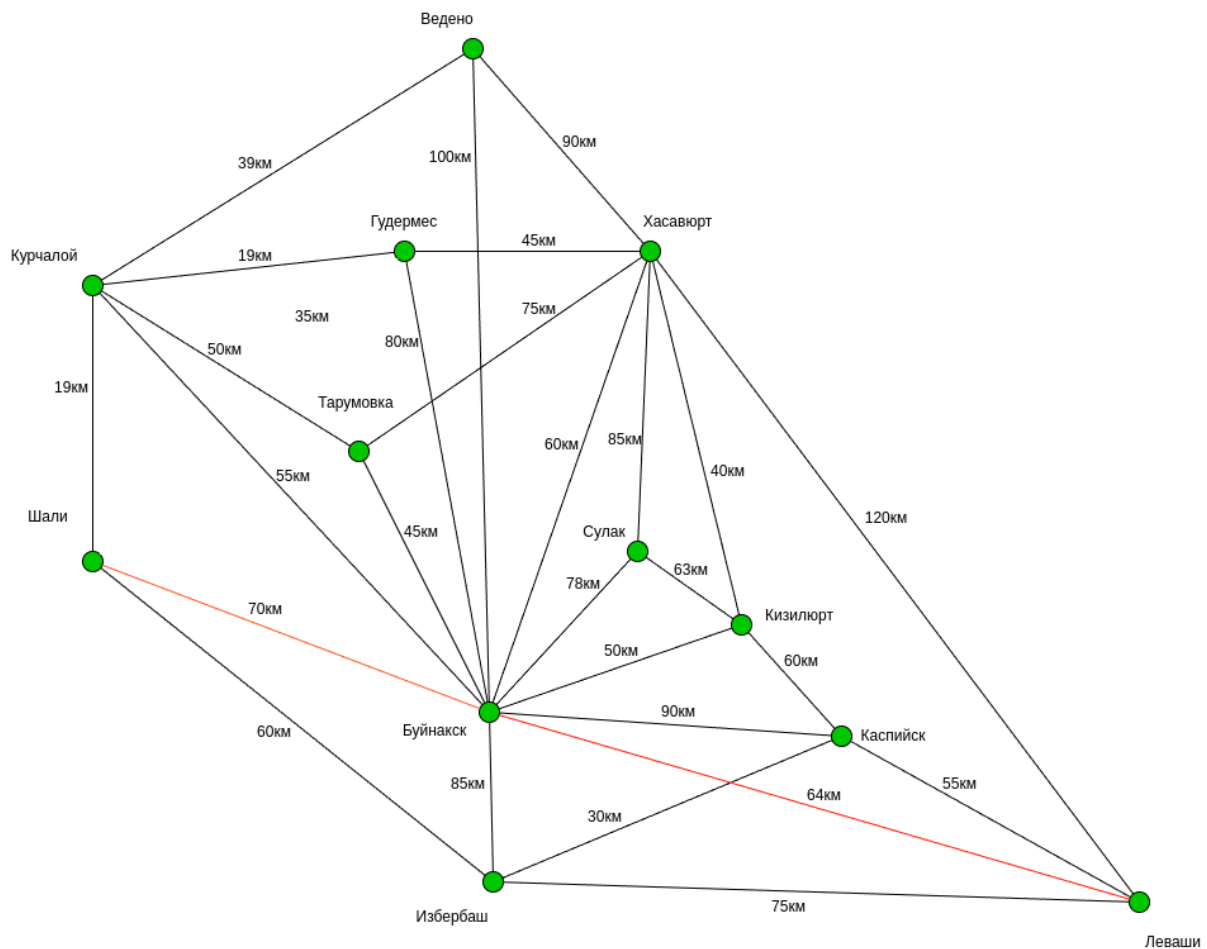
79 if __name__ == "__main__":
80     # Начальное и целевое состояние задачи
81     initial_state = "Шали"
82     goal_state = "Леваши"
83
84     # Создаем объект задачи
85     problem = SimpleGraphProblem(
86         initial=initial_state, goal=goal_state, graph=graph
87     )
88
89     # Используем поиск в ширину для нахождения решения
90     solution_node = breadth_first_search(problem)
91
92     if solution_node != failure:
93         path = path_states(solution_node)
94         actions = path_actions(solution_node)
95         print(f"Путь до цели: {path}")
96         print(f"Действия: {actions}")
97     else:
98         print("Решение не найдено.")

```

```

~/n/AIinPS_LR_2 main !7 ?2 python src/task_4.py
Путь до цели: ['Шали', 'Буйнакск', 'Леваши']
Действия: ['Буйнакск', 'Леваши']

```



9. Зафиксируйте сделанные изменения в репозитории.

```
~ /n/AIinPS_LR_2 main +19 !5 git add .
~ /ncfu/AIinPS_LR_2 main +19 git commit -m "added tasks"
check yam!.....(no files to check)Skipped
fix end of files.....Passed
trim trailing whitespace.....Passed
ruff.....Passed
ruff-format.....Passed
[main f02a44d] added tasks
13 files changed, 448 insertions(+), 17 deletions(-)
delete mode 100644 dependencies/node.py
create mode 100644 requirements.txt
rename {dependencies => src/dependencies}/__init__.py (100%)
rename {dependencies => src/dependencies}/breadth_first_search.py (100%)
rename {dependencies => src/dependencies}/enums.py (100%)
rename {dependencies => src/dependencies}/handlers.py (100%)
create mode 100644 src/dependencies/node.py
rename {dependencies => src/dependencies}/problem.py (100%)
rename {dependencies => src/dependencies}/queue.py (100%)
create mode 100644 src/task_1.py
create mode 100644 src/task_2.py
create mode 100644 src/task_3.py
create mode 100644 src/task_4.py
```

10. Добавьте отчет по лабораторной работе в формате PDF в папку doc репозитория. Зафиксируйте изменения.
11. Отправьте сделанные изменения на сервер GitHub.
12. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

### Контрольные вопросы

1. **Какой тип очереди используется в стратегии поиска в ширину?** Стратегия поиска в ширину использует очередь **FIFO (First In, First Out)**, то есть очередь, в которой элементы извлекаются в том порядке, в котором они были добавлены.
2. **Почему новые узлы в стратегии поиска в ширину добавляются в конец очереди?** Узлы добавляются в конец очереди, чтобы гарантировать, что они будут обработаны в порядке их появления на текущем уровне дерева или графа. Это позволяет искать все возможные решения по уровням, начиная с ближайших узлов к корню.

3. **Что происходит с узлами, которые дольше всего находятся в очереди в стратегии поиска в ширину?** Узлы, которые находятся в очереди дольше всего, будут обработаны позже, так как они добавляются в очередь первыми и извлекаются для расширения на более поздних шагах.

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 корня, так как они находятся на одном уровне и были добавлены в очередь поочередно.

5. **Почему важно расширять узлы с наименьшей глубиной в поиске в ширину?** Это важно для того, чтобы найти кратчайший путь к целевому состоянию. Если мы расширяем узлы по уровням, то как только мы достигнем целевого состояния, мы можем быть уверены, что нашли его на минимальном расстоянии от корня.

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 потомков у каждого узла), а  $d$  — глубина поиска. Алгоритм обрабатывает все узлы до уровня  $d$ .

7. **Каков основной фактор, определяющий пространственную сложность алгоритма поиска в ширину?** Пространственная сложность поиска в ширину зависит от количества узлов на текущем уровне дерева, то есть на уровне  $d$ . Она составляет  $O(b^d)$ , так как в очереди могут находиться все узлы на одном уровне.

8. **В каких случаях поиск в ширину считается полным?** Поиск в ширину считается полным, если существует решение в пределах заданной глубины, и алгоритм обязательно найдет его, если оно существует.

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

потребовать хранения множества узлов в памяти одновременно, особенно если коэффициент разветвления высок, что приводит к высокой потребности в памяти.

10. **В чем заключается оптимальность поиска в ширину?**

Поиск в ширину оптимален, потому что он гарантирует нахождение кратчайшего пути в невзвешенных графах или деревьях.

11. **Какую задачу решает функция `breadth_first_search`?**

Функция `breadth_first_search` решает задачу поиска кратчайшего пути от начального состояния к целевому состоянию в графе или дереве.

12. **Что представляет собой объект `problem`, который передается в функцию?** Объект `problem` представляет задачу поиска, содержащую информацию о начальном состоянии, целевом состоянии, правилах перехода и других характеристиках задачи.

13. **Для чего используется узел `Node(problem.initial)` в начале функции?** Узел `Node(problem.initial)` создается для представления начального состояния задачи и используется как исходная точка для поиска.

14. **Что произойдет, если начальное состояние задачи уже является целевым?** Если начальное состояние уже является целевым, то алгоритм немедленно завершится, не расширяя другие узлы.

15. **Какую структуру данных использует `frontier` и почему выбрана именно очередь FIFO?** Структура данных `frontier` — это очередь FIFO, потому что поиск в ширину работает по принципу «расширять сначала узлы, находящиеся ближе всего к корню» и требует последовательной обработки узлов по уровням.

16. **Какую роль выполняет множество `reached`?** Множество `reached` используется для хранения уже посещенных состояний, чтобы избежать повторной обработки одних и тех же состояний и заикливания.

17. Почему важно проверять, находится ли состояние в множестве **reached**? Это важно для предотвращения циклов и повторной обработки состояний, что может привести к неэффективному расходованию ресурсов.

18. Какую функцию выполняет цикл **while frontier**? Цикл **while frontier** выполняет извлечение узлов из очереди и их расширение до тех пор, пока очередь не пуста или не будет найдено целевое состояние.

19. Что происходит с узлом, который извлекается из очереди в строке **node = frontier.pop()**? Узел, извлеченный из очереди, будет расширен — его потомки добавляются в очередь для дальнейшей обработки.

20. Какова цель функции **expand(problem, node)**? Функция **expand** генерирует все дочерние узлы для текущего узла и добавляет их в очередь для дальнейшего расширения.

21. Как определяется, что состояние узла является целевым? Состояние узла является целевым, если оно совпадает с целевым состоянием задачи, что проверяется с помощью метода **problem.goal\_test(node.state)**.

22. Что происходит, если состояние узла не является целевым, но также не было ранее достигнуто? Если состояние узла еще не было достигнуто, то оно добавляется в множество **reached** и в очередь для дальнейшего расширения.

23. Почему дочерний узел добавляется в начало очереди с помощью **appendleft(child)**? Вопрос содержит ошибку — в поиске в ширину дочерние узлы добавляются в конец очереди, а не в начало. Для добавления в начало используют **deque** (если бы речь шла о поиске в глубину).

24. Что возвращает функция `breadth_first_search`, если решение не найдено? Если решение не найдено, функция возвращает `None` или аналогичный объект, указывающий на отсутствие решения.

25. Каково значение узла `failure` и когда он возвращается? Узел `failure` может быть возвращен, если поиск не находит целевого состояния после исчерпания всех возможных узлов для обработки.