

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»

Вариант _____

Выполнила:
Маньшина Дарья Алексеевна
2 курс, группа ИТС-б-о-22-1,
11.03.02 «Инфокоммуникационные
технологии и системы связи»,
направленность (профиль)
«Инфокоммуникационные системы и
сети», 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 глубину.

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

Ход работы:

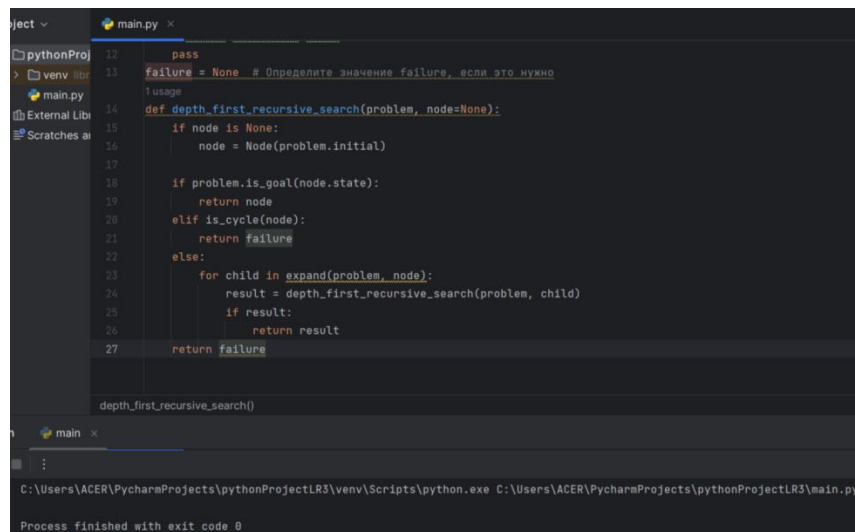
Ссылка на репозиторий: <https://github.com/Dash-AI/3lr3.git>

Необходимо скопировать репозиторий

```
C:\Users\ACER>git clone https://github.com/Dash-AI/3lr3.git
Cloning into '3lr3'...
remote: Enumerating objects: 11, done.
remote: Counting objects: 100% (11/11), done.
remote: Compressing objects: 100% (10/10), done.
Receiving objects: 100% (11/11), 10.08 KiB | 135.00 KiB/s, done.rom 0)
Resolving deltas: 100% (2/2), done.
```

Рисунок 1 – Клон репозитория

Рассмотрим пример представленный в методическом указании



```
main.py x
pythonProj
> venv lib
main.py
External Lib
Scratches a

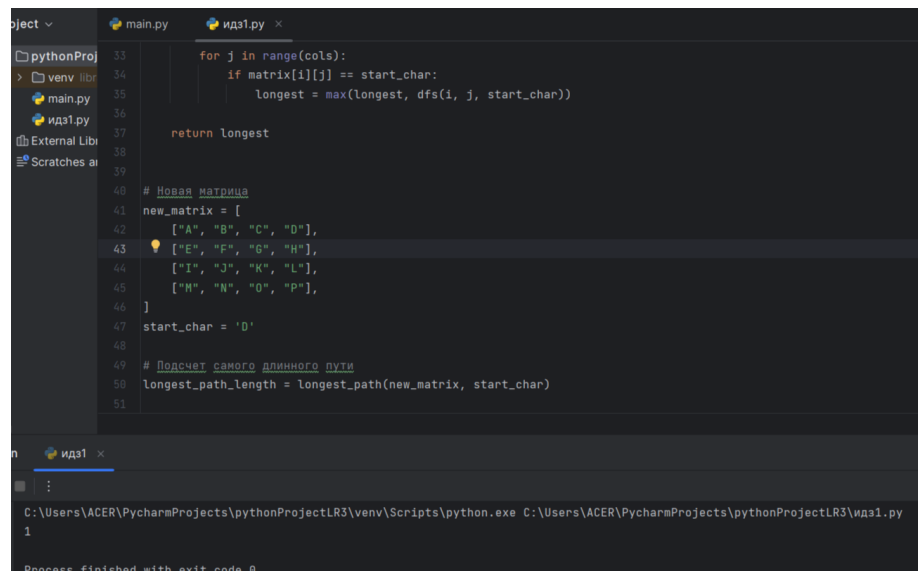
12 pass
13 failure = None # Определите значение failure, если это нужно
14 usage
15 def depth_first_recursive_search(problem, node=None):
16     if node is None:
17         node = Node(problem.initial)
18     if problem.is_goal(node.state):
19         return node
20     elif is_cycle(node):
21         return failure
22     else:
23         for child in expand(problem, node):
24             result = depth_first_recursive_search(problem, child)
25             if result:
26                 return result
27     return failure

depth_first_recursive_search()

main x
C:\Users\ACER\PycharmProjects\pythonProjectLR3\venv\Scripts\python.exe C:\Users\ACER\PycharmProjects\pythonProjectLR3\main.py
Process finished with exit code 0
```

Рисунок 1 – Пример №1

Индивидуальное задание №1: для задачи «Поиск самого длинного пути в матрице» подготовить собственную матрицу, для которой с помощью, разработанной в предыдущем пункте программы, подсчитать самый длинный путь.



```
33     for j in range(cols):
34         if matrix[i][j] == start_char:
35             longest = max(longest, dfs(i, j, start_char))
36
37     return longest
38
39
40 # Новая матрица
41 new_matrix = [
42     ["A", "B", "C", "D"],
43     ["E", "F", "G", "H"],
44     ["I", "J", "K", "L"],
45     ["M", "N", "O", "P"],
46 ]
47 start_char = 'D'
48
49 # Подсчет самого длинного пути
50 longest_path_length = longest_path(new_matrix, start_char)
51
```


Run ID1 x

C:\Users\ACER\PycharmProjects\pythonProjectLR3\venv\Scripts\python.exe C:\Users\ACER\PycharmProjects\pythonProjectLR3\идз1.py

Process finished with exit code 0

Рисунок 3 – Выполнение ИДЗ №1

Индивидуальное задание №2: для задачи «Генерирование списка возможных слов из матрицы символов» подготовить собственную матрицу для генерирования списка возможных слов с помощью разработанной программы.



```
33
34     return found_words
35 # Исходные данные
36 board = [
37     ['O', 'A', 'E'],
38     ['I', 'H', 'T'],
39     ['P', 'C', 'L']
40 ]
41 dictionary = ['КОЛ', 'САЛ', 'ТИНА', 'РИС', 'ЛЕС', 'НО', 'ТЛЕН', 'РИН', 'СТО']
42 # Вызов функции find_words
43 result = find_words(board, dictionary)
44 print(result) # Вывод найденных слов
```

Run ИД32 x

C:\Users\ACER\PycharmProjects\pythonProjectLR3\venv\Scripts\python.exe C:\Users\ACER\PycharmProjects\pythonProjectLR3\ИД32.py

{'РИС', 'РИН', 'НО'}

Process finished with exit code 0

Рисунок 4 – Выполнение ИДЗ №2

Индивидуальное задание №3: с помощью алгоритма поиска в глубину находит минимальное расстояние между начальным и конечным пунктами.

```

1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3 #С помощью алгоритма поиска в глубину находим минимальное расстояние между начальным и конечным пунктами
4 import itertools
5
6 def dfs(start, end, visited, current_distance, min_distance, distance_matrix):
7     """Поиск в глубину для нахождения минимального расстояния."""
8     if start == end:
9         return min(current_distance, min_distance)
10    for next_city in range(len(distance_matrix)):
11        if not visited[next_city]:
12            visited[next_city] = True
13            current_distance += distance_matrix[start][next_city]
14            min_distance = dfs(next_city, end, visited, current_distance, min_distance, distance_matrix)
15            current_distance -= distance_matrix[start][next_city]
16            visited[next_city] = False # Backtrack
17    return min_distance
18
19 usage =
20
21 dfs()

```

Run idz3

C:\Users\ACER\PycharmProjects\pythonProjectLR3\venv\Scripts\python.exe C:\Users\ACER\PycharmProjects\pythonProjectLR3\idz3.py

Минимальное расстояние с помощью DFS от пункта 0 до 9: 455

Process finished with exit code 0

Рисунок 5 – Выполнение ИДЗ №3

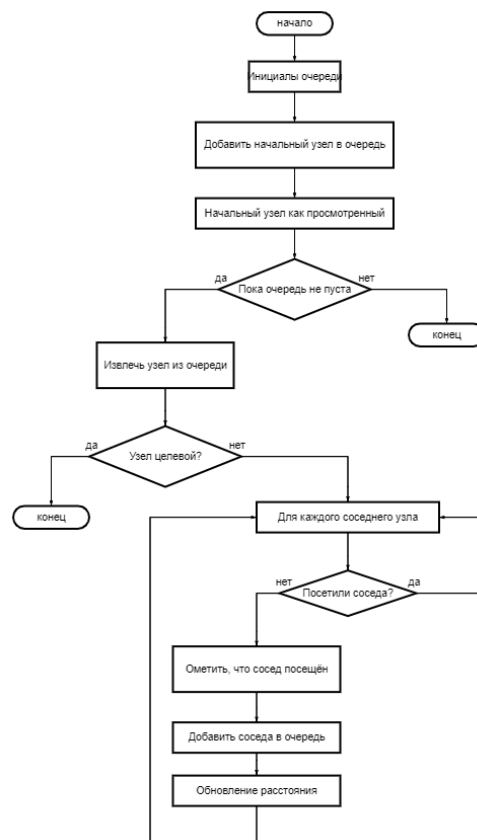


Рисунок 6 – Блок-схема для индивидуального задания №3

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

Контрольные вопросы:

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

Ключевое отличие заключается в порядке обхода узлов. Поиск в глубину (DFS) сначала исследует как можно глубже каждую ветвь, прежде чем вернуться назад, тогда как поиск в ширину (BFS) исследует все соседние узлы текущего уровня перед тем, как перейти на следующий уровень.

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

Критерии качества поиска могут включать: корректность, полноту, оптимальность и эффективность (временная и пространственная сложность).

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

При расширении узла в DFS исследуются все его дочерние узлы, при этом они добавляются в стек для дальнейшего обследования.

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

LIFO-структура стека позволяет DFS сначала вернуться к последнему исследованному узлу, что соответствует подходу исследования глубины.

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

DFS может освобождать память, так как не хранит все узлы на одном уровне одновременно; вместо этого он хранит только текущий путь. Это значительно снижает потребление памяти по сравнению с BFS.

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

Узлы, находящиеся на текущем пути к листьям, остаются в памяти, однако все узлы выше по этому пути могут быть удалены.

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

DFS может застрять в бесконечных циклах или если дерево имеет очень глубокие ветви, не приводящие к решению.

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

Временная сложность DFS составляет $O(b^d)$, где b — число ветвлений, а d — максимальная глубина. При увеличении глубины d время выполнения может экспоненциально увеличиваться.

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

Поскольку DFS не рассматривает все возможные пути до решения и возвращается к предыдущим узлам, он может не найти кратчайший путь к цели.

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

DFS предпочтителен при необходимости поиска в ограниченных пространствах, при серьезных ограничениях по памяти или когда требуется найти одно из возможных решений, а не оптимальное.

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

Эта функция выполняет рекурсивный поиск в глубину, принимая текущий узел и, возможно, другие параметры, такие как цель поиска или используемая структура данных для посещенных узлов.

12. Какую задачу решает проверка `ifnodeisNone`?

Эта проверка используется для определения, достигнут ли конец дерева или если текущий узел не существует. Это важно для предотвращения ошибок обращения к несуществующим узлам.

13. В каком случае функция возвращает узел как решение задачи?

Функция возвращает узел как решение, когда найден узел, соответствующий условию цели.

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

Проверка на циклы предотвращает заикливание алгоритма, которое может привести к бесконечной рекурсии и истощению памяти.

15. Что возвращает функция при обнаружении цикла?

Обычно такая функция возвращает индикатор или значение, указывающее на наличие цикла, чтобы прекратить дальнейшие действия.

16. Как функция обрабатывает дочерние узлы текущего узла?

Функция рекурсивно вызывает саму себя для каждого дочернего узла, передавая ему необходимые параметры.

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

Используется рекурсивный механизм, который вызывает функцию для текущего узла и движется вниз по иерархии дерева.

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

В этом случае функция может вернуться к предыдущему узлу и продолжить поиск в других ветвях или вернуть значение, указывающее на отсутствие решения.

19. Почему функция рекурсивно вызывает саму себя внутри цикла?

Это делается для исследования каждого дочернего узла и дальнейшего населения стека вызовов по мере движения вниз по дереву.

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

Функция `'expand'` генерирует и возвращает возможные дочерние узлы (или состояния) текущего узла, основываясь на заданной проблеме.

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

Функция `'is_cycle'` проверяет, является ли текущий узел частью заикленной последовательности, предотвращая заикливание алгоритма.

22. Почему проверка `ifresult` в рекурсивном вызове важна для корректной работы алгоритма?

Если `'result'` возвращает решение, это позволяет алгоритму сразу завершить выполнение и вернуть результат, избегая ненужных вычислений.

23. В каких ситуациях алгоритм может вернуть `failure`?

Алгоритм возвращает `'failure'`, если не нашел ни одного решения после проверки всех возможных узлов.

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

Рекурсивная реализация использует стек вызовов для хранения состояния, тогда как итеративная реализация явно управляет стеком в коде.

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

Основная проблема заключается в возможности бесконечной рекурсии и переполнения стека, что приводит к краху программы, поскольку алгоритм будет продолжать обход без нахождения решения.