

Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
“НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО”

Факультет	Программной Инженерии и Компьютерной Техники
Направление подготовки (специальность)	Программная инженерия
Дисциплина	Системы искусственного интеллекта

ЛАБОРАТОРНАЯ РАБОТА 2
ОТЧЕТ

Выполнил студент: Кулагин Вячеслав Дмитриевич (408946)
Группа: Р3309
Преподаватель: Болдырева Елена Александровна (157150)

г. Санкт-Петербург

2025

Содержание	
ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ	2
ОТЧЕТ О ХОДЕ ВЫПОЛНЕНИЯ	2
ЗАКЛЮЧЕНИЕ	6

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

- Создать программу, которая позволяет пользователю ввести запрос через командную строку. Например, информацию о себе, своих интересах и предпочтениях в контексте выбора видеоигры - на основе фактов из БЗ (из первой лабы)/Онтологии(из второй).

- Использовать введенные пользователем данные, чтобы выполнить логические запросы к БЗ/Онтологии.

- На основе полученных результатов выполнения запросов, система должна предоставить рекомендации или советы, связанные с выбором из БЗ или онтологии.

- Система должна выдавать рекомендации после небольшого диалога с пользователем.

ОТЧЕТ О ХОДЕ ВЫПОЛНЕНИЯ

Была разработана программа на Python с использованием библиотеки owlready2, которая позволяет считывать онтологию из Protégé и использовать ее как источник данных для стандартных типов данных в Python.

Также была взята библиотека для работы с графами – из онтологии строится граф, по которому выбираются необходимые пользователю маршруты. Чтобы не реализовывать вручную алгоритм поиска в ширину графа (BFS), а также алгоритм Дейкстры, используется библиотека networkx

Исходный код:

```
from owlready2 import *
import networkx as nx

def load_graph_from_ontology(onto):
    G = nx.Graph()

    all_routes = onto.Route.instances()
    for route in all_routes:
        endpoints = route.HasEndpoint
        length_list = route.hasLength
        color_individual_list = route.hasColor

        if len(endpoints) != 2 or not length_list or not color_individual_list:
            continue

        city1, city2 = endpoints[0], endpoints[1]
```

```

        city1_name = str(city1.hasName[0])
        city2_name = str(city2.hasName[0])
        length = int(length_list[0])
        color_name = color_individual_list[0].name.capitalize()

        G.add_edge(city1_name, city2_name, weight=length, color=color_name)

    return G

def find_most_direct_path_bfs(graph, start, end):
    try:
        return nx.shortest_path(graph, source=start, target=end)
    except nx.NetworkXNoPath:
        return None

def find_shortest_path_dijkstra(graph, start, end):
    try:
        return nx.dijkstra_path(graph, source=start, target=end,
weight='weight')
    except nx.NetworkXNoPath:
        return None

def parse_user_input(text, cities):
    pattern = re.compile(r"найти\s+(.+?)\s+из\s+(.+?)\s+в\s+(.+)",
re.IGNORECASE)
    match = pattern.search(text)
    if not match: return None

    intent, start_city, end_city = match.group(1).lower(),
match.group(2).strip(), match.group(3).strip()
    if start_city not in cities or end_city not in cities:
        print(f"Ошибка: один из городов ('{start_city}' или '{end_city}') не
найден.")
        print(f"Доступные города: {' '.join(cities)}")
        return None
    return {"intent": intent, "start": start_city, "end": end_city}

def print_path_details(graph, path, title):
    print(f"\n{title}:")
    print(f"' -> '.join(path)")
    total_length = 0
    for i in range(len(path) - 1):
        city1, city2 = path[i], path[i + 1]
        edge_data = graph.get_edge_data(city1, city2)

```

```

length = edge_data['weight']
color = edge_data['color']

total_length += length
print(f" - Участок: {city1}-{city2}, Длина: {length}, Цвет: {color}")
print(f" > Пересадок: {len(path) - 2}, Общая длина: {total_length}")

def main():
    try:
        onto = get_ontology("file://owl-lab1.owl").load()
        print("Онтология загружена. Построение графа маршрутов...")
        graph = load_graph_from_ontology(onto)
        print("Граф построен. Система готова к работе.")
    except Exception as e:
        print(e)
        return

print("\nДоступный формат: 'найти маршрут из ГородаА в ГородБ'"
      "\nИли: 'найти цвет из ГородаА в ГородБ'")
print("Для выхода введите 'выход'.")

available_cities = list(graph.nodes)

while True:
    user_input = input("\nВаш запрос: ")
    if user_input.lower() == 'выход': break

    parsed_data = parse_user_input(user_input, available_cities)
    if not parsed_data:
        print("Неверный формат. Пример: 'найти маршрут из Киев в Петроград'")
        continue

    start, end, intent = parsed_data["start"], parsed_data["end"],
    parsed_data["intent"]

    if intent == "маршрут":
        direct_path = find_most_direct_path_bfs(graph, start, end)
        shortest_path = find_shortest_path_dijkstra(graph, start, end)

        if not direct_path:
            print(f"Маршруты из г. {start} в г. {end} не найдены.")
            continue

        if direct_path == shortest_path:
            print_path_details(graph, shortest_path, "Самый короткий и прямой маршрут")

```

```

        else:
            print("\nНайдены два оптимальных варианта маршрута:")
            print_path_details(graph, direct_path, "Самый прямой маршрут  
(меньше пересадок)")
            print_path_details(graph, shortest_path, "Самый короткий маршрут  
(меньше общая длина)")

    elif intent == "цвет":
        if graph.has_edge(start, end):
            color = graph.get_edge_data(start, end)['color']
            print(f"Цвет прямого маршрута {start}-{end}: {color}")
        else:
            print(f"Прямого маршрута {start}-{end} нет.")
    else:
        print(f"Неизвестное намерение '{intent}'. Попробуйте 'маршрут' или  
'цвет'.")

if __name__ == "__main__":
    main()

```

Результаты работы программы представлены на Рисунках 1, 2 и 3.

```

Онтология загружена. Построение графа маршрутов...
Граф построен. Система готова к работе.

Доступный формат: 'найти маршрут из ГородА в ГородБ'
Или: 'найти цвет из ГородА в ГородБ'
Для выхода введите 'выход'.

Ваш запрос: найти маршрут из Киев в Петроград

Самый короткий и прямой маршрут:
Киев -> Вильнюс -> Петроград
- Участок: Киев-Вильнюс, Длина: 2, Цвет: Any
- Участок: Вильнюс-Петроград, Длина: 4, Цвет: Blue
> Пересадок: 1, Общая длина: 6

Ваш запрос:

```

Рисунок 1 – Запрос на маршрут между Киевом и Петроградом

Ваш запрос: *найти маршрут из Москва в Петроград*

Самый короткий и прямой маршрут:

Москва -> Петроград

- Участок: Москва-Петроград, Длина: 4, Цвет: White

> Пересадок: 0, Общая длина: 4

Ваш запрос:

Рисунок 2 – Запрос на маршрут между Москвой и Петроградом

Ваш запрос: *найти цвет из Москва в Петроград*

Цвет прямого маршрута Москва-Петроград: White

Ваш запрос:

Рисунок 3 – Запрос на цвет между Москвой и Петроградом

ЗАКЛЮЧЕНИЕ

В ходе лабораторной работы была разработана программа на Python, которая позволяет узнавать маршруты и цвет между населенными пунктами в настольной игре Ticket to Ride, основываясь на онтологии Protégé созданной в рамках предыдущей лабораторной работы.