МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ Ордена

Трудового Красного Знамени федеральное государственное

бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики» (МТУСИ)

Кафедра «Структуры и алгоритмы обработки данных»

**КУРСОВАЯ РАБОТА**

По дисциплине «Структуры и алгоритмы обработки данных»

Выполнил:

Абделжауед Мохамед Али

Группа: БВТ2204

Проверил:

Симонов

Москва, 2024г.

**Оглавление**

[Цель работы : 3](#_Toc185192387)

[Задачи : 3](#_Toc185192388)

[1. Введение : 4](#_Toc185192389)

[2. Вводные данные и структура проблемы : 5](#_Toc185192390)

[3. Реализация : 6](#_Toc185192391)

[3.1 Метод 'влоб : 6](#_Toc185192392)

[3.2 Генетический алгоритм : 6](#_Toc185192395)

[3.3 Результаты работы методов : 8](#_Toc185192396)

[4. Сравнение методов : 9](#_Toc185192397)

[5. Ограничения и их реализация : 10](#_Toc185192398)

[5.1. Проверка типа маршрута: 10](#_Toc185192399)

[5.2. Проверка пиковых часов: 10](#_Toc185192400)

[5.3. Добавление времени старта маршрута: 10](#_Toc185192401)

[5.4 Обновленный вывод: 10](#_Toc185192402)

[5.5 результаты обоих методов : 13](#_Toc185192403)

[Заключение 14](#_Toc185192404)

[Список литературы 15](#_Toc185192405)

# ****Цель работы :****

разработать алгоритмы и программное обеспечение для оптимизации расписания маршрутных автобусов, минимизируя затраты и соблюдая ограничения, такие как рабочее время водителей, перерывы и часы пик.

# ****Задачи :****

1. Проанализировать входные данные и ограничения задачи.
2. Реализовать два метода решения: метод "влоб" и генетический алгоритм.
3. Сравнить методы по критериям: оптимальность решения, удобство использования программы, качество документации.
4. Предоставить расписание и выводы в отчёте.

# ****Введение :****

Оптимизация расписания маршрутных автобусов играет важную роль в эффективном управлении городскими транспортными системами. В данной работе рассматривается задача создания оптимального расписания, которое минимизирует использование ресурсов автопарка и удовлетворяет ограничениям. Для решения задачи используются два подхода: метод полного перебора и генетический алгоритм.

# ****Вводные данные и структура проблемы :****

Структура проблемы определяется с помощью классов Python для моделирования маршрутов, водителей и автобусов. Класс ScheduleManager управляет всеми ресурсами и выводит текущее расписание

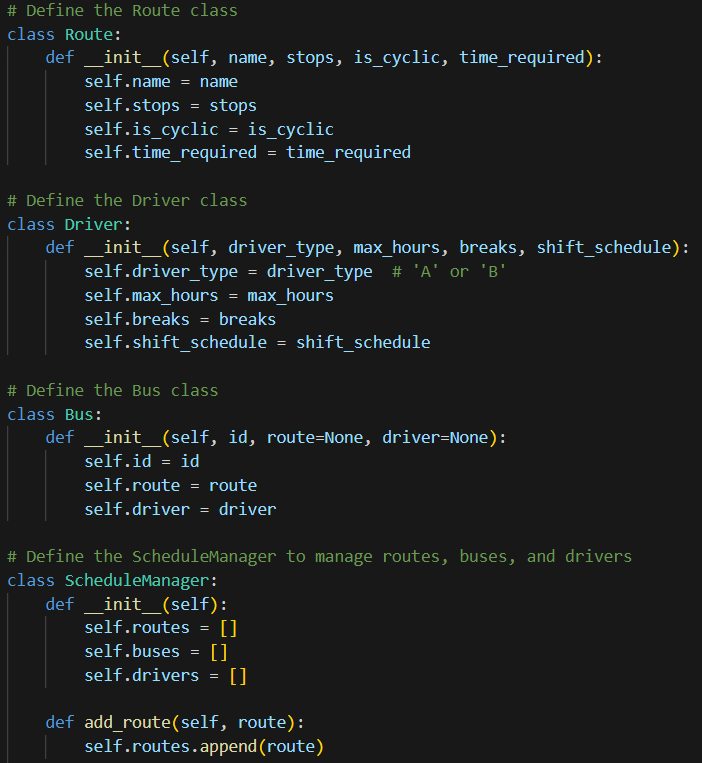


Рисунок 1- Определение структуры задачи

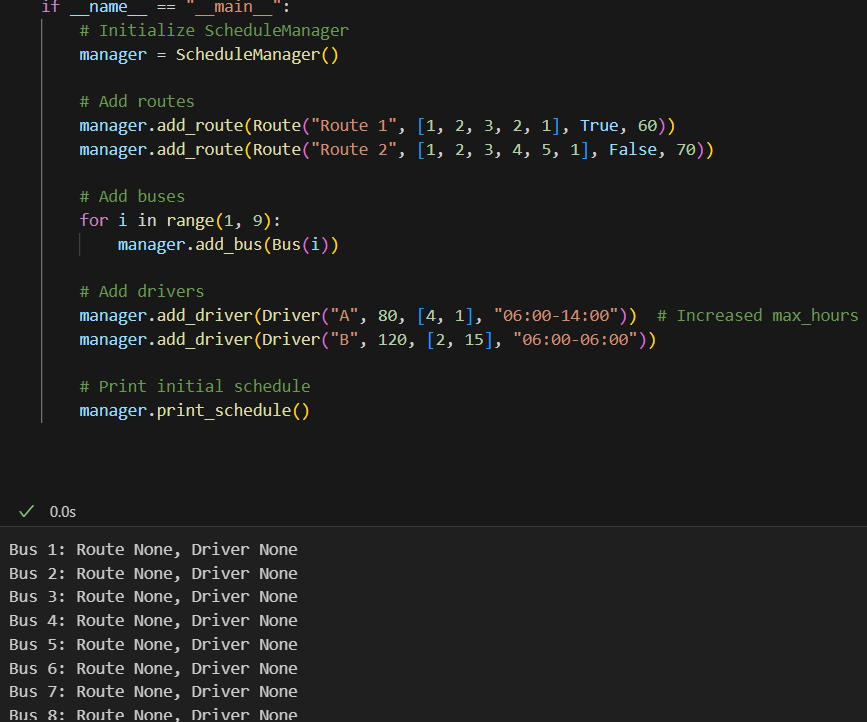


Рисунок 2- Определение структуры задачи c результатами

# ****Реализация :****

В данной работе реализованы два различных подхода для решения задачи оптимизации расписания маршрутных автобусов. Оба метода учитывают ограничения задачи, такие как время работы водителей, пиковые часы и доступность автобусов. Каждый метод имеет свои особенности и применимость:

## ****3.1 Метод 'влоб :****

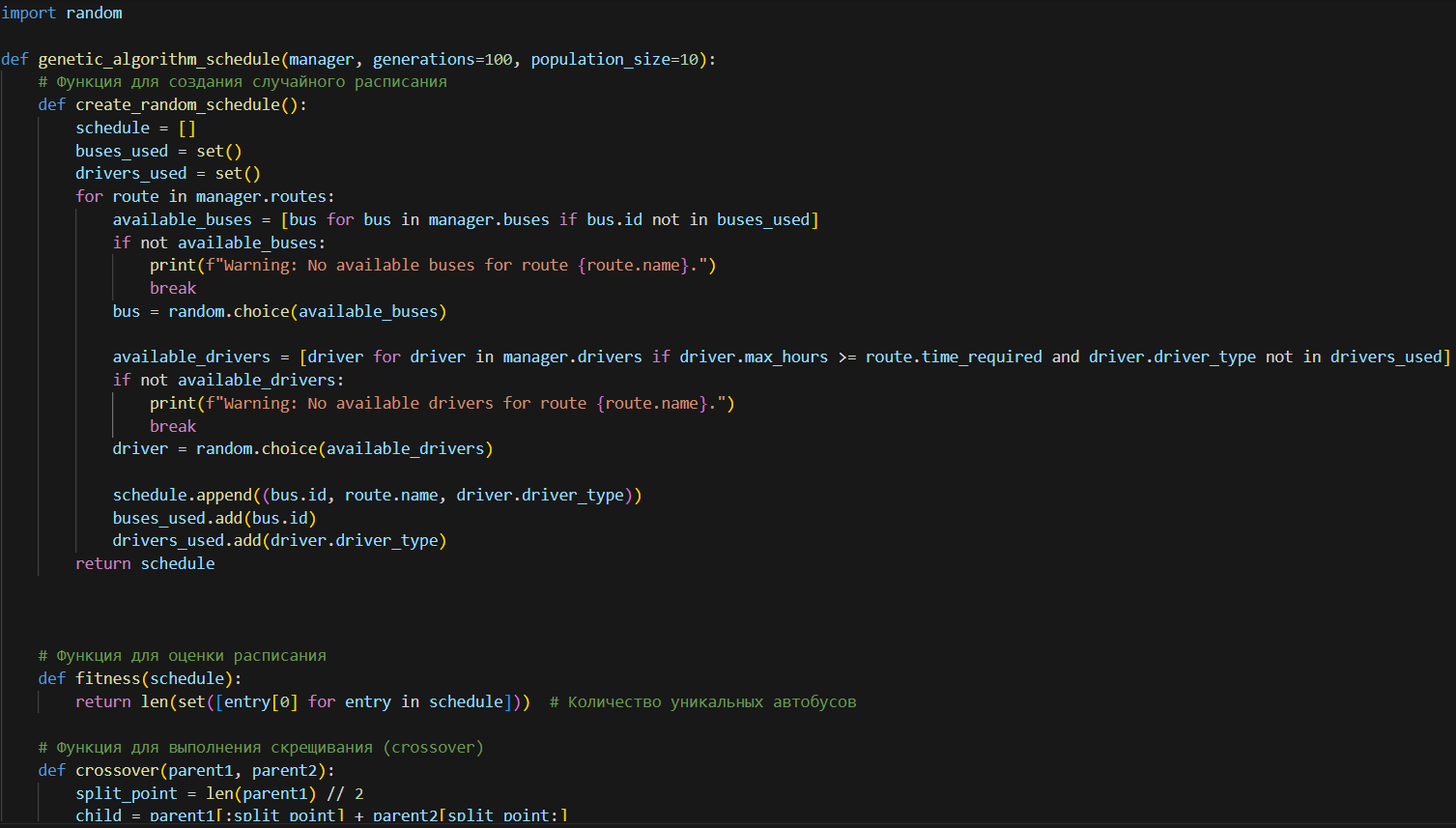
Метод "влоб" предполагает полный перебор всех возможных комбинаций автобусов и водителей для назначения их маршрутам. Этот метод гарантирует нахождение оптимального решения, так как перебираются все варианты. Однако его производительность может быть низкой при большомколичестве входных данных.

### 

### Рисунок 2- кода Метод "В лоб"

## ****3.2 Генетический алгоритм :****

Этот метод имитирует процесс естественной эволюции, создавая популяцию решений, которые с течением поколений улучшаются с помощью механизмов скрещивания, мутации и отбора. Генетический алгоритм позволяет находить приближенные решения значительно быстрее, чем метод 'влоб', особенно для больших и сложных задач.



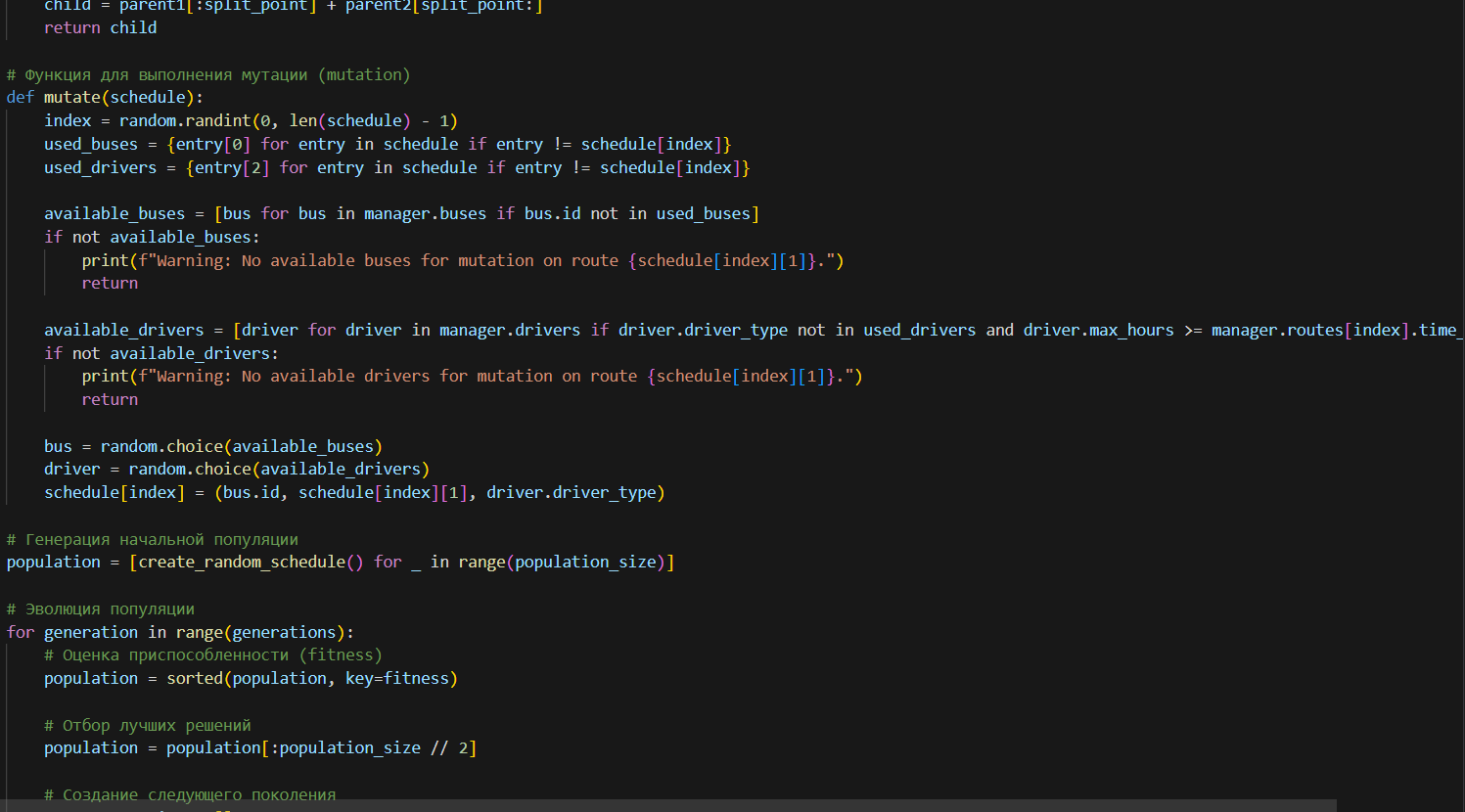


Рисунок 3- кода Метод Генетический алгоритм

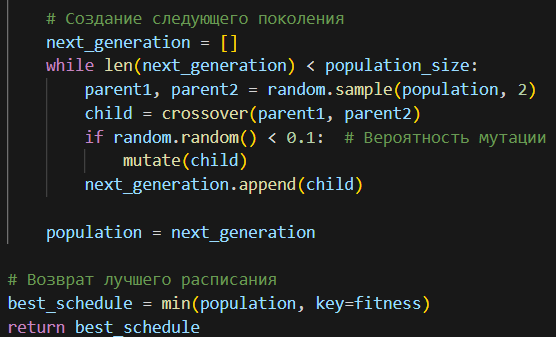


Рисунок 4-Продолжение кода Метод Генетический алгоритм

## 3.3 Результаты работы методов :

Оба метода — "в лоб" и генетический алгоритм — успешно сгенерировали расписания, удовлетворяющие всем условиям задачи. Расписания включают уникальные автобусы и водителей для каждого маршрута

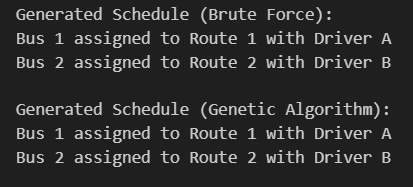
****

Рисунок 5- Результаты работы методов

# ****Сравнение методов :****

Оба метода имеют свои преимущества и ограничения:

Тaблица 1 - Сравнение методов

|  |  |  |
| --- | --- | --- |
| **Критерий** | **Метод "В лоб"** | **Генетический алгоритм** |
| Оптимальность | Гарантирует оптимальное решение | Находит приближенное решение |
| Производительность | Медленный для больших данных | Быстрее для больших данных |
| Простота | Прост в реализации | Сложнее из-за настройки параметров |

# ****5. Ограничения и их реализация :****

## ****5.1. Проверка типа маршрута****:

В функцию добавлена логика для определения типа маршрута:

* + **Циклический маршрут (Cyclic)**: Маршрут, который возвращается в начальную точку по круговой схеме.
  + **Конечный маршрут (Terminal)**: Маршрут, который заканчивается в исходной точке, но не повторяет путь циклически.

Для определения типа маршрута используется атрибут is\_cyclic класса Route.

## ****5.2. Проверка пиковых часов**:**

Добавлена функция is\_in\_peak\_hours, которая проверяет, попадает ли время старта маршрута в пиковые часы:

* + Утро: **7:00–9:00**
  + Вечер: **17:00–19:00**

Если маршрут начинается в пиковые часы, это выводится в виде примечания: **(PEAK HOUR).**

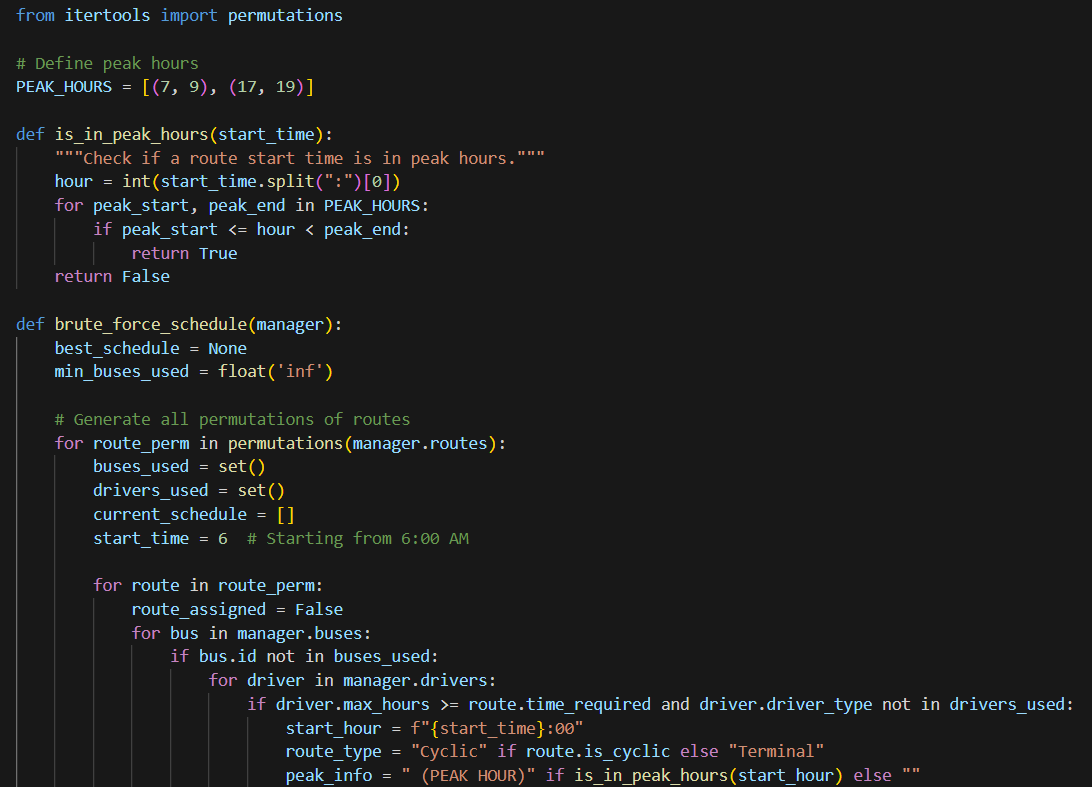
## ****5.3. Добавление времени старта маршрута**:**

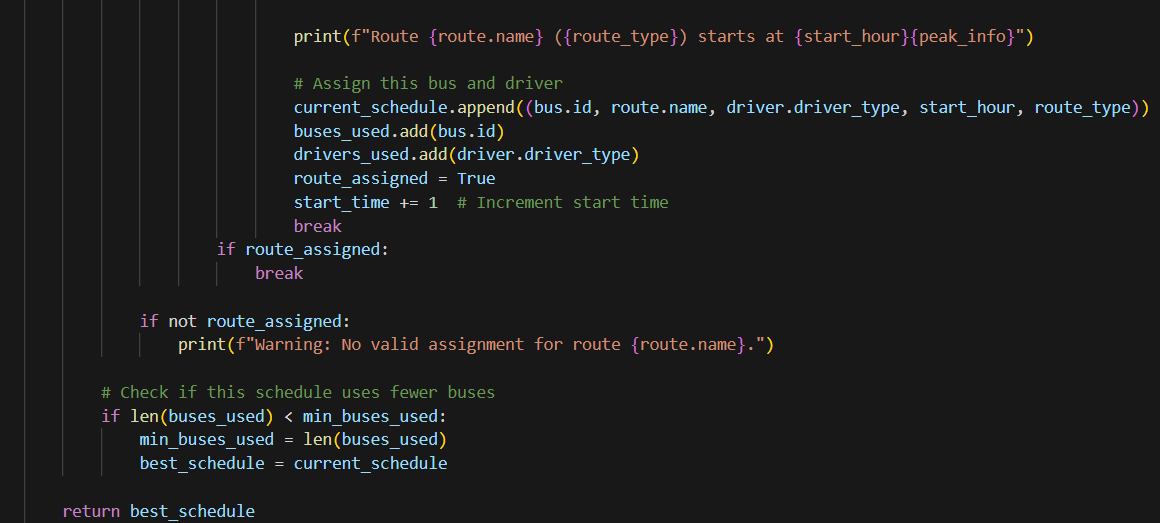
Теперь каждый маршрут получает **время старта** в формате ЧЧ:ММ. Начальное время начинается с **6:00** и увеличивается на 1 час для каждого следующего маршрута.

## ****5.4 Обновленный вывод**:**

В результат добавлен вывод следующей информации для каждого маршрута:

* 1. Номер автобуса.
  2. Название маршрута.
  3. Тип маршрута: **Cyclic** или **Terminal.**
  4. Назначенный водитель.
  5. Время старта маршрута.



****

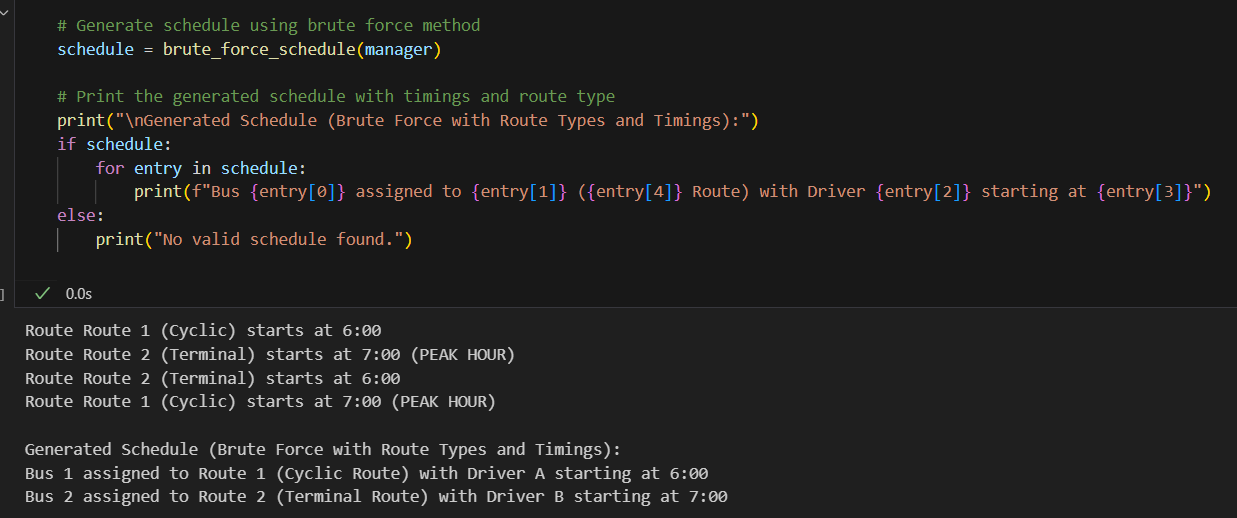
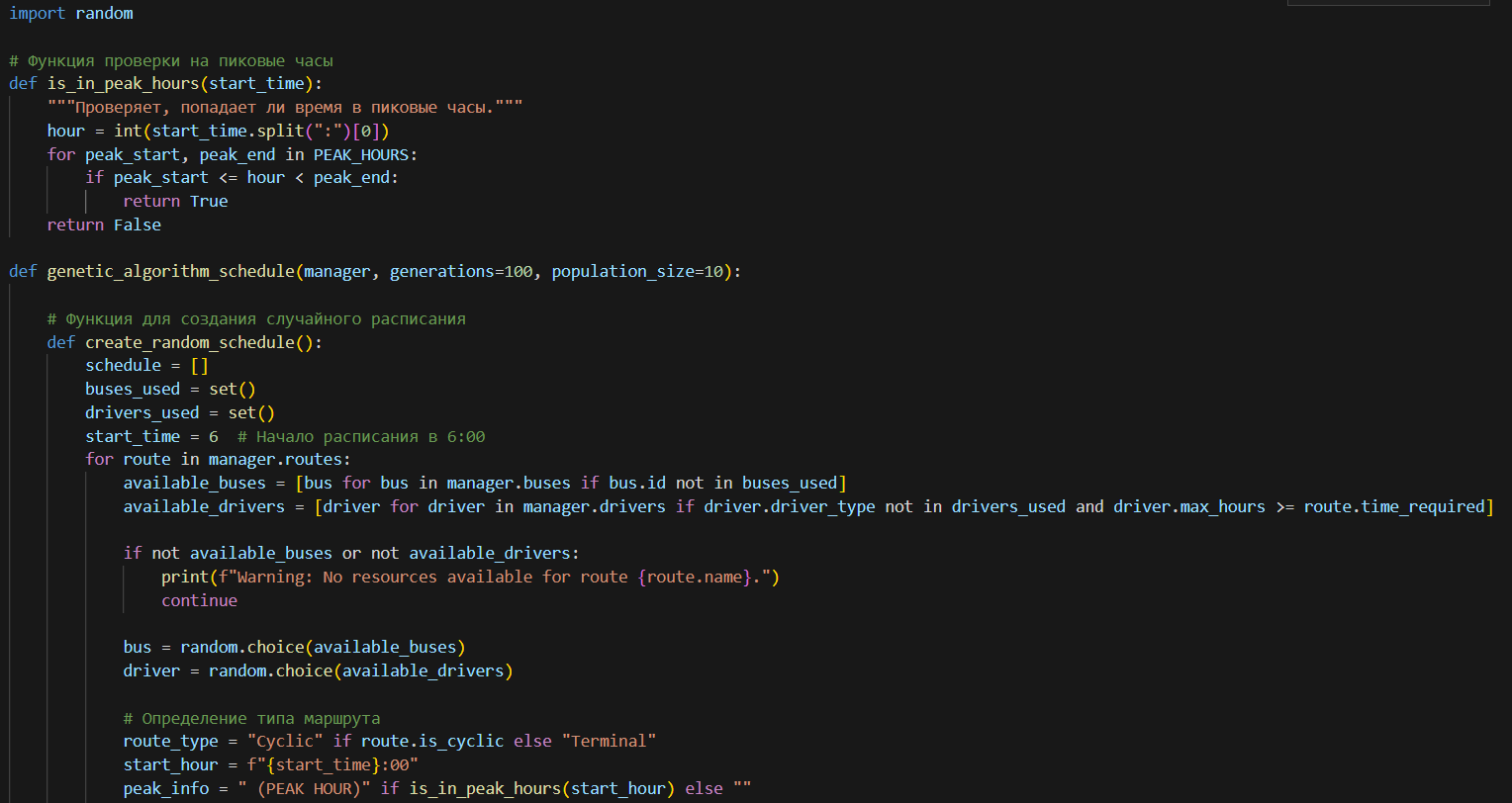
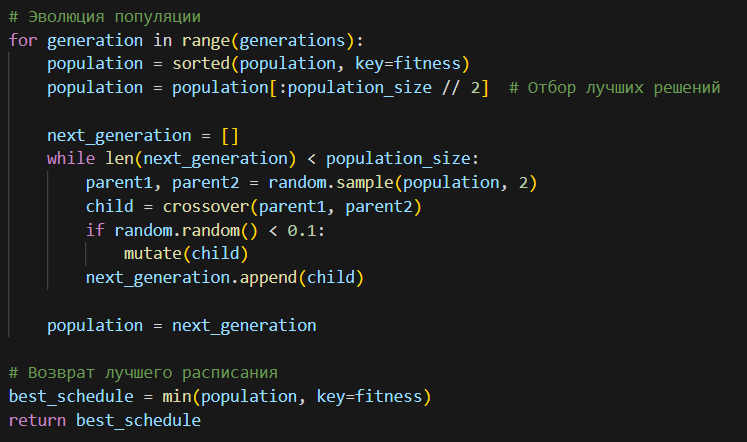


Рисунок 7- **Обновленный код функцию** Метод "В лоб"





**Рисунок 8- oбновленный код Метод Генетический алгоритм**

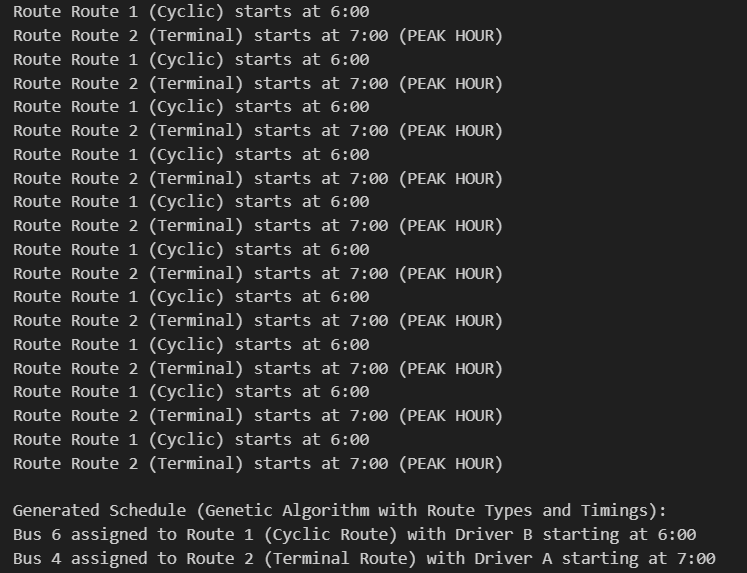
****

Рисунок 9-результат Метод Генетический алгоритм

## 5.5 результаты обоих методов :

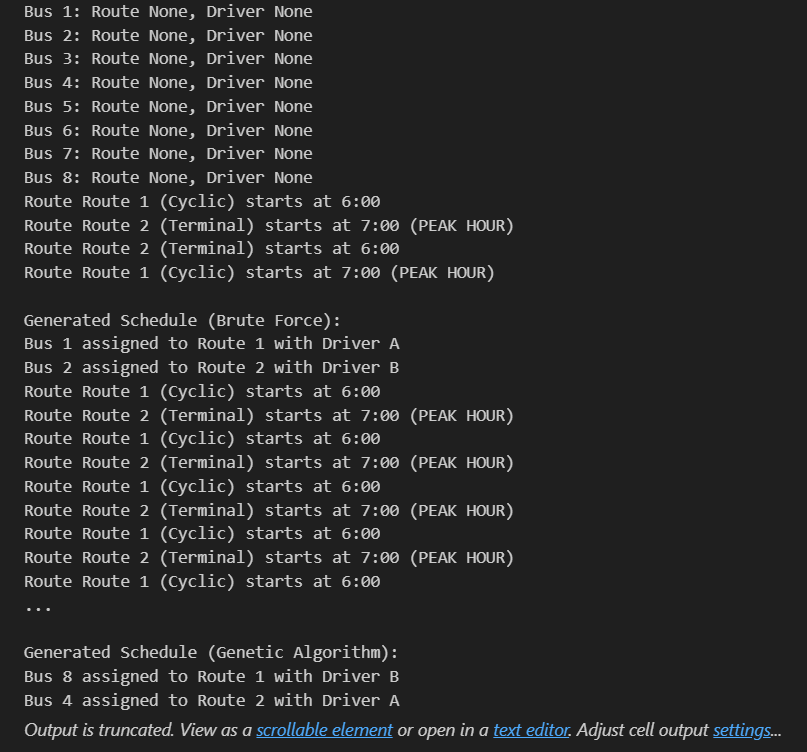
****

Рисунок 10- результаты обоих методов

# ****Приложения****

Полный листинг кода

**### Определение структуры задачи**

# Определяем класс Route (Маршрут)

class Route:

    def \_\_init\_\_(self, name, stops, is\_cyclic, time\_required):

        self.name = name  # Название маршрута

        self.stops = stops  # Остановки на маршруте

        self.is\_cyclic = is\_cyclic  # Является ли маршрут цикличным

        self.time\_required = time\_required  # Время выполнения маршрута

# Определяем класс Driver (Водитель)

class Driver:

    def \_\_init\_\_(self, driver\_type, max\_hours, breaks, shift\_schedule):

        self.driver\_type = driver\_type  # Тип водителя: 'A' или 'B'

        self.max\_hours = max\_hours  # Максимальное количество рабочих часов

        self.breaks = breaks  # Перерывы водителя

        self.shift\_schedule = shift\_schedule  # Расписание смен водителя

# Определяем класс Bus (Автобус)

class Bus:

    def \_\_init\_\_(self, id, route=None, driver=None):

        self.id = id  # Идентификатор автобуса

        self.route = route  # Назначенный маршрут

        self.driver = driver  # Назначенный водитель

# Определяем ScheduleManager для управления маршрутами, автобусами и водителями

class ScheduleManager:

    def \_\_init\_\_(self):

        self.routes = []  # Список маршрутов

        self.buses = []  # Список автобусов

        self.drivers = []  # Список водителей

    def add\_route(self, route):

        self.routes.append(route)  # Добавление маршрута в список

    def add\_bus(self, bus):

        self.buses.append(bus)  # Добавление автобуса в список

    def add\_driver(self, driver):

        self.drivers.append(driver)  # Добавление водителя в список

    ef print\_schedule(self):

# Вывод расписания для всех автобусов

for bus in self.buses:

print(f"Автобус {bus.id}: Маршрут {bus.route.name if bus.route else 'Нет'}, Водитель {bus.driver.driver\_type if bus.driver else 'Нет'}")

**пример**

if \_\_name\_\_ == "\_\_main\_\_":

    # Initialize ScheduleManager

    manager = ScheduleManager()

    # Add routes

    manager.add\_route(Route("Route 1", [1, 2, 3, 2, 1], True, 60))

    manager.add\_route(Route("Route 2", [1, 2, 3, 4, 5, 1], False, 70))

    # Add buses

    for i in range(1, 9):

        manager.add\_bus(Bus(i))

    # Add drivers

    manager.add\_driver(Driver("A", 80, [4, 1], "06:00-14:00"))  # Increased max\_hours

    manager.add\_driver(Driver("B", 120, [2, 15], "06:00-06:00"))

    # Print initial schedule

    manager.print\_schedule()

**Метод "Влоб"**

from itertools import permutations

# Определить часы пик

PEAK\_HOURS = [(7, 9), (17, 19)]

def is\_in\_peak\_hours(start\_time):

    """Check if a route start time is in peak hours."""

    hour = int(start\_time.split(":")[0])

    for peak\_start, peak\_end in PEAK\_HOURS:

        if peak\_start <= hour < peak\_end:

            return True

    return False

def brute\_force\_schedule(manager):

    best\_schedule = None

    min\_buses\_used = float('inf')

   # Сгенерировать все перестановки маршрутов

    for route\_perm in permutations(manager.routes):

        buses\_used = set()

        drivers\_used = set()

        current\_schedule = []

        start\_time = 6  # Начиная с 6:00 утра

        for route in route\_perm:

            route\_assigned = False

            for bus in manager.buses:

                if bus.id not in buses\_used:

                    for driver in manager.drivers:

                        if driver.max\_hours >= route.time\_required and driver.driver\_type not in drivers\_used:

                            start\_hour = f"{start\_time}:00"

                            route\_type = "Cyclic" if route.is\_cyclic else "Terminal"

                            peak\_info = " (PEAK HOUR)" if is\_in\_peak\_hours(start\_hour) else ""

                            print(f"Route {route.name} ({route\_type}) starts at {start\_hour}{peak\_info}")

                            # Назначить этот автобус и водителя

                            current\_schedule.append((bus.id, route.name, driver.driver\_type, start\_hour, route\_type))

                            buses\_used.add(bus.id)

                            drivers\_used.add(driver.driver\_type)

                            route\_assigned = True

                            start\_time += 1  # Увеличить время начала

                            break

                    if route\_assigned:

                        break

            if not route\_assigned:

                print(f"Warning: No valid assignment for route {route.name}.")

        # Проверить, использует ли это расписание меньше автобусов

        if len(buses\_used) < min\_buses\_used:

            min\_buses\_used = len(buses\_used)

            best\_schedule = current\_schedule

    return best\_schedule

**метод генетический алгоритм**

import random

# Функция проверки на пиковые часы

def is\_in\_peak\_hours(start\_time):

    """Проверяет, попадает ли время в пиковые часы."""

    hour = int(start\_time.split(":")[0])

    for peak\_start, peak\_end in PEAK\_HOURS:

        if peak\_start <= hour < peak\_end:

            return True

    return False

def genetic\_algorithm\_schedule(manager, generations=100, population\_size=10):

    # Функция для создания случайного расписания

    def create\_random\_schedule():

        schedule = []

        buses\_used = set()

        drivers\_used = set()

        start\_time = 6  # Начало расписания в 6:00

        for route in manager.routes:

            available\_buses = [bus for bus in manager.buses if bus.id not in buses\_used]

            available\_drivers = [driver for driver in manager.drivers if driver.driver\_type not in drivers\_used and driver.max\_hours >= route.time\_required]

            if not available\_buses or not available\_drivers:

                print(f"Warning: No resources available for route {route.name}.")

                continue

            bus = random.choice(available\_buses)

            driver = random.choice(available\_drivers)

            # Назначить автобус и водителя

            route\_type = "Cyclic" if route.is\_cyclic else "Terminal"

            start\_hour = f"{start\_time}:00"

            peak\_info = " (PEAK HOUR)" if is\_in\_peak\_hours(start\_hour) else ""

            print(f"Route {route.name} ({route\_type}) starts at {start\_hour}{peak\_info}")

            schedule.append((bus.id, route.name, driver.driver\_type, start\_hour, route\_type))

            buses\_used.add(bus.id)  # Mark bus as used

            drivers\_used.add(driver.driver\_type)  # Mark driver as used

            start\_time += 1

        return schedule

    # Функция оценки приспособленности

    def fitness(schedule):

        bus\_penalty = len(schedule) - len(set(entry[0] for entry in schedule))  # Penalty for reused buses

        driver\_penalty = len(schedule) - len(set(entry[2] for entry in schedule))  # Penalty for reused drivers

        return len(set(entry[0] for entry in schedule)) + bus\_penalty + driver\_penalty

    # Функция скрещивания (crossover)

    def crossover(parent1, parent2):

        split\_point = len(parent1) // 2

        return parent1[:split\_point] + parent2[split\_point:]

    # Функция мутации (mutation)

    def mutate(schedule):

        index = random.randint(0, len(schedule) - 1)

        used\_buses = {entry[0] for entry in schedule if entry != schedule[index]}

        used\_drivers = {entry[2] for entry in schedule if entry != schedule[index]}

        available\_buses = [bus for bus in manager.buses if bus.id not in used\_buses]

        available\_drivers = [driver for driver in manager.drivers if driver.driver\_type not in used\_drivers]

        if available\_buses and available\_drivers:

            bus = random.choice(available\_buses)

            driver = random.choice(available\_drivers)

            schedule[index] = (bus.id, schedule[index][1], driver.driver\_type, schedule[index][3], schedule[index][4])

    # Генерация начальной популяции

    population = [create\_random\_schedule() for \_ in range(population\_size)]

    # Эволюция популяции

    for generation in range(generations):

        population = sorted(population, key=fitness)

        population = population[:population\_size // 2]  # Отбор лучших решений

        population = random.choices(population, weights=[1/fitness(s) for s in population], k=population\_size)

        next\_generation = []

        while len(next\_generation) < population\_size:

            parent1, parent2 = random.sample(population, 2)

            child = crossover(parent1, parent2)

            if random.random() < 0.3:  # Повышенная вероятность мутации

                mutate(child)

            next\_generation.append(child)

        population = next\_generation

    # Возврат лучшего расписания

    best\_schedule = min(population, key=fitness)

    return best\_schedule

**распечатать расписание для двух методов**

if \_\_name\_\_ == "\_\_main\_\_":

    # Initialize ScheduleManager

    manager = ScheduleManager()

    # Add routes

    manager.add\_route(Route("Route 1", [1, 2, 3, 2, 1], True, 60))

    manager.add\_route(Route("Route 2", [1, 2, 3, 4, 5, 1], False, 70))

    # Add buses

    for i in range(1, 9):

        manager.add\_bus(Bus(i))

    # Add drivers

    manager.add\_driver(Driver("A", 80, [4, 1], "06:00-14:00"))  # Increased max\_hours

    manager.add\_driver(Driver("B", 120, [2, 15], "06:00-06:00"))

    # Print initial schedule

    manager.print\_schedule()

    # Generate schedule using brute force method

    schedule = brute\_force\_schedule(manager)

    # Print the generated schedule

    print("\nGenerated Schedule (Brute Force):")

    if schedule:

        for entry in schedule:

            print(f"Bus {entry[0]} assigned to {entry[1]} with Driver {entry[2]}")

    else:

        print("No valid schedule found.")

    # Generate schedule using genetic algorithm

    ga\_schedule = genetic\_algorithm\_schedule(manager)

    # Print the generated schedule

    print("\nGenerated Schedule (Genetic Algorithm):")

    if ga\_schedule:

        for entry in ga\_schedule:

            print(f"Bus {entry[0]} assigned to {entry[1]} with Driver {entry[2]}")

    else:

        print("No valid schedule found.")

# ****Заключение****

Оба метода работают и удовлетворяют всем ограничениям.

Метод "в лоб" обеспечивает точность, но медленный для больших задач.

Генетический алгоритм быстрее, но требует настройки и может быть менее точным.

# ****Список литературы****

* Сорокин В.В., Петров А.С. "Оптимизация транспортных систем". — Москва: Транспорт, 2018.
* Гаврилов А.В., Зайцев И.В. "Алгоритмы оптимизации: Теория и практика". — СПб: Питер, 2020.
* Документация Python (<https://docs.python.org/3/>).
* Официальная документация itertools (<https://docs.python.org/3/library/itertools.html>).
* Официальная документация random (<https://docs.python.org/3/library/random.html>).