

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

Искусственный интеллект в профессиональной сфере
Отчет по лабораторной работе №2
Исследование поиска в ширину

Выполнил:
Тихоненко Борис Витальевич
3 курс, группа ИТС-б-з-22-1,
11.03.02
«Инфокоммуникационные
технологии и системы связи»,
заочная форма обучения

(подпись)

Проверил: доцент департамента
цифровых робототехнических систем и
электроники
Воронкин Роман Александрович

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

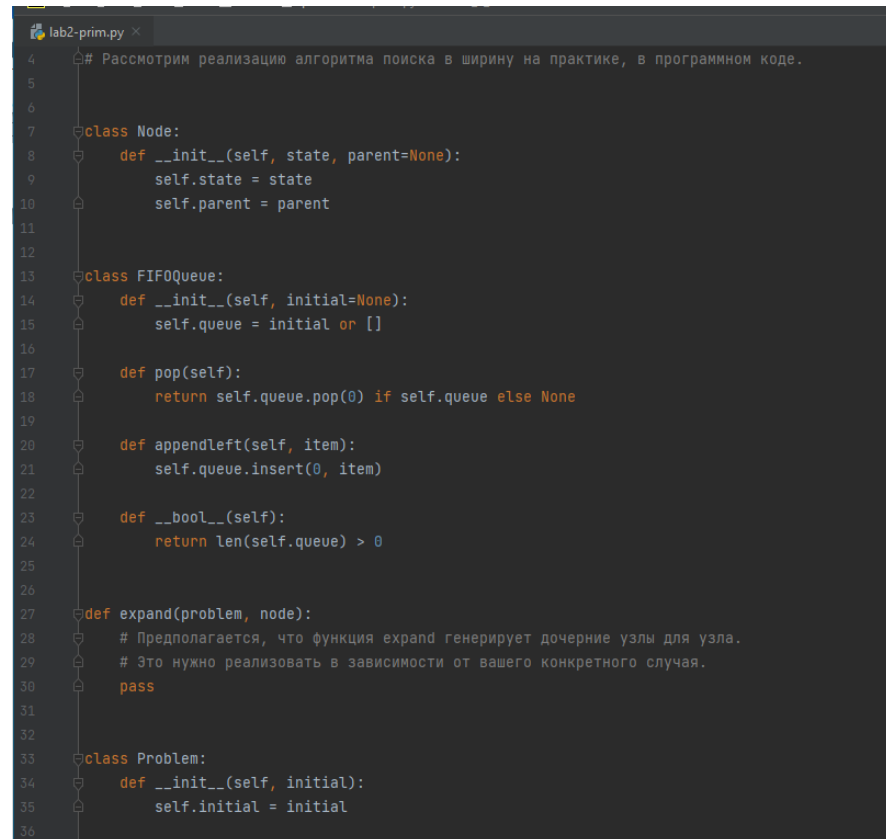
Ставрополь, 2025 г.

Тема: исследование поиска в ширину.

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

Ход работы:

Задание 1. Создал проект в папке репозитория. Приступил к работе с примером. Добавил новый файл lab2-prim.py.



```
lab2-prim.py x
4  # Рассмотрим реализацию алгоритма поиска в ширину на практике, в программном коде.
5
6
7  class Node:
8      def __init__(self, state, parent=None):
9          self.state = state
10         self.parent = parent
11
12
13  class FIFOQueue:
14      def __init__(self, initial=None):
15          self.queue = initial or []
16
17      def pop(self):
18          return self.queue.pop(0) if self.queue else None
19
20      def appendleft(self, item):
21          self.queue.insert(0, item)
22
23      def __bool__(self):
24          return len(self.queue) > 0
25
26
27  def expand(problem, node):
28      # Предполагается, что функция expand генерирует дочерние узлы для узла.
29      # Это нужно реализовать в зависимости от вашего конкретного случая.
30      pass
31
32
33  class Problem:
34      def __init__(self, initial):
35          self.initial = initial
36
```

Рисунок 1. Выполнение lab2-prim.py

Задание 2. Необходимо для задачи "Расширенный подсчет количества островов в бинарной матрице" подготовить собственную матрицу, для которой с помощью разработанной в предыдущем пункте программы, подсчитаем количество островов. Разработаем матрицу:

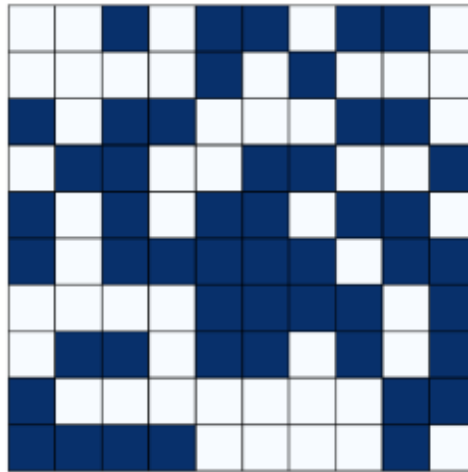


Рисунок 2. Создание матрицы.

Из данной среды образуем бинарную матрицу, где белым цветом будет представлена вода, а синим представлена земля. Связанные единицы формируют остров. Необходимо подсчитать общее количество островов в данной матрице. Острова могут соединяться как по вертикали и горизонтали, так и по диагонали. Напишем программу:

```

zadanie1.py
def countIslands(mat):
    # Базовый вариант
    if not mat or not len(mat):
        return 0

    # Матрица 'M x N'
    (M, N) = (len(mat), len(mat[0]))

    # запоминает, обработана ячейка или нет
    processed = [[False for x in range(N)] for y in range(M)]

    island = 0
    for i in range(M):
        for j in range(N):
            # запускает BFS с каждого необработанного узла и увеличивает количество островов
            if mat[i][j] == 1 and not processed[i][j]:
                BFS(mat, processed, i, j)
                island = island + 1

    return island

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

```

Рисунок 3. Разработанная программа

Вывод программы показал результат 3, что соответствует выбранной нами матрице с островами.

Задание 4. Необходимо 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. Заполним матрицу:

```
maze = [
    [1, 0, 1, 1, 1, 0, 1, 1, 1, 1],
    [1, 0, 1, 0, 1, 0, 0, 0, 0, 1],
    [1, 1, 1, 0, 1, 1, 1, 1, 0, 1],
    [0, 0, 1, 0, 0, 0, 0, 1, 0, 1],
    [1, 1, 1, 1, 1, 1, 0, 1, 1, 1],
    [1, 0, 0, 0, 0, 1, 0, 0, 0, 1],
    [1, 1, 1, 1, 0, 1, 1, 1, 1, 1],
    [0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
    [1, 1, 1, 1, 1, 1, 1, 0, 1, 1],
    [1, 0, 0, 0, 0, 0, 1, 1, 1, 1],
]
```

Рисунок 4. Матрица в виде лабиринта

Напишем программу для поиска кратчайшего пути через лабиринт, используя алгоритм поиска в ширину (BFS). Лабиринт представлен в виде бинарной матрицы, где 1 обозначает проход, а 0 — стену.

Задание 6. Необходимо для построенного графа лабораторной работы 1 написать программу на языке программирования Python, которая с помощью алгоритма поиска в ширину находит минимальное расстояние между начальным и конечным пунктами. Сравним найденное решение с решением, полученным вручную. Найдем минимальное расстояние между городами Углич и Фурманов.

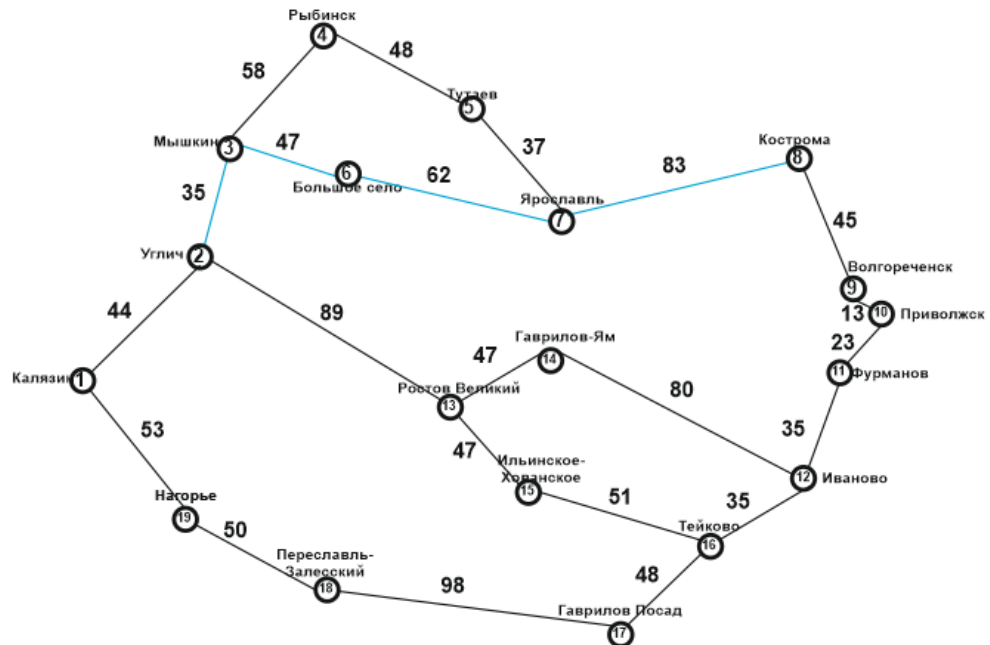


Рисунок 5. Граф из лабораторной работы №1.

Если считать вручную, то минимальное расстояние составляет 251 км.

Далее составим программу и проверим:

```

107  }
108
109
110  start = 2
111  goal = 11
112  problem = Problem(start, goal)
113  path, distance = breadth_first_search(problem, graph)
114  print("Кратчайший путь:", path)
115  print("Длина пути:", distance)
116
117
118  if __name__ == "__main__":
119
120
121
122
123
124
125

```

zadanie3

C:\ProgramData\Anaconda3\python.exe "C:/Users/София/Desktop/Воронкин_искусств/Laba_ii_2-main/Laba_ii_2-main/src/zadanie3.py"

Кратчайший путь: [2, 13, 14, 12, 11]

Длина пути: 251

Process finished with exit code 0

Результат программы вывел так же 251 км.

Ответы на контрольные вопросы:

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

В поиске в ширину используется очередь FIFO (First In, First Out), где узлы извлекаются в том порядке, в котором были добавлены.

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

Это позволяет гарантировать, что узлы будут расширяться в порядке их глубины, т.е., сначала обрабатываются более близкие к корню узлы, затем более удаленные. Это является основной стратегией поиска в ширину.

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

Узлы, которые дольше находятся в очереди, будут извлекаться и расширяться первыми, так как очередь FIFO гарантирует, что первым выходит узел, который был добавлен раньше всех.

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

Следующими будут расширены узлы, которые непосредственно связаны с корневым узлом, то есть узлы на глубине 1.

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

Это гарантирует, что первое найденное решение является оптимальным (самым коротким путём) в терминах количества шагов от корня до цели.

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

Временная сложность поиска в ширину зависит от двух факторов: коэффициента разветвления (то есть количества потомков у каждого узла) и глубины целевого узла (то есть минимального числа шагов до цели). Поиск в ширину проходит все узлы уровня за уровнем, начиная с корня, поэтому на каждом новом уровне количество узлов для обработки резко возрастает. Чем больше потомков у каждого узла и чем глубже находится целевое состояние, тем больше узлов нужно обработать. Это приводит к экспоненциальному росту времени выполнения при увеличении этих двух параметров.

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

Основной фактор, влияющий на объем памяти, который требует поиск в ширину, — это количество узлов, которые нужно сохранить на самом

нижнем уровне поиска. Так как алгоритм должен хранить в памяти все узлы на каждом уровне, пока они не будут обработаны, наибольшее количество узлов накапливается на последнем уровне. Чем больше у узлов потомков и чем

глубже находится целевое состояние, тем больше узлов нужно хранить одновременно, и это сильно увеличивает потребность в памяти.

8. В каких случаях поиск в ширину считается полным?

Поиск в ширину считается полным, если пространство состояний конечно или если решение существует на конечной глубине, т.е. если есть гарантии достижения цели.

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

Поскольку поиск в ширину хранит в памяти все узлы на каждом уровне, он требует много памяти, особенно при высоком коэффициенте разветвления и большой глубине .

10. В чем заключается оптимальность поиска в ширину?

Поиск в ширину является оптимальным по количеству шагов, если все шаги имеют одинаковую длину, так как он первым находит кратчайший путь от начального состояния к целевому.

11. Какую задачу решает функция `breadth_first_search`?

`Breadth_first_search` решает задачу поиска пути от начального состояния к целевому состоянию, используя алгоритм поиска в ширину.

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

`Problem` представляет собой объект задачи, который содержит начальное состояние, целевое состояние, а также методы для определения допустимых действий и проверки достижения цели.

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

Node(problem.initial) создаёт корневой узел дерева поиска, представляющий начальное состояние задачи, с которого начинается процесс поиска.

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

Если начальное состояние уже является целевым, функция `breadth_first_search` немедленно вернет этот узел, завершая поиск.

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

Frontier использует очередь FIFO для обеспечения расширения узлов в порядке их глубины, что соответствует стратегии поиска в ширину.

16. Какую роль выполняет множество `reached`?

Множество `reached` хранит состояния, которые уже были достигнуты, чтобы избежать повторного расширения одного и того же состояния и предотвратить заикливание.

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

Это предотвращает повторное расширение одного и того же состояния, экономя время и память.

18. Какую функцию выполняет цикл `while frontier`?

Цикл `while frontier` продолжает процесс поиска, пока остаются узлы для расширения. Он завершится, когда либо будет найдено решение, либо будут исчерпаны все узлы.

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

Узел извлекается из очереди для дальнейшего расширения, и его дочерние узлы (возможные новые состояния) будут добавлены в очередь.

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

Функция `expand` генерирует дочерние узлы для данного узла, используя допустимые действия и правила перехода в задаче `problem`.

21. Как определяется, что состояние узла является целевым?

Целевое состояние определяется с помощью метода `is_goal` объекта `problem`, который проверяет, соответствует ли текущее состояние целевому.

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

Если состояние узла не является целевым и не было достигнуто ранее, оно добавляется в множество `reached` и очередь `frontier` для дальнейшего расширения.

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

В алгоритме поиска в ширину дочерний узел добавляется в конец очереди, а не в начало, чтобы соблюсти принцип FIFO (очередь с извлечением элементов в порядке их поступления). Это гарантирует, что узлы будут обрабатываться по мере их добавления в очередь, начиная с узлов, расположенных ближе к корневому, и заканчивая узлами на более глубоких уровнях. Использование метода `appendleft(child)` применимо, скорее, для алгоритма поиска в глубину, который следует стратегии LIFO (стек), где узлы обрабатываются в порядке последнего добавления.

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

Если решение не найдено, функция возвращает специальное значение `failure`, показывающее, что достижение цели невозможно.

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

Узел `failure` обычно имеет состояние `None` или «неудача» и длина пути бесконечность. Он возвращается, если поиск завершился, но не было найдено решения.

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