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 №3

Дисциплины «Искусственный интеллект в профессиональной сфере»

Выполнил:
Говоров Егор Юрьевич
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных
систем», очная форма обучения

_____

(подпись)

Руководитель практики:

Воронкин Р. А., доцент департамента
цифровых и робототехнических
систем и электроники института
перспективной инженерии

_____

(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: исследование поиска в глубину.

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

Ссылка на репозиторий: https://github.com/Artorias1469/Artificial_Intelligence_laboratory_work_3.git

Ход работы:

Задача 1. Проработка примера из лабораторной работы

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
# Рассмотрим реализацию алгоритма поиска в глубину на практике, в программном коде.


class Node:  1 usage  new *
    def __init__(self, state):  new *
        self.state = state


def is_cycle(node):  1 usage  new *
    # Логика проверки циклов
    pass


def expand(problem, node):  1 usage  new *
    # Логика расширения узлов
    pass


failure = None  # Определите значение failure, если это нужно

```

Рисунок 1. Выполнение примера

Задача 2. Поиск самого длинного пути в матрице

Дана матрица символов размером M×N. Необходимо найти длину самого длинного пути в матрице, начиная с заданного символа. Каждый следующий символ в пути должен алфавитно следовать за предыдущим без пропусков. Разработать функцию поиска самого длинного пути в матрице символов, начиная с заданного символа. Символы в пути должны следовать в алфавитном порядке и быть последовательными. Поиск возможен во всех восьми направлениях.

```
["P", "Q", "R", "S", "T", "U", "V"],
["O", "N", "M", "L", "K", "J", "I"],
["A", "B", "C", "D", "E", "F", "G"],
["H", "G", "F", "E", "D", "C", "B"],
["I", "J", "K", "L", "M", "N", "O"],
["V", "U", "T", "S", "R", "Q", "P"],
["W", "X", "Y", "Z", "A", "B", "C"]
```

Рисунок 2. Матрица для выполнения задания №1

```
D:\Users\Admin\anaconda3\envs\Artificial_Intelligence_laboratory_work_2\python.exe
Длина самого длинного пути: 23

Process finished with exit code 0
```

Рисунок 3. Итоги выполнения программы №1

Задача 3. Генерирование списка возможных из матрицы символов

Вам дана матрица символов размером M × N. Ваша задача — найти и вывести список всех возможных слов, которые могут быть сформированы из последовательности соседних символов в этой матрице. При этом слово может формироваться во всех восьми возможных направлениях (север, юг, восток, запад, северо-восток, северо-запад, юго-восток, юго-запад), и каждая клетка может быть использована в слове только один раз.

```
['К', 'О', 'Т'],
['А', 'Р', 'Ь'],
['С', 'Л', 'О']
```

Рисунок 4. Матрица для выполнения задания №2

```
dictionary = ['КОТ','КОЛ','ОРК','СОК','ТОРС','ТОКАРЬ','СЛОТ','СОЛЬ','ЛОСЬ','ЛОТ','КОРОЛЬ','СОРТ']
```

Рисунок 5. Набор возможных слов

```
D:\Users\Admin\anaconda3\envs\Artificial_Intelligence_laboratory_work_2\python.exe
Найденные слова: {'ОРК', 'КОТ', 'КОРОЛЬ', 'ТОКАРЬ', 'ТОРС'}

Process finished with exit code 0
```

Рисунок 6. Итог выполнения программы №2

Задача 4. Поиск кратчайшего расстояния между городами с помощью метода поиска

Напишите программу на языке программирования Python, которая с помощью алгоритма поиска в глубину находит минимальное расстояние между начальным и конечным пунктами. Сравните найденное решение с решением, полученным вручную.

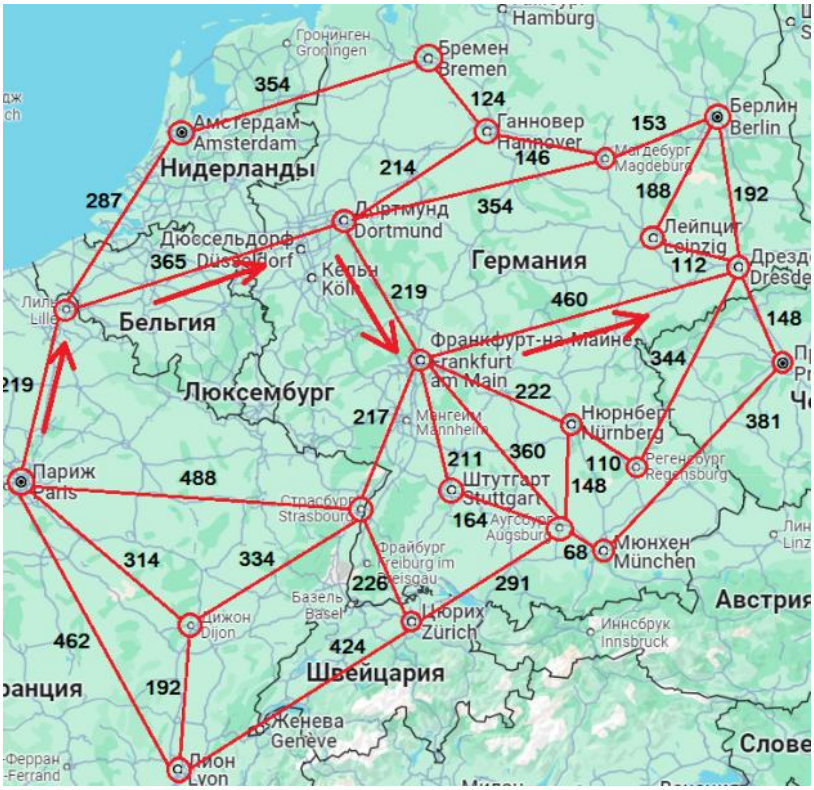

Рисунок 7. Граф расстояний между городами

```
1: [(2, 219), (3, 488), (4, 314), (5, 462)],
2: [(1, 219), (6, 287), (7, 365)],
3: [(1, 488), (4, 334), (8, 226), (9, 217)],
4: [(1, 314), (3, 334), (5, 192)],
5: [(1, 462), (4, 192), (8, 424)],
6: [(2, 287), (10, 354)],
7: [(2, 365), (11, 214), (12, 354), (9, 219)],
8: [(3, 226), (5, 424), (14, 291)],
9: [(3, 217), (7, 219), (15, 211), (16, 222), (20, 460), (14, 360)],
10: [(6, 354), (11, 124)],
11: [(7, 214), (10, 124), (12, 146)],
12: [(7, 354), (11, 146), (13, 153)],
13: [(12, 153), (19, 188), (20, 192)],
14: [(8, 291), (9, 360), (15, 164), (16, 148), (17, 68)],
15: [(9, 211), (14, 164)],
16: [(9, 222), (14, 148), (17, 110)],
17: [(14, 68), (16, 110), (18, 381)],
18: [(17, 381), (20, 148)],
19: [(13, 188), (20, 112)],
20: [(9, 460), (13, 192), (18, 148), (19, 112)],
21: [(16, 344)],
```

Рисунок 8. Список расстояний

```
Кратчайший путь: [2, 7, 9, 16, 17]
Длина пути: 916
```

Рисунок 9. Расстояние пол

Ответы на контрольные вопросы:

1. В чем ключевое отличие поиска в глубину от поиска в ширину?

Ключевое отличие состоит в стратегии обхода. Поиск в глубину (DFS) исследует одну ветвь дерева/графа до самого глубокого уровня, прежде чем перейти к следующей ветви. Поиск в ширину (BFS) исследует все узлы на одном уровне глубины перед переходом к следующему уровню.

2. Какие четыре критерия качества поиска обсуждаются в тексте для оценки алгоритмов?

- Полнота — гарантирует ли алгоритм нахождение решения, если оно существует.
- Оптимальность — находит ли алгоритм лучшее (минимальное по стоимости) решение.
- Временная сложность — сколько времени требуется для нахождения решения.
- Пространственная сложность — сколько памяти использует алгоритм.

3. Что происходит при расширении узла в поиске в глубину?

При расширении узла генерируются его дочерние узлы. Это включает создание новых узлов на основе соседей текущего узла, которые затем добавляются в стек или передаются в рекурсию.

4. Почему поиск в глубину использует очередь типа "последним пришел — первым ушел" (LIFO)?

Очередь LIFO (стек) обеспечивает приоритетное исследование недавно добавленных узлов, что позволяет алгоритму "углубляться" в ветви графа или дерева.

5. Как поиск в глубину справляется с удалением узлов из памяти, и почему это преимущество перед поискм в ширину?

Поиск в глубину удаляет узлы из памяти, как только они обработаны и нет необходимости возвращаться к ним. Это снижает объем памяти по сравнению с поиском в ширину, который должен хранить все узлы на текущем уровне.

6. Какие узлы остаются в памяти после того, как достигнута максимальная глубина дерева?

В памяти остаются узлы текущего пути от корня до текущего узла и узлы, которые еще не были исследованы.

7. В каких случаях поиск в глубину может "застрять" и не найти решение?

- Если граф или дерево бесконечно глубокие.
- Если существует цикл, и алгоритм не имеет проверки на циклы.

8. Как временная сложность поиска в глубину зависит от максимальной глубины дерева?

Временная сложность DFS составляет $O(b^m)$, где b — фактор ветвления, а m — максимальная глубина дерева. Она растет экспоненциально с глубиной дерева.

9. Почему поиск в глубину не гарантирует нахождение оптимального решения?

DFS не рассматривает все пути одновременно, поэтому может найти не самый короткий путь, если решение обнаружено до исследования более выгодного варианта.

10. В каких ситуациях предпочтительно использовать поиск в глубину, несмотря на его недостатки?

- Когда пространство поиска ограничено и важно минимизировать потребление памяти.
- Если известна приблизительная глубина решения.
- Когда нужно найти любое решение быстро, а не обязательно оптимальное.

11. Что делает функция depth_first_recursive_search, и какие параметры она принимает?

Функция выполняет рекурсивный поиск в глубину для нахождения решения. Она принимает:

- problem — задачу, содержащую начальный узел и цель.
- graph — граф для обхода.
- node (опционально) — текущий узел.

Дополнительно может передаваться текущая минимальная длина пути и путь.

12. каком случае функция возвращает узел как решение задачи?

Когда состояние узла совпадает с целевым состоянием задачи (problem.is_goal(node.state)).

13. Почему важна проверка на циклы в алгоритме рекурсивного поиска в глубину?

Проверка на циклы предотвращает бесконечный возврат к ранее посещенным узлам, особенно в графах с циклическими структурами.

14. Что возвращает функция при обнаружении цикла?

Она возвращает None (или failure), указывая, что цикл был обнаружен и продолжение поиска по этому пути невозможно.

15. Как функция обрабатывает дочерние узлы текущего узла?

Функция генерирует дочерние узлы через expand и рекурсивно вызывает саму себя для каждого из них.

16. Какой механизм используется для обхода дерева поиска в этой реализации?

Используется рекурсия для перехода между узлами, а стек вызовов автоматически сохраняет текущий путь.

17. Что произойдет, если не будет найдено решение в ходе рекурсии?

Функция вернет failure, что указывает на отсутствие пути к цели.

18. Почему функция рекурсивно вызывает саму себя внутри цикла?

Это позволяет исследовать все ветви графа/дерева, начиная с текущего узла.

19. Как функция expand(problem, node) взаимодействует с текущим узлом?

Она генерирует список дочерних узлов текущего узла на основе графа и его соседей.

20. Какова роль функции is_cycle(node) в этом алгоритме?

Она проверяет, встречался ли текущий узел ранее в пути, предотвращая зацикливание.

21. Почему проверка if result в рекурсивном вызове важна для корректной работы алгоритма?

Эта проверка определяет, было ли найдено решение по данному пути, и завершает дальнейший поиск, если оно найдено.

22. В каких ситуациях алгоритм может вернуть failure?

- Если узел не может быть расширен (нет дочерних узлов).
- Если все пути исследованы, но цель не достигнута.

23. Как рекурсивная реализация отличается от итеративного поиска в глубину?

В рекурсивной реализации используется стек вызовов, управляемый автоматически, в то время как в итеративной используется явный стек для хранения состояния узлов.

24. Какие потенциальные проблемы могут возникнуть при использовании этого алгоритма для поиска в бесконечных деревьях?

- Бесконечная рекурсия при отсутствии проверки на глубину или циклы.
- Переполнение стека вызовов, что приведет к ошибке сегментации (stack overflow).

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