Федеральное государственное бюджетное образовательное учреждение высшего образования «Байкальский государственный университет»

Прикладная информатика

Объектно-ориентированное программирование

Лабораторная работа №2.1

Вариант: 3

Выполнил студент

Вайкус Яков Михайлович

Группа № ИС-23-1

Преподаватель: Федотов Никита Андреевич

г. Иркутск

2025

Оглавление

[**Задание** 3](#_Toc207876137)

[**Код программы** 3](#_Toc207876138)

[Вывод программы: 4](#_Toc207876139)

# **Задание**

# Лабораторная работа 3: Симуляция логистической системы с циклами и словарями Цель: Моделировать логистическую систему с использованием циклов и словарей. Задание: Разработайте программу для симуляции доставки грузов. Используйте словарь для хранения данных о маршрутах (ключ — город, значение — список расстояний до соседей). Реализуйте цикл while для поиска кратчайшего пути между городами с условием if для проверки лимита топлива (1000 км). Добавьте рекурсивную функцию для оптимизации маршрута через промежуточные узлы (глубина 6). Обработайте KeyError для отсутствующих городов и используйте декоратор для логирования. Требования: • Начальные данные: {"Москва": [700, 800], "Спб": [180]}. • Декоратор логирует время и маршрут. • Результат — словарь с оптимальным путём. Ожидаемый результат: Например, {"Москва-Спб": 700} с логом

# **Код программы**

import time

def logger(func):

    def wrapper(\*args, \*\*kwargs):

        start = time.strftime("%Y-%m-%d %H:%M:%S")

        try:

            result = func(\*args, \*\*kwargs)

            if result and not (len(result) == 1 and list(result.values())[0] == 0):

              print(f"[{start}] Маршрут: {result}")

            return result

        except Exception as e:

            print(f"[{start}] Ошибка: {e}")

            raise

    return wrapper

routes = {

    "Москва": {"Спб": 700, "Нижний": 800},

    "Спб": {"Новгород": 180, "Хельсинки": 600},

    "Нижний": {"Казань": 400},

    "Новгород": {"Псков": 250},

    "Казань": {"Уфа": 500},

    "Уфа": {"Екатеринбург": 900},

}

for city, neighbors in list(routes.items()):

    for neighbor in neighbors:

        if neighbor not in routes:

            routes[neighbor] = {}

fuel\_limit = 1000

@logger

def find\_route(start, end, visited=None, depth=0, max\_depth=6):

    if visited is None:

        visited = set()

    if depth > max\_depth:

        return None

    if start == end:

        return {f"{start}": 0}

    if start not in routes:

        raise KeyError(f"Город {start} не найден")

    best = None

    for neighbor, dist in routes[start].items():

        if neighbor not in visited:

            subpath = find\_route(neighbor, end, visited | {start}, depth+1, max\_depth)

            if subpath:

                total = dist + list(subpath.values())[0]

                if total > fuel\_limit:

                    continue

                path = f"{start}-" + list(subpath.keys())[0]

                candidate = {path: total}

                if best is None or total < list(best.values())[0]:

                    best = candidate

    return best

# Тесты

print("Москва → Спб:", find\_route("Москва", "Спб"))

print("Спб → Новгород:", find\_route("Спб", "Новгород"))

print("Нижний → Казань:", find\_route("Нижний", "Казань"))

print("Новгород → Псков:", find\_route("Новгород", "Псков"))

print("Спб → Хельсинки:", find\_route("Спб", "Хельсинки"))

# **Вывод программы:**

[2025-09-24 14:22:00] Маршрут: {'Москва-Спб': 700}

Москва → Спб: {'Москва-Спб': 700}

[2025-09-24 14:22:00] Маршрут: {'Спб-Новгород': 180}

Спб → Новгород: {'Спб-Новгород': 180}

[2025-09-24 14:22:00] Маршрут: {'Нижний-Казань': 400}

Нижний → Казань: {'Нижний-Казань': 400}

[2025-09-24 14:22:00] Маршрут: {'Новгород-Псков': 250}

Новгород → Псков: {'Новгород-Псков': 250}

[2025-09-24 14:22:00] Маршрут: {'Спб-Хельсинки': 600}

Спб → Хельсинки: {'Спб-Хельсинки': 600}