

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:  
Гуляницкий Александр  
3 курс, группа ИТС-б-о-22-1,  
11.03.02 «Инфокоммуникационные  
технологии и системы связи», очная  
форма обучения

---

(подпись)

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

---

(подпись)

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

Ставрополь, 2024 г.

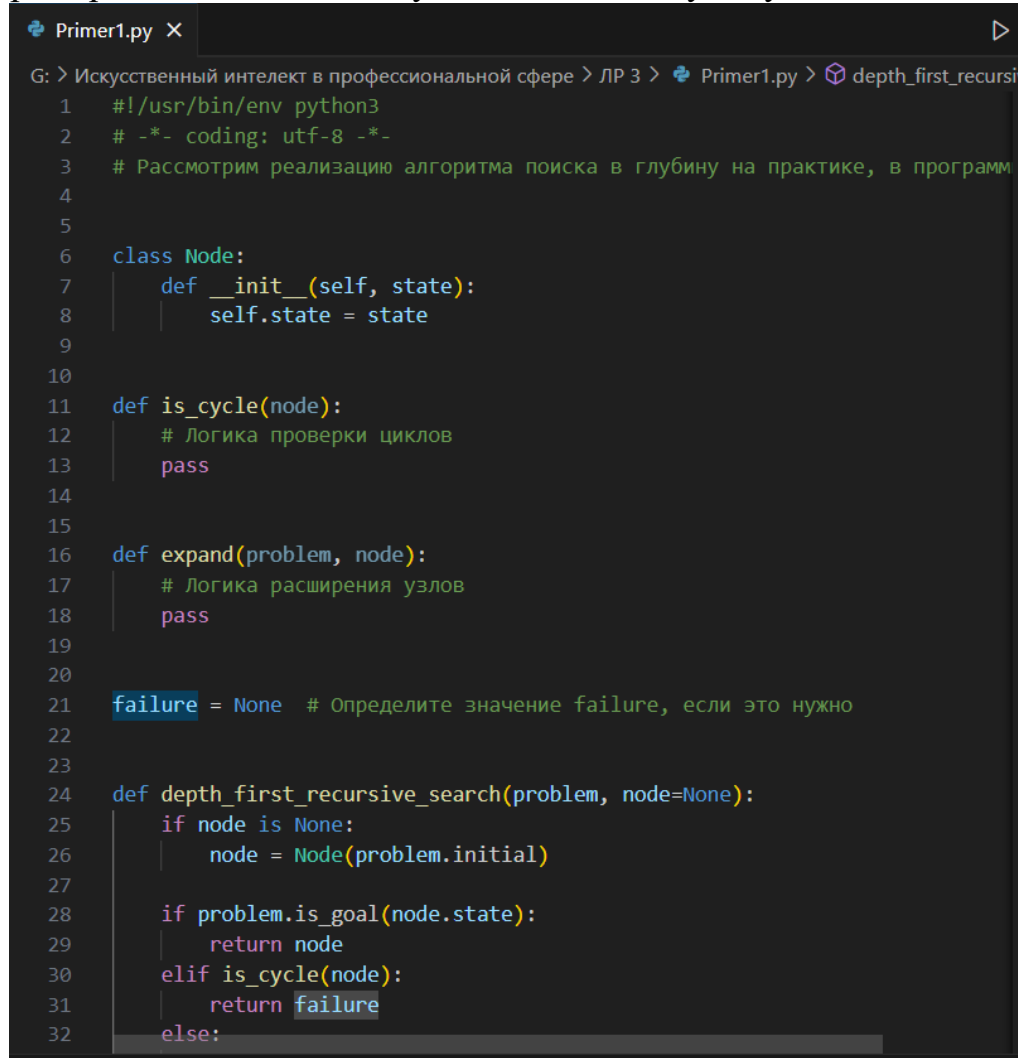
## Исследование поиска в глубину.

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

Ссылка на репозиторий: <https://github.com/Alexander-its/LR3.git>

Ход работы:

Пример 1. Реализация на Python поиска в глубину.

A screenshot of a code editor window titled 'Primer1.py'. The editor shows Python code for a depth-first search algorithm. The code includes a Node class, a cycle check function, an expansion function, and a main search function. The code is as follows:

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  # Рассмотрим реализацию алгоритма поиска в глубину на практике, в программ
4
5
6  class Node:
7      def __init__(self, state):
8          self.state = state
9
10
11  def is_cycle(node):
12      # Логика проверки циклов
13      pass
14
15
16  def expand(problem, node):
17      # Логика расширения узлов
18      pass
19
20
21  failure = None # Определите значение failure, если это нужно
22
23
24  def depth_first_recursive_search(problem, node=None):
25      if node is None:
26          node = Node(problem.initial)
27
28      if problem.is_goal(node.state):
29          return node
30      elif is_cycle(node):
31          return failure
32      else:
```

Рисунок 1 – Код для выполнения примера 1

Задание 1. Поиск самого длинного пути в матрице.

```
IDZ1.py X
G: > Искусственный интеллект в профессиональной сфере > ЛР 3 > IDZ1.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  # Поиск самого длинного пути в матрице
4
5  def longest_path(matrix, start_char):
6      rows = len(matrix)
7      cols = len(matrix[0])
8
9      directions = [(-1, -1), (-1, 0), (-1, 1),
10                  (0, -1), (0, 1),
11                  (1, -1), (1, 0), (1, 1)]
12
13      memo = {}
14
15      def dfs(x, y, prev_char):
16          if (x, y) in memo:
17              return memo[(x, y)]
18
19          max_length = 1
20
21          for dx, dy in directions:
22              new_x, new_y = x + dx, y + dy
23              if (0 <= new_x < rows and 0 <= new_y < cols and
24                  ord(matrix[new_x][new_y]) == ord(prev_char) + 1):
25                  max_length = max(max_length, 1 + dfs(new_x, new_y, matrix[new_x][new_y]))
26
27          memo[(x, y)] = max_length
28          return max_length
29
30      longest = 0
31
32      for x in range(rows):
33          for y in range(cols):
34              if matrix[x][y] == start_char:
35                  longest = max(longest, dfs(x, y, matrix[x][y]))
36
37      return longest
```

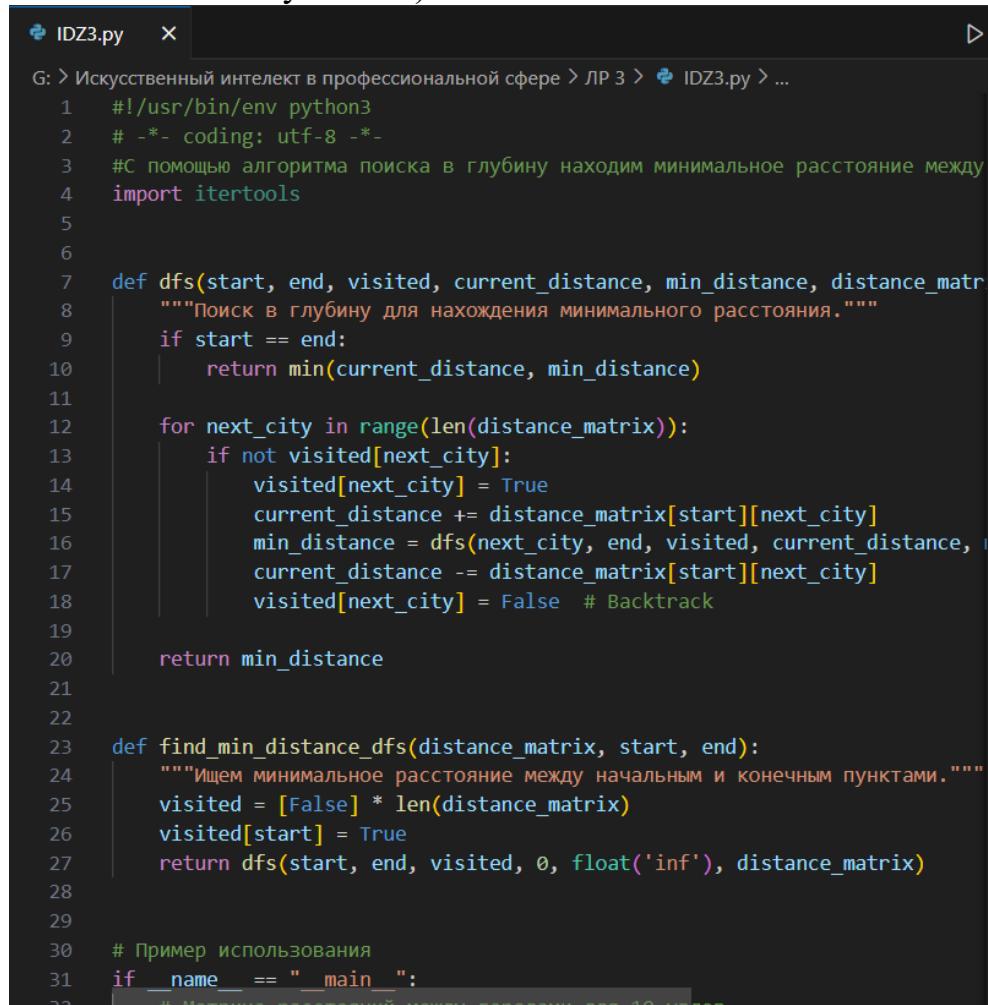
Рисунок 2 – Код для выполнения задания 1

Задание 2. Генерирование списка возможных слов из матрицы символов.

```
IDZ2.py X
G: > Искусственный интеллект в профессиональной сфере > ЛР 3 > IDZ2.py > find_words > dfs
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  # Генерирование списка возможных слов из матрицы символов
4
5  def find_words(board, dictionary):
6      found_words = set()
7      rows, cols = len(board), len(board[0])
8
9      directions = [(-1, -1), (-1, 0), (-1, 1),
10                  (0, -1), (0, 1),
11                  (1, -1), (1, 0), (1, 1)]
12
13      def dfs(x, y, path, visited):
14          path += board[x][y] # Добавляем текущий символ к пути
15          if path in dictionary:
16              found_words.add(path) # Если слово найдено, добавляем его в найденные
17
18          if len(path) > max_length:
19              return
20
21          for dx, dy in directions: # Проверяем все возможные направления
22              new_x, new_y = x + dx, y + dy
23
24              if (0 <= new_x < rows and
25                  0 <= new_y < cols and
26                  (new_x, new_y) not in visited):
27                  visited.add((new_x, new_y)) # Добавляем в посещенные
28                  dfs(new_x, new_y, path, visited)
29                  visited.remove((new_x, new_y)) # Убираем из посещенных
30
31      max_length = max(len(word) for word in dictionary) # Длина самого длинного слова
32
33      for x in range(rows):
34          for y in range(cols):
35              dfs(x, y, "", set())
36
37      return found_words
```

Рисунок 3 - Код для выполнения задания 2

Задание 3. Алгоритм поиска в глубину (минимальное расстояние между начальным и конечным пунктами).



```
IDZ3.py x
G: > Искусственный интеллект в профессиональной сфере > ЛР 3 > IDZ3.py > ...
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3  #С помощью алгоритма поиска в глубину находим минимальное расстояние между
4  import itertools
5
6
7  def dfs(start, end, visited, current_distance, min_distance, distance_matrix):
8      """Поиск в глубину для нахождения минимального расстояния."""
9      if start == end:
10         return min(current_distance, min_distance)
11
12     for next_city in range(len(distance_matrix)):
13         if not visited[next_city]:
14             visited[next_city] = True
15             current_distance += distance_matrix[start][next_city]
16             min_distance = dfs(next_city, end, visited, current_distance, min_distance, distance_matrix)
17             current_distance -= distance_matrix[start][next_city]
18             visited[next_city] = False # Backtrack
19
20     return min_distance
21
22
23  def find_min_distance_dfs(distance_matrix, start, end):
24      """Ищем минимальное расстояние между начальным и конечным пунктами."""
25      visited = [False] * len(distance_matrix)
26      visited[start] = True
27      return dfs(start, end, visited, 0, float('inf'), distance_matrix)
28
29
30  # Пример использования
31  if __name__ == "__main__":
32      # Матрица расстояний между городами для 10 городов
```

Рисунок 4 - Код для выполнения задания 3

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