

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:
Кожуховский Виктор Андреевич
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 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:

1. Изучил теоретический материал работы.
2. Создал общедоступный репозиторий на GitHub, в котором использована лицензия MIT и язык программирования Python.
3. Выполнил клонирование созданного репозитория.
4. Дополнил файл .gitignore необходимыми правилами для работы с IDE.
5. Организовал свой репозиторий в соответствии с моделью ветвления git-flow.
6. Создал проект в папке репозитория.
7. Проработал примеры лабораторной работы.
8. 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.

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

```

121 class ProblemIslands(Problem):
122     def __init__(self, grid):
123         initial = self.find_initial_state(grid)
124         goal = None
125         super().__init__(initial=initial, goal=goal, grid=grid)
126
127     def find_initial_state(self, grid):
128         return tuple(tuple(row) for row in grid)
129
130     def actions(self, state):
131         return [(dx, dy) for dx, dy in ((1, 0), (-1, 0), (0, 1), (0, -1),
132                                         (1, 1), (1, -1), (-1, 1), (-1, -1))]
133
134     def result(self, state, action):
135         grid = [list(row) for row in state]
136         for i in range(len(grid)):
137             for j in range(len(grid[0])):
138                 if grid[i][j] == 1:
139                     self.explore_island(grid, i, j)
140         return tuple(tuple(row) for row in grid)
141
142     def explore_island(self, grid, i, j):
143         queue = deque([(i, j)])
144         grid[i][j] = 0 # Посещенное
145         while queue:
146             x, y = queue.popleft()
147             for dx, dy in ((1, 0), (-1, 0), (0, 1), (0, -1),
148                           (1, 1), (1, -1), (-1, 1), (-1, -1)):
149                 nx, ny = x + dx, y + dy
150                 if 0 <= nx < len(grid) and 0 <= ny < len(grid[0]) and \
151                     grid[nx][ny] == 1:
152                     grid[nx][ny] = 0 # Посещенное
153                     queue.append((nx, ny))
154
155     def is_goal(self, state):
156         return False
157
158     def count_islands_with_bfs(problem):
159         count = 0
160         initial_state = problem.initial
161         grid = [list(row) for row in initial_state]
162         rows, cols = len(grid), len(grid[0])
163
164         for i in range(rows):
165             for j in range(cols):
166                 if grid[i][j] == 1:
167                     count += 1
168                     problem.explore_island(grid, i, j)
169         return count
170
171 if __name__ == "__main__":
172     grid = [
173         [1, 1, 0, 0, 0],
174         [0, 1, 0, 0, 1],
175         [1, 0, 0, 1, 1],
176         [0, 0, 0, 0, 0],
177         [1, 0, 1, 0, 1]
178     ]
179
180     problem = ProblemIslands(grid)
181     print(count_islands_with_bfs(problem))

```

Рисунок 1. Решение задачи по поиску островов

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.

```

class ProblemLabyrinth(Problem):
    def __init__(self, grid, initial, goal):
        super().__init__(initial=initial, goal=goal, grid=grid)

    def actions(self, state):
        i, j = state
        possible_actions = []
        directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

        for di, dj in directions:
            ni, nj = i + di, j + dj
            if (
                0 <= ni < len(self.grid)
                and 0 <= nj < len(self.grid[0])
                and self.grid[ni][nj] == 1
            ):
                possible_actions.append((ni, nj))

```

```

        return possible_actions

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

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

def breadth_first_search(problem):
    node = Node(problem.initial)
    if problem.is_goal(problem.initial):
        return node

    frontier = FIFOQueue([node])
    reached = {problem.initial}

    while frontier:
        node = frontier.pop()

        for child in node.expand(problem):
            s = child.state

            if problem.is_goal(s):
                return child

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

    return Node.failure

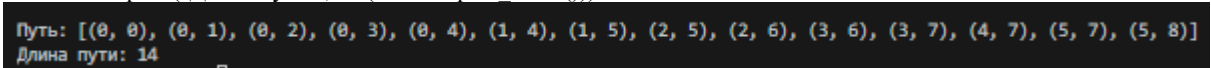
if __name__ == "__main__":
    grid = [
        [1, 1, 1, 1, 1, 0, 0, 1, 1, 1],
        [0, 1, 1, 1, 1, 1, 0, 1, 0, 1],
        [0, 0, 1, 0, 1, 1, 1, 0, 0, 1],
        [1, 0, 1, 1, 1, 0, 1, 1, 0, 1],
        [0, 0, 0, 1, 0, 0, 0, 1, 0, 1],
        [1, 0, 1, 1, 1, 0, 0, 1, 1, 0],
        [0, 0, 0, 0, 1, 0, 0, 1, 0, 1],
        [0, 1, 1, 1, 1, 1, 1, 0, 0, 0],
        [1, 1, 1, 1, 1, 0, 0, 1, 1, 1],
        [0, 0, 1, 0, 0, 1, 1, 0, 0, 1],
    ]

    initial = (0, 0)
    goal = (5, 8)

    problem = ProblemLabyrinth(grid, initial, goal)
    solution = breadth_first_search(problem)

    if solution is Node.failure:
        print("Решение не найдено.")
    else:
        print("Решение найдено:")
        print("Путь:", solution.path_states())
    print("Длина пути:", len(solution.path_states()))

```



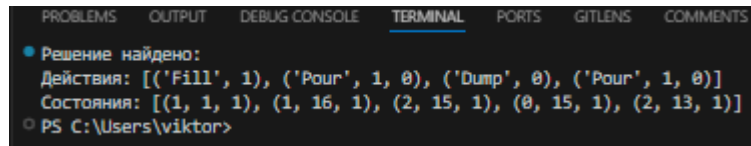
```

Путь: [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (1, 5), (2, 5), (2, 6), (3, 6), (3, 7), (4, 7), (5, 7), (5, 8)]
Длина пути: 14

```

Рисунок 2. Решение задачи по поиску кратчайшего пути в лабиринте

Для задачи о льющихся кувшинах получить заданный объем воды в одном из кувшинов.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS COMMENTS
• Решение найдено:
Действия: [("Fill", 1), ("Pour", 1, 0), ("Dump", 0), ("Pour", 1, 0)]
Состояния: [(1, 1, 1), (1, 16, 1), (2, 15, 1), (0, 15, 1), (2, 13, 1)]
PS C:\Users\viktor>
```

Рисунок 3. Решение задачи о льющихся кувшинах

10. Для построенного графа лабораторной работы 1 (имя файла начинается с PR.AI.001.) напишите программу на языке программирования Python, которая с использованием алгоритма поиска в ширину находит минимальное расстояние между начальным и конечным пунктами.

```
class CityProblem:
    def __init__(self, cities, distances, initial, goal):
        self.cities = cities
        self.distances = distances
        self.initial = initial
        self.goal = goal

    def actions(self, state):
        return [target for target in self.cities if (state, target) in self.distances]

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

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

def breadth_first_search(problem):
    node = problem.initial
    if problem.is_goal(node):
        return [node]

    frontier = deque([node])
    came_from = {node: None}

    while frontier:
        current = frontier.popleft()

        for action in problem.actions(current):
            if action not in came_from:
                came_from[action] = current
                if problem.is_goal(action):
                    return reconstruct_path(came_from, action)
                frontier.append(action)

    return None

def reconstruct_path(came_from, current):
    path = []
    while current is not None:
        path.append(current)
        current = came_from[current]
```

```

path.reverse()
return path

if __name__ == "__main__":
    with open("elem.json", "r", encoding="utf-8") as file:
        data = json.load(file)

    selected_ids = {"8", "9", "2", "15", "6", "1", "3", "7", "13", "18"}

    cities = {}
    distances = {}

    for item in data:
        if "label" in item["data"]:
            if item["data"]["id"] in selected_ids:
                cities[item["data"]["id"]] = item["data"]["label"]
            elif "source" in item["data"]:
                source = item["data"]["source"]
                target = item["data"]["target"]
                if source in selected_ids and target in selected_ids:
                    weight = item["data"]["weight"]
                    distances[(source, target)] = weight
                    distances[(target, source)] = weight

    start_city = "8"
    goal_city = "15"

    problem = CityProblem(cities, distances, start_city, goal_city)
    solution = breadth_first_search(problem)

    if solution is None:
        print("Решение не найдено.")
    else:
        print("Кратчайший путь:")
        print(" -> ".join(cities[city] for city in solution))

```

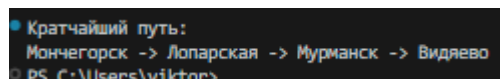


Рисунок 3. Код программы решения задания и его выполнение

11. Зафиксировал сделанные изменения в репозитории.
12. Выполнил слияние ветки для разработки с веткой master/main.
13. Отправил сделанные изменения на сервер GitHub.

Ссылка: https://github.com/Viktorkozh/AI_2

Контрольные вопросы:

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

В стратегии поиска в ширину используется очередь типа "первым пришел - первым ушел" (FIFO).

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

Новые узлы в стратегии поиска в ширину добавляются в конец очереди, чтобы обеспечить порядок обработки узлов, начиная с наименее глубоких.

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

Узлы, которые дольше всего находятся в очереди, находятся впереди и обрабатываются в первую очередь.

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

Следующим для расширения будет узел, находящийся на том же уровне глубины, что и корневой узел, в данном случае узел C.

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

Важно расширять узлы с наименьшей глубиной в поиске в ширину, чтобы гарантировать нахождение решения на минимальной глубине.

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

Временная сложность алгоритма поиска в ширину зависит от коэффициента разветвления (b) и глубины наименее дорогого решения (d) по формуле $O(b^{(d+1)})$.

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

Основной фактор, определяющий пространственную сложность алгоритма поиска в ширину, — это максимальное количество узлов, которые необходимо хранить в памяти одновременно.

8. В каких случаях поиск в ширину считается полным?

Поиск в ширину считается полным, если коэффициент ветвления (b) конечен.

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

Поиск в ширину может быть неэффективен с точки зрения памяти из-за необходимости хранения всех узлов в памяти одновременно, что может привести к огромному количеству узлов.

10. В чем заключается оптимальность поиска в ширину?

Оптимальность поиска в ширину заключается в способности находить решение с наименьшей стоимостью среди всех возможных.

11. Какую задачу решает функция `breadth_first_search`?

Функция `breadth_first_search` решает задачу поиска решения для заданной задачи.

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

Объект `problem`, который передается в функцию, описывает задачу поиска.

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

Узел `Node(problem.initial)` в начале функции используется для создания начального узла поиска на основе начального состояния задачи.

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

Если начальное состояние задачи уже является целевым, то функция возвращает начальный узел как решение.

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

`Frontier` использует очередь FIFO, чтобы хранить узлы, которые нужно расширить, обеспечивая порядок обработки.

16. Какую роль выполняет множество `reached` ?

Множество `reached` выполняет роль отслеживания посещенных состояний, чтобы избежать повторного посещения одного и того же состояния.

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

Важно проверять, находится ли состояние в множестве `reached`, чтобы избежать повторного посещения одного и того же состояния и тем самым предотвратить заикливание.

18. Какую функцию выполняет цикл `while frontier` ?

Цикл `while frontier` выполняет обработку узлов, пока в очереди есть узлы для обработки.

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

Узел, который извлекается из очереди в строке `node = frontier.pop()`, становится текущим узлом для расширения.

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

Цель функции `expand(problem, node)` — генерировать дочерние узлы для текущего узла.

21. Как определяется, что состояние узла является целевым?

Состояние узла является целевым, если оно удовлетворяет условию, определенному в методе `is_goal` объекта `problem`.

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

Если состояние узла не является целевым, но также не было ранее достигнуто, оно добавляется в множество `reached` и дочерний узел добавляется в очередь `frontier`.

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

Дочерний узел добавляется в начало очереди с помощью `appendleft(child)` для обеспечения порядка обработки узлов.

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

Если решение не найдено, функция `breadth_first_search` возвращает специальный узел `failure`.

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

Значение узла `failure` — это индикатор того, что решение не было найдено, и он возвращается, когда все узлы были обработаны, но целевое состояние не было достигнуто.

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