Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Цифровая кафедра

Лабораторная работа №2

по дисциплине «Системы искусственного интеллекта»

Выполнил:

студент

группы АУЗ-561

Жолобов А.В.

Проверил:

преподаватель

Фокин Р.О.

Волгоград, 2023 г.

|  |  |  |
| --- | --- | --- |
| Лабораторная работа №2 | Ф.И.О. | Жолобов А.В. |
| Группа | АУЗ-561 |
| Преподаватель | Фокин Р.О. |
| Дата сдачи | 12.11.2023 |

**Цель работы:**

Целью данной работы является изучение алгоритмов поиска на графах и их применение для решения задач искусственного интеллекта с использованием Python.

**Задачи:**

– Протестировать программу на различных графах и сценариях поиска;

– Убедится, что алгоритмы корректно находят путь (если он существует) и работает эффективно.

**Ход работы:**

Пошаговое выполнение примера из презентации:

Лабораторная работа №2.

**Выполнение индивидуального задания:**

1. Импорт библиотек:

import networkx as nx  
import matplotlib.pyplot as plt

1. Реализация класса Graph:

class Graph:  
 def \_\_init\_\_(self):  
 self.nodes = nx.Graph()  
  
 def add\_node(self, node):  
 self.nodes.add\_node(node)  
  
 def add\_edge(self, node1, node2):  
 self.nodes.add\_edge(node1, node2)  
  
 def get\_neighbors(self, node):  
 return list(self.nodes.neighbors(node))  
  
 def get\_all\_nodes(self):  
 return list(self.nodes.nodes())

1. Метод поиска в глубину:

def dfs(self, start, end):  
 visited = set()  
 path = []  
 self.\_dfs\_util(start, end, visited, path)  
  
 if not path:  
 print("Path not found")  
 else:  
 print("Path:", " -> ".join(str(path)))  
  
def \_dfs\_util(self, node, end, visited, path):  
 visited.add(node)  
 path.append(node)  
  
 if node == end:  
 return True  
  
 for neighbor in self.nodes.neighbors(node):  
 if neighbor not in visited:  
 if self.\_dfs\_util(neighbor, end, visited, path):  
 return True  
  
 path.pop()  
 return False

1. Метод поиска в ширину:

def bfs(self, start, end):  
 visited = set()  
 queue = [(start, [])]  
   
 while queue:  
 node, path = queue.pop(0)  
   
 if node not in visited:  
 visited.add(node)  
 path.append(node)  
   
 if node == end:  
 print("Path:", " -> ".join(str(path)))  
 return  
   
 for neighbor in self.nodes.neighbors(node):  
 if neighbor not in visited:  
 queue.append((neighbor, list(path)))  
   
 print("Path not found")

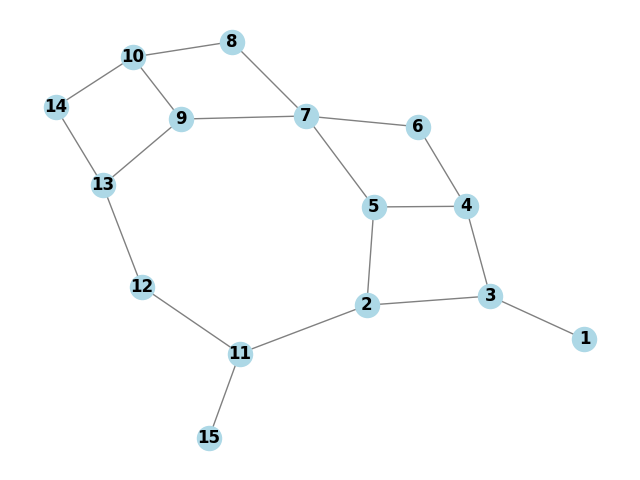
1. Визуализация:

def visualize(self):  
 nx.draw(self.nodes, with\_labels=True, node\_color='lightblue', edge\_color='gray', font\_weight='bold')  
 plt.show()

1. Реализуем метод визуализация:

if \_\_name\_\_ == '\_\_main\_\_':  
 *# Создание нового графа* graph = Graph()  
  
 *# Добавление вершин в граф* graph.add\_node(1)  
 graph.add\_node(2)  
 graph.add\_node(3)  
 graph.add\_node(4)  
 graph.add\_node(5)  
 graph.add\_node(6)  
 graph.add\_node(7)  
 graph.add\_node(8)  
 graph.add\_node(9)  
 graph.add\_node(10)  
 graph.add\_node(11)  
 graph.add\_node(12)  
 graph.add\_node(13)  
 graph.add\_node(14)  
 graph.add\_node(15)  
   
 *# Добавление ребер в граф* graph.add\_edge(1,3)  
 graph.add\_edge(2,5)  
 graph.add\_edge(2,11)  
 graph.add\_edge(3,2)  
 graph.add\_edge(3,4)  
 graph.add\_edge(4,5)  
 graph.add\_edge(4,6)  
 graph.add\_edge(5,7)  
 graph.add\_edge(6,7)  
 graph.add\_edge(7,8)  
 graph.add\_edge(7,9)  
 graph.add\_edge(8,10)  
 graph.add\_edge(9,10)  
 graph.add\_edge(9,13)  
 graph.add\_edge(10,14)  
 graph.add\_edge(11,12)  
 graph.add\_edge(11,15)  
 graph.add\_edge(12,13)  
 graph.add\_edge(13,14)  
   
 *# Инициализация метода поиска в глубину* print("DFS:")  
 graph.dfs(1,9)  
   
 *# Инициализация метода поиска в ширину* print("BFS:")  
 graph.bfs(1,9)  
   
 *# Визуализация графа* graph.visualize()

1. Результаты выполнения тестовой программы:



DFS:

Path: [ -> 1 -> , -> -> 3 -> , -> -> 2 -> , -> -> 5 -> , -> -> 4 -> , -> -> 6 -> , -> -> 7 -> , -> -> 8 -> , -> -> 1 -> 0 -> , -> -> 9 -> ]

BFS:

Path: [ -> 1 -> , -> -> 3 -> , -> -> 2 -> , -> -> 5 -> , -> -> 7 -> , -> -> 9 -> ]

1. Создадим и протестируем программу на другом графе:

# Создание нового графа

graph2 **=** Graph**()**

# Добавление вершин в граф

graph2**.**add\_node**(**1**)**

graph2**.**add\_node**(**2**)**

graph2**.**add\_node**(**3**)**

graph2**.**add\_node**(**4**)**

graph2**.**add\_node**(**5**)**

graph2**.**add\_node**(**6**)**

graph2**.**add\_node**(**7**)**

graph2**.**add\_node**(**8**)**

graph2**.**add\_node**(**9**)**

graph2**.**add\_node**(**10**)**

# Добавление ребер в граф

graph2**.**add\_edge**(**1**,** 3**)**

graph2**.**add\_edge**(**2**,** 5**)**

graph2**.**add\_edge**(**2**,** 7**)**

graph2**.**add\_edge**(**3**,** 2**)**

graph2**.**add\_edge**(**3**,** 4**)**

graph2**.**add\_edge**(**4**,** 5**)**

graph2**.**add\_edge**(**4**,** 6**)**

graph2**.**add\_edge**(**5**,** 7**)**

graph2**.**add\_edge**(**6**,** 7**)**

graph2**.**add\_edge**(**7**,** 8**)**

graph2**.**add\_edge**(**8**,** 9**)**

graph2**.**add\_edge**(**9**,** 10**)**

graph2**.**add\_edge**(**10**,** 1**)**

# Инициализация метода поиска в глубину

**print(**"DFS:"**)**

graph2**.**dfs**(**1**,** 6**)**

# Инициализация метода поиска в ширину

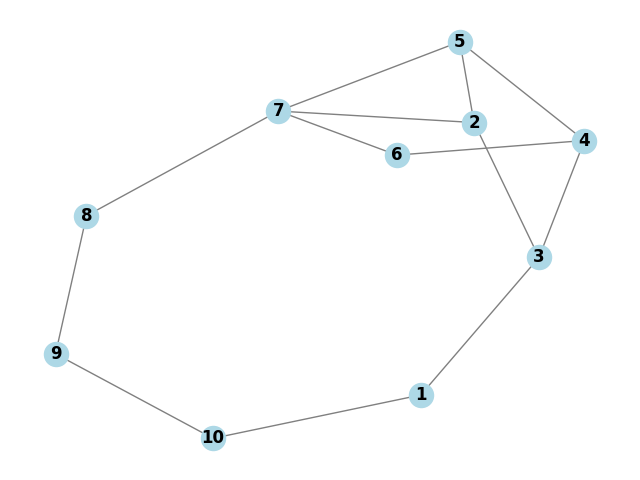
**print(**"BFS:"**)**

graph2**.**bfs**(**1**,** 6**)**

# Визуализация графа

graph2**.**visualize**()**

1. Результаты выполнения программы:



DFS:

Path: [ -> 1 -> , -> -> 3 -> , -> -> 2 -> , -> -> 5 -> , -> -> 4 -> , -> -> 6 -> ]

BFS:

Path: [ -> 1 -> , -> -> 3 -> , -> -> 4 -> , -> -> 6 -> ]

**Вывод:**

В ходе выполнения работы были успешно выполнены задачи по тестированию программы на различных графах и сценариях поиска. Примененный алгоритм (в данном случае, алгоритм поиска в ширину – BFS) продемонстрировал корректное функционирование, обеспечивая надежное нахождение пути между заданными точками, при условии, что такой путь существует. Эффективность алгоритма также была проверена, и он продемонстрировал хорошие результаты на тестовых данных.