

МИНЕСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ
КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное
учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра «Математическая кибернетика и информационные технологии»

«Структуры и алгоритмы обработки данных»

Курсовая работа
по теме: «Оптимизация расписания автобусов»

Работу выполнила
Студентка группы БВТ2203
Шедова Анастасия Романовна

Работу проверил:
ассистент
Симонов Сергей Евгеньевич

Москва 2024

Содержание

ВВЕДЕНИЕ	3
МАРШРУТ АВТОБУСОВ И УСЛОВИЯ РАБОТЫ ВОДИТЕЛЕЙ	4
Маршрут автобусов.....	4
Распределение водителей.....	5
Автобусы	6
Заданные параметры для начального расчёта.....	6
ГЛАВА 1 НАИВНЫЙ СПОСОБ	7
ОПИСАНИЕ СТРУКТУРЫ И ЛОГИКИ РАБОТЫ ПРОГРАММЫ.....	7
Импорт библиотек и определение вспомогательных классов.....	7
Инициализация маршрутов и автобусов, водителей	10
Создание расписания рейсов.....	12
Генерация расписания автобусов	15
Расписание для конкретного водителя	17
Пользовательское взаимодействие.....	18
ИТОГОВЫЙ ВЫВОД ПРОГРАММЫ.....	20
Вывод расписания для всех остановок	20
Вывод расписания для водителей разных типов	22
Проверка статуса водителей на заданное время	23
ГЛАВА 2 ГЕНЕТИЧЕСКИЙ АЛГОРИТМ.....	24
ОПИСАНИЕ СТРУКТУРЫ И ЛОГИКИ РАБОТЫ ПРОГРАММЫ.....	24
Инициализация данных	24
Планировщик рейсов и работа с расписанием.....	26
Генетический алгоритм для оптимизации.....	27
Формирование итогового расписания.....	33
ИТОГОВЫЙ ВЫВОД ПРОГРАММЫ.....	36
ОБЩИЙ ВЫВОД И СРАВНЕНИЕ ОПТИМАЛЬНОСТИ.....	38

ВВЕДЕНИЕ

В современном мире оптимизация транспортных потоков является актуальной задачей, решение которой позволяет повысить эффективность использования ресурсов и улучшить качество жизни населения. В рамках данной работы я рассматриваю проблему оптимизации расписания движения автобусов.

Проблема заключается в том, что существующее расписание автобусов не всегда удовлетворяет потребности людей, что приводит к неудобствам и потерям времени. Например, студенты могут опаздывать на занятия из-за того, что автобус приходит слишком поздно или слишком рано. Также пассажиры вынуждены тратить время на ожидание автобуса на остановках, что также снижает эффективность использования времени.

Цель работы: целью данного исследования является создание расписания движения автобусов для маршрута, включающего восемь остановок, с учётом нескольких факторов, таких как типы водителей, время работы, перерывы, а также оптимизация расписания с использованием двух методов: прямого наивного поиска оптимальности и генетического алгоритма.

Задачи:

1. Разработка модели маршрута и факторов расписания. Определить основные параметры, влияющие на создание расписания: типы водителей, их рабочие смены, перерывы, время в пути и интервалы рейсов.
2. Реализация алгоритмов оптимизации расписания. Разработать и реализовать два подхода к оптимизации расписания: прямой наивный поиск и генетический алгоритм для более эффективного поиска оптимального решения.
3. Сравнение и анализ методов оптимизации

МАРШРУТ АВТОБУСОВ И УСЛОВИЯ РАБОТЫ ВОДИТЕЛЕЙ

Маршрут автобусов

Маршрут состоит из четырёх остановок, расположенных по определённой последовательности:

1. Улица маршала Тухачевского
2. Университет связи и информатики
3. Метро Октябрьское поле
4. МЦК Панфиловская

Для этого маршрута определены два направления:

- Туда: Улица маршала Тухачевского → Университет связи и информатики → Метро Октябрьское поле → МЦК Панфиловская
- Обратно: МЦК Панфиловская → Метро Октябрьское поле → Университет связи и информатики → Улица маршала Тухачевского

Автобус отходит от остановки Улица Маршала Тухачевского каждые 20 минут. Также на протяжении всего маршрута автобус проходит от одной остановки до другой за 20 минут.

Рабочий день маршрута ограничен интервалом с 06:00 до 03:00. Система учитывает следующие особенности работы: пересменки водителей, перерывы и необходимость своевременной смены автобусов.

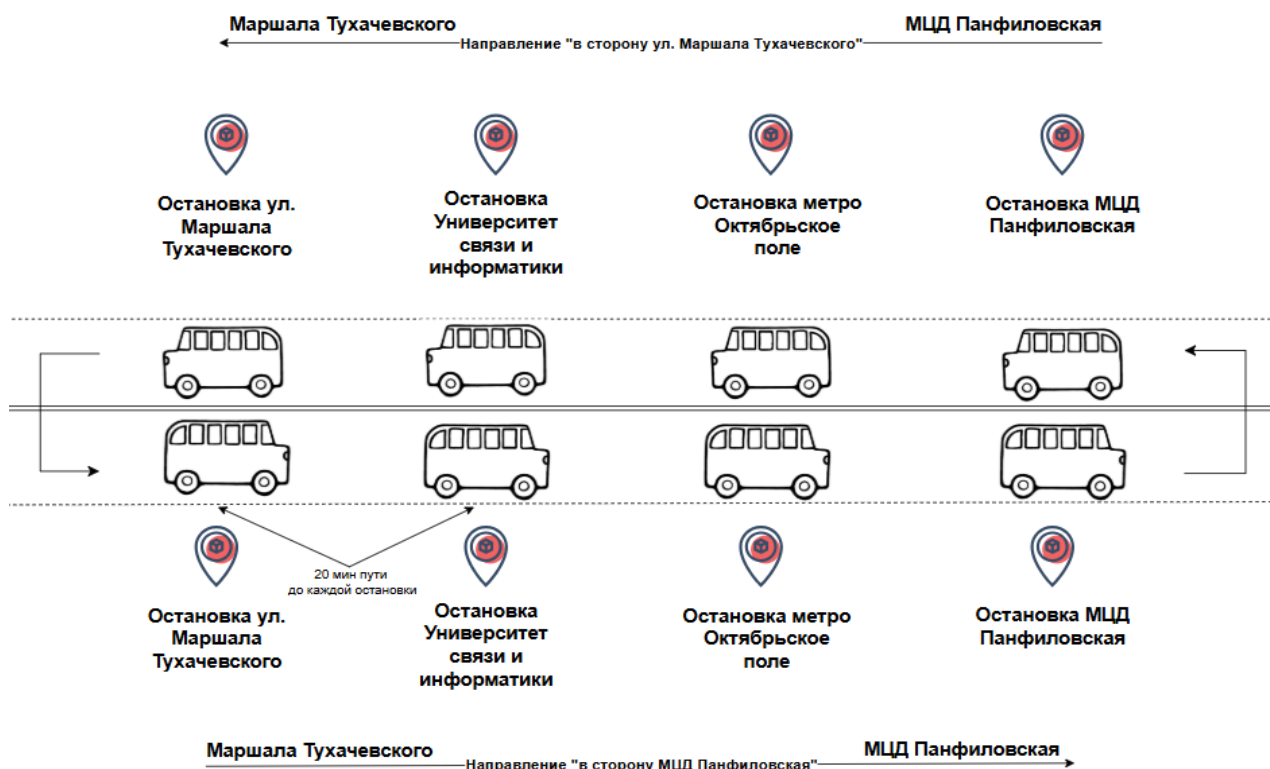


Рисунок 1 – схема движения транспорта

Распределение водителей

Водители распределены на два типа с разными режимами работы:

Водители 1-го типа:

- Работают только в будние дни (понедельник — пятница).
- Рабочее время ограничено 8 часами, не включая 1 час обеда, который должен быть в середине рабочей смены (после первых 4 часов работы).

Водители 2-го типа:

- Работают 12 часов подряд.
- Следуют графику "два дня работы, один день отдыха".
- В течение смены делают 10-минутные перерывы каждые 2 часа.

Также учитывается условие, что все водители не могут одновременно уйти на перерыв.

Автобусы

Автобусы имеют уникальные номера (например, A001AA, A002AA и так далее до A00NAA, где N — количество автобусов). Каждый автобус работает на маршруте до конца своей смены. В случае необходимости (например, при пересменке или перерыве) водитель может сменить автобус на свободный.

Заданные параметры для начального расчёта

Для проведения расчёта исходных данных заданы следующие параметры:

Количество автобусов: 25

Количество водителей: 30

- 15 водителей 1-го типа (работают 8 часов в сутки, только в будни, с обедом).
- 15 водителей 2-го типа (работают 12 часов в сутки, с перерывами).

Продолжительность маршрута в одну сторону: 60 минут.

Интервал между автобусами:

- В часы пик (7:00–9:00, 18:00–20:00): интервал 10 минут.
- В остальное время: интервал 20 минут.

ГЛАВА 1 НАИВНЫЙ СПОСОБ

ОПИСАНИЕ СТРУКТУРЫ И ЛОГИКИ РАБОТЫ ПРОГРАММЫ Импорт библиотек и определение вспомогательных классов

Подключаю модуль `datetime` для работы с датами и временем. Также создаю базовые классы `Stop`, `Bus`, `Driver`, `Route`, `Trip`.

1. Class `Stop` для представления остановки, из атрибутов есть только `name`: str название остановки.

2. Class `Bus` для представления автобуса.

Атрибуты:

- `bus_number`: str Номер автобуса,
- `current_trip`: `Trip` или `None` текущий выполняемый рейс автобуса.

3. Class `Driver` для представления водителя.

Атрибуты:

- `name`: str имя водителя.
- `driver_type` : int тип водителя (например, 1 или 2).
- `bus`: `Bus` привязанный автобус.
- `shift_start`: `datetime` начало смены водителя.
- `shift_end`: `datetime` конец смены водителя.
- `breaks`: list список перерывов [(start, end), ...].
- `trips`: list список выполненных рейсов.
- `loop_count`: int счётчик циклов рейсов для водителя.

Метод `is_available` проверяет, доступен ли водитель для указанного рейса.

Параметры:

- `trip_start`: `datetime` время начала рейса.
- `trip_duration`: `timedelta` продолжительность рейса.

Возвращает: bool: True, если водитель доступен иначе False.

4. Class Route описывает маршрут, по которому следует автобус, включая список остановок и направление.

- stops: list список остановок маршрута.
- Direction: str направление движения маршрута (например, "в сторону МЦД Панфиловская").
- self.duration: timedelta длительность поездки по маршруту. По умолчанию 60 минут.

5. Class Trip представляет конкретный рейс автобуса, включая информацию о времени отправления, автобусе, водителе, маршруте и времени прибытия на остановки.

- departure_time: datetime время отправления рейса.
- bus: str объект автобуса, который выполняет рейс.
- driver: str объект водителя, закрепленного за рейсом.
- route: str объект маршрута, по которому следует рейс.
- self.arrival_times: list список времени прибытия на каждую остановку маршрута. Изначально пустой, наполняется позже


```

from datetime import datetime, timedelta

class Stop:
    def __init__(self, name):
        self.name = name

class Bus:
    def __init__(self, bus_number):
        self.bus_number = bus_number
        self.current_trip = None

class Driver:
    def __init__(self, name, driver_type, bus, shift_start, shift_end, breaks):
        self.name = name
        self.driver_type = driver_type
        self.bus = bus
        self.shift_start = shift_start
        self.shift_end = shift_end
        self.breaks = breaks
        self.trips = []
        self.loop_count = 0

    def is_available(self, trip_start, trip_duration):
        trip_end = trip_start + trip_duration
        #проверка на рабочее время
        if trip_start < self.shift_start or trip_end > self.shift_end:
            return False
        #проверка на пересечение с перерывами
        for break_start, break_end in self.breaks:
            if trip_start < break_end and trip_end > break_start:
                return False
        #проверка на пересечение с другими рейсами
        for trip in self.trips:
            if trip_start < trip.departure_time + trip.route.duration and trip_end > trip.departure_time:
                return False
        #проверка занятости автобуса
        if self.bus.current_trip:
            bus_trip_end = self.bus.current_trip.departure_time + self.bus.current_trip.route.duration
            if trip_start < bus_trip_end:
                return False
        return True

class Route:
    def __init__(self, stops, direction):
        self.stops = stops
        self.direction = direction
        self.duration = timedelta(minutes=60)

class Trip:
    def __init__(self, departure_time, bus, driver, route):
        self.departure_time = departure_time
        self.bus = bus
        self.driver = driver
        self.route = route
        self.arrival_times = []

```

Рисунок 2 – фрагмент кода, где определены базовые классы

Также есть еще один важный класс Schedule для управления расписанием и применяется он для хранения списка всех рейсов.

- Метод `add_trip` добавляет новый рейс.
- Метод `generate_arrival_times` вычисляет время прибытия автобусов на остановки для каждого рейса (это всегда 20 мин)

```

class Schedule:
    def __init__(self):
        self.trips = []

    def add_trip(self, trip):
        self.trips.append(trip)

    def generate_arrival_times(self):
        for trip in self.trips:
            current_time = trip.departure_time
            for stop in trip.route.stops:
                trip.arrival_times.append((stop.name, current_time))
                current_time += timedelta(minutes=20)

```

Рисунок 3 – класс Schedule

Инициализация маршрутов и автобусов, водителей

```

stop_names = [
    "Улица маршала Тухачевского",
    "Университет связи и информатики",
    "Метро Октябрьское поле",
    "МЦК Панфиловская"
]
stops = [Stop(name) for name in stop_names]

route_there = Route(stops, "в сторону МЦД Панфиловская")
route_back = Route(stops[::-1], "в сторону улицы Маршала Тухачевского")

buses = [Bus(f"A00{i}AA") for i in range(1, 26)]

```

Рисунок 4 – инициализирую остановки, маршрут и автобусы

Определяю остановки маршрутов, указанные выше. Также создаю два маршрута: в одном и обратном направлениях. Генерирую список автобусов с уникальными номерами.

DRIVERS_TYPE1 и DRIVERS_TYPE2 — списки имен водителей.

- DRIVERS_TYPE1: Водители первого типа с фиксированными сменами, имена начинаются на «П» для удобства при отслеживании работы программы (8 часов).
- DRIVERS_TYPE2: Водители второго типа с гибкими сменами (12 часов) и более частыми перерывами, имена начинаются на «В»

start_date: Начало расписания — 1 октября 2024 года, 6:00 утра.

end_date: Конец расписания — 2 октября 2024 года, 3:00 ночи

Создается список `shift_start_times`, содержащий возможные времена начала смен для водителей. Смены стартуют каждый час, начиная с 6:00 утра первого дня и заканчивая 03:00 ночи второго дня.

Каждый водитель из `DRIVERS_TYPE1` имеет:

- Смену продолжительностью 8 часов, начинающуюся с 6:00.
- Один перерыв, начинающийся через 4 часа после начала смены и длительностью 1 час.
- Водитель закрепляется за автобусом из списка `buses` (номер автобуса выбирается циклически).
- Экземпляры класса `Driver` создаются и добавляются в список `drivers`.

Аналогичным образом, поступаю с `DRIVERS_TYPE2`:

- Смены для каждого водителя начинаются согласно `shift_start_times`.
- Продолжительность смены — 12 часов.
- Перерывы добавляются каждые 2 часа 10 минут и длятся 10 минут
- Водитель также закрепляется за автобусом из `buses`.
- Водители добавляются в список `drivers`.

Список `all_drivers` создается как копия `drivers`. Это необходимо для дальнейшего использования без изменений исходного списка.

```

DRIVERS_TYPE1 = [
    "Павел", "Петр", "Панкрат", "Прокл", "Платон",
    "Полина", "Порфирий", "Пересвет", "Пимен", "Прохор",
    "Потап", "Пиона", "Протас", "Президий", "Преслав"
]

DRIVERS_TYPE2 = [
    "Вадим", "Валерий", "Василий", "Виктор", "Владимир",
    "Владислав", "Вячеслав", "Виталий", "Валентин", "Венедикт",
    "Велор", "Вилен", "Витольд", "Валдар", "Варлам",
]

start_date = datetime.strptime("2024-10-01 06:00", "%Y-%m-%d %H:%M")
end_date = datetime.strptime("2024-10-02 03:00", "%Y-%m-%d %H:%M")

shift_start_times = [start_date + timedelta(hours=i) for i in range(int((end_date - start_date).total_seconds() / (60*60)))]

drivers = []

for i, name in enumerate(DRIVERS_TYPE1):
    shift_start = start_date
    shift_end = shift_start + timedelta(hours=8)
    break_start = shift_start + timedelta(hours=4)
    break_end = break_start + timedelta(hours=1)
    breaks = [(break_start, break_end)]
    drivers.append(Driver(name, 1, buses[i % len(buses)], shift_start, shift_end, breaks))

for i, name in enumerate(DRIVERS_TYPE2):
    shift_start = shift_start_times[i % len(shift_start_times)]
    shift_end = shift_start + timedelta(hours=12)
    breaks = []
    break_time = shift_start + timedelta(hours=2, minutes=10)
    while break_time < shift_end:
        break_start = break_time
        break_end = break_start + timedelta(minutes=10)
        breaks.append((break_start, break_end))
        break_time += timedelta(hours=2, minutes=10)
    drivers.append(Driver(name, 2, buses[i % len(buses)], shift_start, shift_end, breaks))

all_drivers = drivers.copy()

```

Рисунок 5 – создание списка водителей с учетом фиксированных и гибких смен, а также перерывов

Создание расписания рейсов

Создаю переменные `start_time` и `end_time`, которые указывают начало и конец временного интервала для генерации расписания.

Функция `is_peak_time` определяет, относится ли заданное время к периоду пиковой нагрузки:

- Утренний пик: с 7:00 до 9:00.
- Вечерний пик: с 17:00 до 19:00.

```

start_time = datetime.strptime("2024-10-01 06:00", "%Y-%m-%d %H:%M")
end_time = datetime.strptime("2024-10-02 03:00", "%Y-%m-%d %H:%M")

schedule = Schedule()

def is_peak_time(time):
    return (7 <= time.hour < 9) or (17 <= time.hour < 19)

```

Рисунок 6 – начало и конец работы транспорта и функция для определения пикового времени

generate_trips основная функция, отвечающая за генерацию поездок в расписании.

Параметры

- schedule: Экземпляр класса Schedule, в который добавляются поездки.
- start_time, end_time: Временной интервал, в течение которого формируется расписание.
- available_drivers: Список доступных водителей с их свойствами и состоянием.

Логика работы функции

1. Итерация по времени:

```

def generate_trips(schedule, start_time, end_time, available_drivers):
    current_time = start_time
    while current_time < end_time:

```

Рисунок 7 - В цикле обрабатывается каждое временное окно, начиная с start_time и заканчивая end_time.

2.Интервал между поездками:

```

while current_time < end_time:
    interval = timedelta(minutes=10 if is_peak_time(current_time) else 20)

```

Рисунок 8 - Интервал между поездками 10 минут в пиковое время и 20 минут в непиковое время.

3. Проверка доступности водителей:

```
for driver in available_drivers:
    on_break = any(
        break_start <= current_time < break_end
        for break_start, break_end in driver.breaks
    )
```

Рисунок 9 - Для каждого водителя проверяется, находится ли он в это время на перерыве (перерыв задается в виде списка интервалов)

4. Создание поездок:

"Туда":

```
if not on_break and driver.is_available(current_time, route_there.duration + route_back.duration):
    selected_bus = driver.bus

    trip_there = Trip(current_time, selected_bus, driver, route_there)
    schedule.add_trip(trip_there)
    driver.trips.append(trip_there)
```

Рисунок 10 - если водитель доступен, создаются объекты поездок

Поездка на маршрут `route_there` с фиксированной длительностью.

"Обратно":

```
trip_back_departure = current_time + route_there.duration
trip_back = Trip(trip_back_departure, selected_bus, driver, route_back)
schedule.add_trip(trip_back)
driver.trips.append(trip_back)

selected_bus.current_trip = trip_back
break
```

Рисунок 11 - Возвратная поездка на маршрут `route_back` и переход ко времени следующей поездки

5. Генерация расписания

```
print(f"Генерация расписания на {start_date.strftime('%Y-%m-%d')} ({start_date.strftime('%A')})")
generate_trips(schedule, start_time, end_time, all_drivers.copy())
schedule.generate_arrival_times()
```

Рисунок 12 - Формируется расписание на конкретный день с учетом всех водителей `all_drivers` и расписания их перерывов.

```
def generate_trips(schedule, start_time, end_time, available_drivers):
    current_time = start_time
    while current_time < end_time:
        interval = timedelta(minutes=10 if is_peak_time(current_time) else 20)
        for driver in available_drivers:
            on_break = any(
                break_start <= current_time < break_end
                for break_start, break_end in driver.breaks
            )

            if not on_break and driver.is_available(current_time, route_there.duration + route_back.duration):
                selected_bus = driver.bus

                trip_there = Trip(current_time, selected_bus, driver, route_there)
                schedule.add_trip(trip_there)
                driver.trips.append(trip_there)

                trip_back_departure = current_time + route_there.duration
                trip_back = Trip(trip_back_departure, selected_bus, driver, route_back)
                schedule.add_trip(trip_back)
                driver.trips.append(trip_back)

                selected_bus.current_trip = trip_back
                break

            current_time += interval

print(f"Генерация расписания на {start_date.strftime('%Y-%m-%d')} ({start_date.strftime('%A')})")
generate_trips(schedule, start_time, end_time, all_drivers.copy())
schedule.generate_arrival_times()
```

Рисунок 13 – полный код функции для генерации расписания

Генерация расписания автобусов

Создается словарь `stop_schedules`, где:

Ключи верхнего уровня — это названия остановок (`stop.name`).

Значения — это вложенные словари с двумя ключами: "в сторону МЦД Панфиловская" — для хранения рейсов в этом направлении; "в сторону улицы

Маршала Тухачевского" — для хранения рейсов в противоположном направлении. Каждый ключ направления содержит список, куда будут добавляться записи о рейсах.

Логика заполнения расписания:

1. Перебор всех рейсов: `schedule.trips` — список всех поездок, сгенерированных ранее.
2. Обработка остановок и времени прибытия: `trip.arrival_times` — список пар (`stop_name`, `arrival_time`), где `stop_name` — название остановки, а `arrival_time` — время прибытия автобуса на эту остановку.
3. Добавление записи: для каждой остановки информация о рейсе добавляется в список соответствующего направления:

— "Время прибытия": фиксируется время прибытия автобуса.

— "Водитель": указывается имя водителя (`trip.driver.name`).

— "Номер рейса": указывается номер автобуса (`trip.bus.bus_number`).

— "Направление": указывается направление маршрута (`trip.route.direction`)

```
for trip in schedule.trips:
    for stop_name, arrival_time in trip.arrival_times:
        direction = trip.route.direction
        stop_schedules[stop_name][direction].append({
            "Время прибытия": arrival_time,
            "Водитель": trip.driver.name,
            "Номер рейса": trip.bus.bus_number,
            "Направление": direction
        })

for stop_name, directions in stop_schedules.items():
    print(f"Остановка: {stop_name}")
    for direction in ["в сторону МЦД Панфиловская", "в сторону улицы Маршала Тухачевского"]:
        print(f" {direction}:")
        sorted_entries = sorted(directions[direction], key=lambda x: x["Время прибытия"])
        for entry in sorted_entries:
            print(f"    Время: {entry['Время прибытия'].strftime('%H:%M')}, Водитель: {entry['Водитель']}, Номер рейса: {entry['Номер рейса']}")

print(f"Всего сгенерированных рейсов: {len(schedule.trips)}")
```

Рисунок 14 – Заполнение расписания и его вывод с сортировкой записей для каждой остановки, а также выводится количество всех рейсов за день

Расписание для конкретного водителя

Функция `print_driver_schedule` выводит детальное расписание для указанного водителя, включая:

- Начало и конец смены.
- Перерывы.
- Выезды на маршруты.

Аргументом является `driver`: объект водителя, содержащий информацию о сменах, перерывах и рейсах.

Логика работы заключается в том, что создается список событий, куда добавляются:

- Смена (начало и конец).
- Перерывы (начало и конец).
- Рейсы (время выезда, направление, номер автобуса).

Затем происходит сортировка списка событий по времени, и дальше я вывожу полное расписание.

```
def print_driver_schedule(driver):
    events = []
    #начало и конец смены
    events.append({'time': driver.shift_start, 'description': 'Начало смены'})
    events.append({'time': driver.shift_end, 'description': 'Конец смены'})
    #все перерывы
    for break_start, break_end in driver.breaks:
        events.append({'time': break_start, 'description': 'Начало перерыва'})
        events.append({'time': break_end, 'description': 'Конец перерыва'})
    #все выезды
    for trip in driver.trips:
        events.append({
            'time': trip.departure_time,
            'description': f'Выезд в "{trip.route.direction}" на автобусе {trip.bus.bus_number}'
        })
    events.sort(key=lambda x: x['time'])

    print(f"Расписание для водителя {driver.name}:")
    for event in events:
        print(f"    {event['time'].strftime('%Y-%m-%d %H:%M')}: {event['description']}")
```

Рисунок 15 – функция для вывода расписания водителей

Отображение статуса водителей в заданное время

Функция `print_driver_status_at_hour` проверяет и выводит статус каждого водителя на заданное время:

- На работе (в рабочее время, не в перерыве).
- В перерыве.
- Не на смене.

Аргументы:

- `target_time`: Время, для которого проверяется статус.
- `drivers`: Список объектов водителей.

```
def print_driver_status_at_hour(target_time, drivers):
    working_drivers = []
    on_break_drivers = []
    off_duty_drivers = []

    for driver in drivers:
        if driver.shift_start <= target_time < driver.shift_end:
            on_break = any(break_start <= target_time < break_end for break_start, break_end in driver.breaks)
            if on_break:
                on_break_drivers.append(driver.name)
            else:
                working_drivers.append(driver.name)
        else:
            off_duty_drivers.append(driver.name)

    print(f"Статус на {target_time.strftime('%Y-%m-%d %H:%M')}: \n")
    print("Водители на работе:")
    for name in working_drivers:
        print(f" - {name}")
    print("\nВодители в перерыве:")
    for name in on_break_drivers:
        print(f" - {name}")
    print("\nВодители не на смене:")
    for name in off_duty_drivers:
        print(f" - {name}")
```

Рисунок 16 - функция `print_driver_status_at_hour`

Пользовательское взаимодействие

Создается список имен водителей и словарь для быстрого поиска водителя по имени.

Пользователю предлагается выбрать водителя по имени. Если введено некорректное имя, появляется уведомление.

```

driver_names = [driver.name for driver in all_drivers]
driver_dict = {driver.name: driver for driver in all_drivers}

while True:
    print("Список водителей:")
    print(", ".join(driver_names))
    input_name = input("Введите имя водителя: ").strip()

    if input_name in driver_dict:
        selected_driver = driver_dict[input_name]
        break
    else:
        print("Водитель с таким именем не найден. Пожалуйста, введите корректное имя.")

print_driver_schedule(selected_driver)

```

Рисунок 17 – выбор водителя

Пользователь может дополнительно запросить статус водителей на конкретное время:

```

print("\nХотите увидеть статус водителей в конкретное время? (да/нет)")
choice = input().strip().lower()

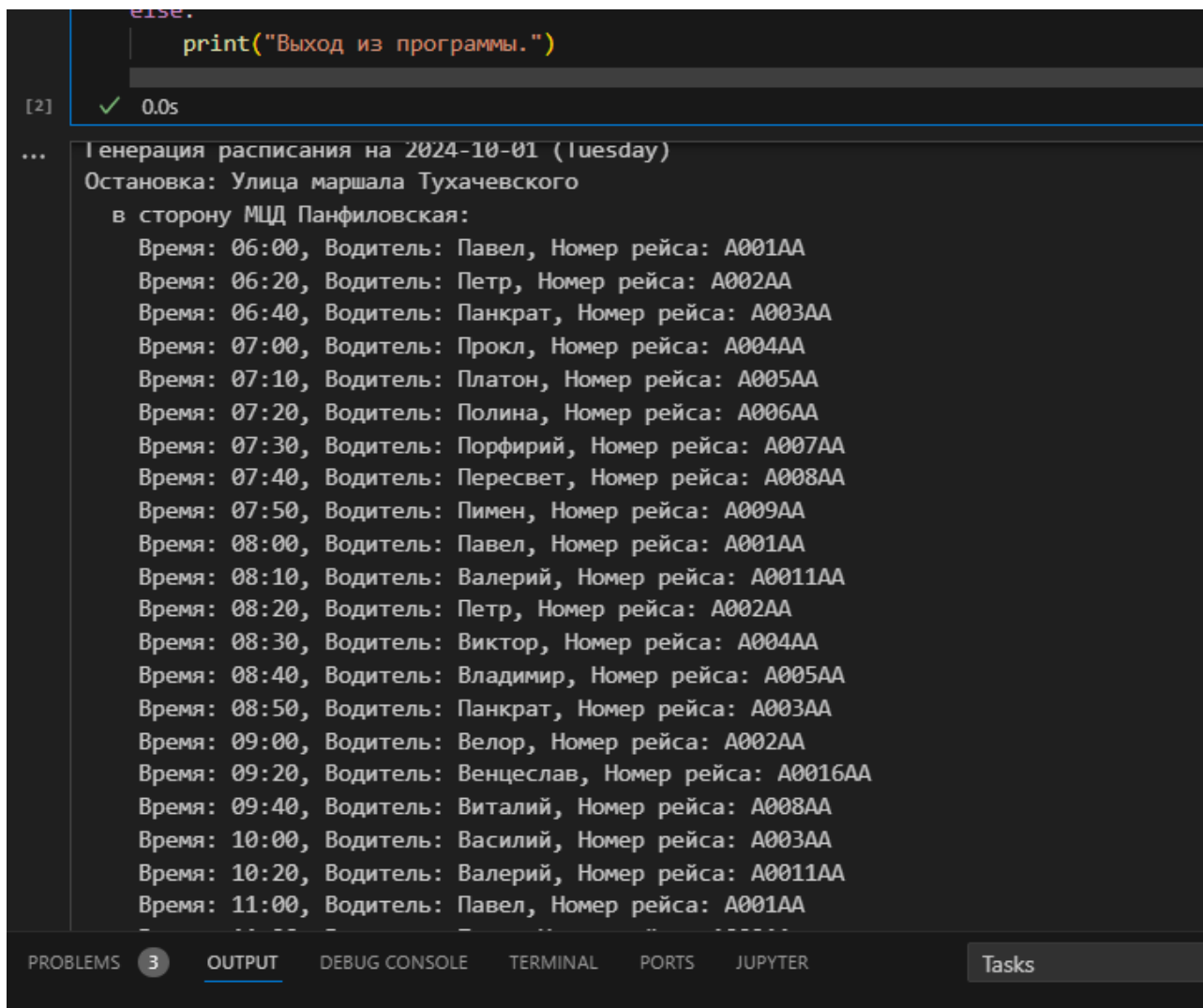
if choice == 'да':
    while True:
        time_input = input("Введите время в формате 'ГГГГ-ММ-ДД ЧЧ:ММ': ")
        try:
            target_time = datetime.strptime(time_input, "%Y-%m-%d %H:%M")
            break
        except ValueError:
            print("Неверный формат времени. Пожалуйста, попробуйте снова.")
    print_driver_status_at_hour(target_time, all_drivers)
else:
    print("Выход из программы.")

```

Рисунок 18 - проверка статуса водителей на заданное время

ИТОГОВЫЙ ВЫВОД ПРОГРАММЫ

Вывод расписания для всех остановок



```
else:  
    print("Выход из программы.")
```

[2] ✓ 0.0s

... Генерация расписания на 2024-10-01 (Tuesday)
Остановка: Улица маршала Тухачевского
в сторону МЦД Панфиловская:
Время: 06:00, Водитель: Павел, Номер рейса: A001AA
Время: 06:20, Водитель: Петр, Номер рейса: A002AA
Время: 06:40, Водитель: Панкрат, Номер рейса: A003AA
Время: 07:00, Водитель: Прокл, Номер рейса: A004AA
Время: 07:10, Водитель: Платон, Номер рейса: A005AA
Время: 07:20, Водитель: Полина, Номер рейса: A006AA
Время: 07:30, Водитель: Порфирий, Номер рейса: A007AA
Время: 07:40, Водитель: Пересвет, Номер рейса: A008AA
Время: 07:50, Водитель: Пимен, Номер рейса: A009AA
Время: 08:00, Водитель: Павел, Номер рейса: A001AA
Время: 08:10, Водитель: Валерий, Номер рейса: A0011AA
Время: 08:20, Водитель: Петр, Номер рейса: A002AA
Время: 08:30, Водитель: Виктор, Номер рейса: A004AA
Время: 08:40, Водитель: Владимир, Номер рейса: A005AA
Время: 08:50, Водитель: Панкрат, Номер рейса: A003AA
Время: 09:00, Водитель: Велор, Номер рейса: A002AA
Время: 09:20, Водитель: Венцеслав, Номер рейса: A0016AA
Время: 09:40, Водитель: Виталий, Номер рейса: A008AA
Время: 10:00, Водитель: Василий, Номер рейса: A003AA
Время: 10:20, Водитель: Валерий, Номер рейса: A0011AA
Время: 11:00, Водитель: Павел, Номер рейса: A001AA

PROBLEMS 3 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER Tasks

Рисунок 19 – вывод расписания

Ввиду того, что вывод массивный, я решила перенести данные в Excel таблицу для удобной визуализации расписания каждой остановки. Желтым цветом я отметила часы пик в течение дня. При изменении даты, вывод отличается, в выходные дни бывают интервалы размером в час, особенно в час пик, что не совсем хорошо, чтобы избежать этих проблем существует генетический алгоритм, который рассмотрим дальше. В данном примере на рис 19, большие окна появляются только вечером около 23.00, так как новые смены водителей не назначаются так поздно.

Остановка	Направление	Время	Водитель	Номер рейса
Остановка Улица маршала Тухачевского	направление МЦД Панфиловская	06:00	Павел	A001AA
		06:20	Петр	A002AA
		06:40	Павел	A003AA
		07:00	Прокл	A004AA
		07:10	Платон	A005AA
		07:20	Полина	A006AA
		07:30	Порфирий	A007AA
		07:50	Пересвет	A008AA
		08:00	Павел	A009AA
		08:10	Платон	A001AA
		08:20	Виктор	A002AA
		08:30	Виктор	A004AA
		08:40	Владимир	A005AA
		08:50	Павел	A003AA
		09:00	Виктор	A002AA
		09:20	Венцеслав	A0016AA
		09:40	Виталий	A008AA
		09:50	Василий	A003AA
		10:00	Василий	A003AA
		10:20	Василий	A0011AA
		11:00	Павел	A001AA
		11:20	Петр	A002AA
		11:40	Платон	A005AA
		12:00	Павел	A003AA
		12:20	Викон	A0012AA
		12:40	Василий	A0011AA
		13:00	Павел	A001AA
		13:20	Петр	A002AA
		13:40	Василий	A003AA
		14:00	Платон	A005AA
		14:20	Виктор	A004AA
		14:40	Василий	A0011AA
		15:00	Венедикт	A0010AA
		15:20	Виталий	A008AA
		15:40	Венцеслав	A007AA
		16:00	Владислав	A006AA
		16:20	Владимир	A005AA
		16:40	Василий	A003AA
		17:00	Василий	A003AA
		17:20	Василий	A0011AA
		17:40	Венедикт	A002AA
		18:00	Платон	A005AA
		18:20	Венцеслав	A007AA
		18:40	Венцеслав	A0016AA
		18:50	Владимир	A005AA
		19:00	Виктор	A004AA
		19:20	Василий	A003AA
		19:40	Платон	A005AA
		20:00	Василий	A003AA
		20:10	Виталий	A008AA
		20:20	Венцеслав	A007AA
		20:40	Владимир	A005AA
		21:00	Полина	A006AA
		21:20	Венедикт	A0010AA
		21:40	Василий	A003AA
		22:00	Валентин	A009AA
		22:20	Виталий	A008AA
		22:40	Владислав	A006AA
		00:00	Валентин	A009AA
		00:20	Венцеслав	A007AA
Остановка Университет связи и информатики	направление МЦД Панфиловская	06:20	Павел	A001AA
		06:40	Петр	A002AA
		07:00	Павел	A003AA
		07:20	Прокл	A004AA
		07:30	Платон	A005AA
		07:40	Полина	A006AA
		07:50	Порфирий	A007AA
		08:00	Пересвет	A008AA
		08:10	Павел	A009AA
		08:20	Василий	A0011AA
		08:30	Василий	A003AA
		08:40	Петр	A002AA
		08:50	Виктор	A004AA
		09:00	Владимир	A005AA
		09:10	Павел	A003AA
		09:20	Венцеслав	A0016AA
		09:40	Виталий	A008AA
		10:00	Василий	A003AA
		10:20	Василий	A003AA
		10:40	Василий	A0011AA
		11:00	Павел	A001AA
		11:20	Петр	A002AA
		11:40	Платон	A005AA
		12:00	Павел	A003AA
		12:20	Викон	A0012AA
		12:40	Василий	A0011AA
		13:00	Павел	A001AA
		13:20	Петр	A002AA
		13:40	Василий	A003AA
		14:00	Платон	A005AA
		14:20	Виктор	A004AA
		14:40	Василий	A0011AA
		15:00	Венедикт	A0010AA
		15:20	Виталий	A008AA
		15:40	Венцеслав	A007AA
		16:00	Владислав	A006AA
		16:20	Владимир	A005AA
		16:40	Василий	A003AA
		17:00	Василий	A003AA
		17:20	Василий	A0011AA
		17:40	Венедикт	A002AA
		18:00	Платон	A005AA
		18:20	Венцеслав	A007AA
		18:40	Венцеслав	A0016AA
		18:50	Владимир	A005AA
		19:00	Виктор	A004AA
		19:20	Василий	A003AA
		19:40	Платон	A005AA
		20:00	Василий	A003AA
		20:10	Виталий	A008AA
		20:20	Венцеслав	A007AA
		20:40	Владимир	A005AA
		21:00	Полина	A006AA
		21:20	Венедикт	A0010AA
		21:40	Василий	A003AA
		22:00	Валентин	A009AA
		22:20	Виталий	A008AA
		22:40	Владислав	A006AA
		00:00	Валентин	A009AA
		00:20	Венцеслав	A007AA
Остановка метро Октябрьское поле	направление МЦД Панфиловская	06:40	Павел	A001AA
		07:00	Петр	A002AA
		07:20	Павел	A003AA
		07:40	Прокл	A004AA
		07:50	Платон	A005AA
		08:00	Полина	A006AA
		08:10	Порфирий	A007AA
		08:20	Пересвет	A008AA
		08:30	Павел	A009AA
		08:40	Василий	A0011AA
		08:50	Василий	A003AA
		09:00	Петр	A002AA
		09:10	Виктор	A004AA
		09:20	Владимир	A005AA
		09:30	Павел	A003AA
		09:40	Венцеслав	A0016AA
		09:50	Виталий	A008AA
		10:00	Василий	A003AA
		10:20	Василий	A003AA
		10:40	Василий	A0011AA
		11:00	Павел	A001AA
		11:20	Петр	A002AA
		11:40	Платон	A005AA
		12:00	Павел	A003AA
		12:20	Викон	A0012AA
		12:40	Василий	A0011AA
		13:00	Павел	A001AA
		13:20	Петр	A002AA
		13:40	Василий	A003AA
		14:00	Платон	A005AA
		14:20	Виктор	A004AA
		14:40	Василий	A0011AA
		15:00	Венедикт	A0010AA
		15:20	Виталий	A008AA
		15:40	Венцеслав	A007AA
		16:00	Владислав	A006AA
		16:20	Владимир	A005AA
		16:40	Василий	A003AA
		17:00	Василий	A003AA
		17:20	Василий	A0011AA
		17:40	Венедикт	A002AA
		18:00	Платон	A005AA
		18:20	Венцеслав	A007AA
		18:40	Венцеслав	A0016AA
		18:50	Владимир	A005AA
		19:00	Виктор	A004AA
		19:20	Василий	A003AA
		19:40	Платон	A005AA
		20:00	Василий	A003AA
		20:10	Виталий	A008AA
		20:20	Венцеслав	A007AA
		20:40	Владимир	A005AA
		21:00	Полина	A006AA
		21:20	Венедикт	A0010AA
		21:40	Василий	A003AA
		22:00	Валентин	A009AA
		22:20	Виталий	A008AA
		22:40	Владислав	A006AA
		00:00	Валентин	A009AA
		00:20	Венцеслав	A007AA
Остановка МЦК Панфиловская	направление МЦД Панфиловская	07:00	Павел	A001AA
		07:20	Петр	A002AA
		07:40	Павел	A003AA
		07:50	Прокл	A004AA
		08:00	Платон	A005AA
		08:10	Полина	A006AA
		08:20	Порфирий	A007AA
		08:30	Пересвет	A008AA
		08:40	Павел	A009AA
		08:50	Василий	A0011AA
		09:00	Василий	A003AA
		09:10	Петр	A002AA
		09:20	Виктор	A004AA
		09:30	Владимир	A005AA
		09:40	Павел	A003AA
		09:50	Венцеслав	A0016AA
		10:00	Виталий	A008AA
		10:10	Василий	A003AA
		10:20	Василий	A003AA
		10:40	Василий	A0011AA
		11:00	Павел	A001AA
		11:20	Петр	A002AA
		11:40	Платон	A005AA
		12:00	Павел	A003AA
		12:20	Викон	A0012AA
		12:40	Василий	A0011AA
		13:00	Павел	A001AA
		13:20	Петр	A002AA
		13:40	Василий	A003AA
		14:00	Платон	A005AA
		14:20	Виктор	A004AA
		14:40	Василий	A0011AA
		15:00	Венедикт	A0010AA
		15:20	Виталий	A008AA
		15:40	Венцеслав	A007AA
		16:00	Владислав	A006AA
		16:20	Владимир	A005AA
		16:40	Василий	A003AA
		17:00	Василий	A003AA
		17:20	Василий	A0011AA
		17:40	Венедикт	A002AA
		18:00	Платон	A005AA
		18:20	Венцеслав	A007AA
		18:40	Венцеслав	A0016AA
		18:50	Владимир	A005AA
		19:00	Виктор	A004AA
		19:20	Василий	A003AA
		19:40	Платон	A005AA
		20:00	Василий	A003AA
		20:10	Виталий	A008AA
		20:20	Венцеслав	A007AA
		20:40	Владимир	A005AA
		21:00	Полина	A006AA
		21:20	Венедикт	A0010AA
		21:40	Василий	A003AA
		22:00	Валентин	A009AA
		22:20	Виталий	A008AA
		22:40	Владислав	A006AA
		00:00	Валентин	A009AA
		00:20	Венцеслав	A007AA

Рисунок 20 – расписание остановок в направлении МЦД Панфиловская

Остановка	Направление	Время	Водитель	Номер рейса
Остановка Улица маршала Тухачевского	направление улицы м. Тухачевского	06:00	Павел	A001AA
		06:20	Петр	A002AA
		06:40	Павел	A003AA
		07:00	Прокл	A004AA
		07:10	Платон	A005AA
		07:20	Полина	A006AA</

Вывод расписания для водителей разных типов

Чтобы убедиться в корректности вывода расписания каждого водителя, я в Excel таблице отметила 2 водителей разных типов цветами, благодаря им можно удобно проследить маршрут каждого водителя и сопоставить с выводом программы. Для примера я выведу расписание двух водителей Павла и Василия.

```
else:
    print("Выход из программы.")
✓ 3.9s
```

Время: 01:40, Водитель: Витольд, Номер рейса: A0013AA
Время: 02:40, Водитель: Валдар, Номер рейса: A0014AA
Всего сгенерированных рейсов: 106
Расписание для водителя Павел:

2024-10-01 06:00: Начало смены
2024-10-01 06:00: Выезд в "в сторону МЦД Панфиловская" на автобусе A001AA
2024-10-01 07:00: Выезд в "в сторону улицы Маршала Тухачевского" на автобусе A001AA
2024-10-01 08:00: Выезд в "в сторону МЦД Панфиловская" на автобусе A001AA
2024-10-01 09:00: Выезд в "в сторону улицы Маршала Тухачевского" на автобусе A001AA
2024-10-01 10:00: Начало перерыва
2024-10-01 11:00: Конец перерыва
2024-10-01 11:00: Выезд в "в сторону МЦД Панфиловская" на автобусе A001AA
2024-10-01 12:00: Выезд в "в сторону улицы Маршала Тухачевского" на автобусе A001AA
2024-10-01 13:00: Выезд в "в сторону МЦД Панфиловская" на автобусе A001AA
2024-10-01 14:00: Выезд в "в сторону улицы Маршала Тухачевского" на автобусе A001AA
2024-10-01 15:00: Конец смены

Рисунок 22 – вывод расписания для водителя Павла (1 тип)

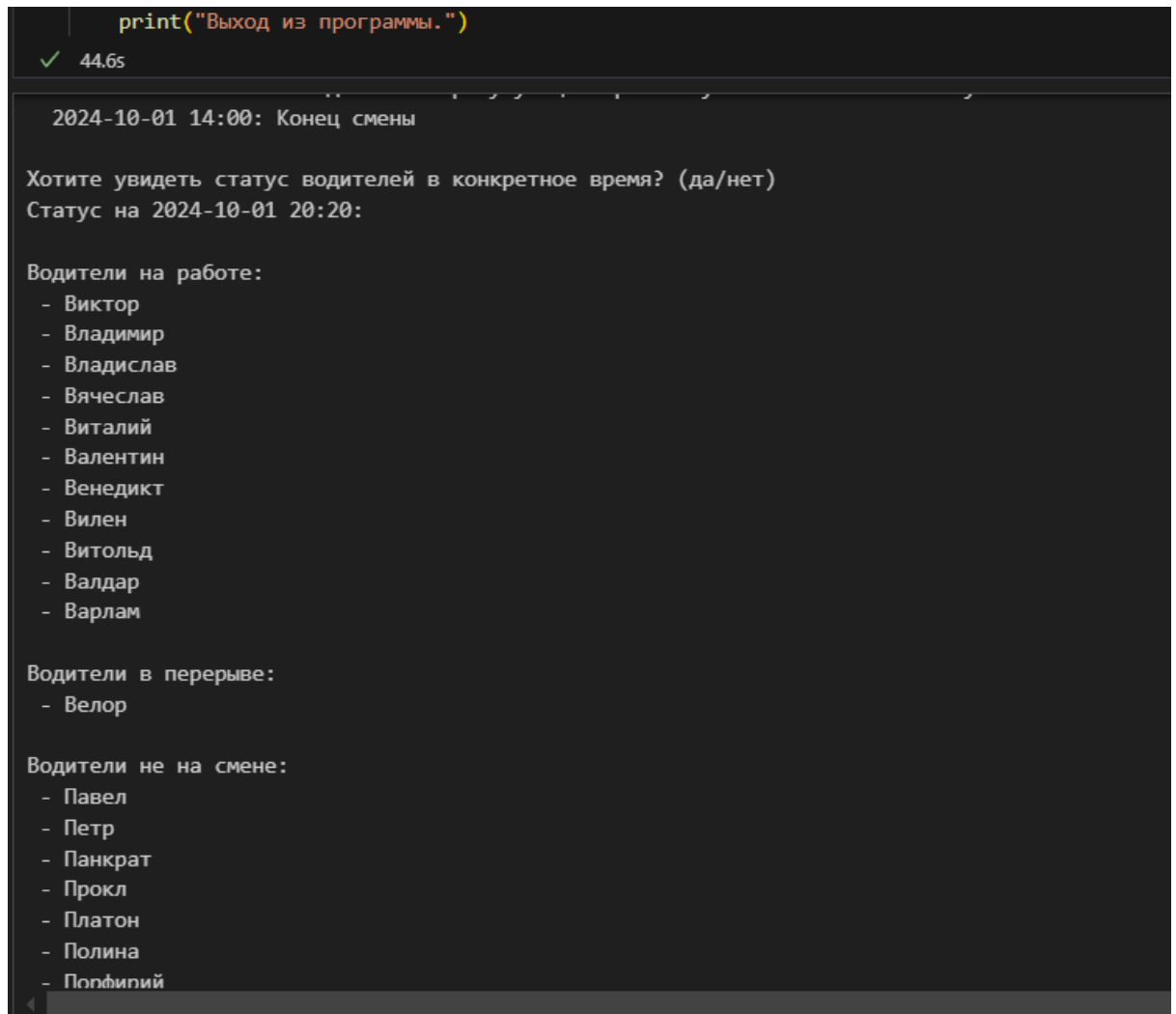
```
Расписание для водителя Василий:
```

2024-10-01 10:00: Начало смены
2024-10-01 10:00: Выезд в "в сторону МЦД Панфиловская" на автобусе A003AA
2024-10-01 11:00: Выезд в "в сторону улицы Маршала Тухачевского" на автобусе A003AA
2024-10-01 12:00: Начало перерыва
2024-10-01 12:10: Конец перерыва
2024-10-01 13:40: Выезд в "в сторону МЦД Панфиловская" на автобусе A003AA
2024-10-01 14:40: Выезд в "в сторону улицы Маршала Тухачевского" на автобусе A003AA
2024-10-01 15:40: Начало перерыва
2024-10-01 15:50: Конец перерыва
2024-10-01 16:40: Выезд в "в сторону МЦД Панфиловская" на автобусе A003AA
2024-10-01 17:40: Выезд в "в сторону улицы Маршала Тухачевского" на автобусе A003AA
2024-10-01 18:40: Начало перерыва
2024-10-01 18:50: Конец перерыва
2024-10-01 18:50: Выезд в "в сторону МЦД Панфиловская" на автобусе A003AA
2024-10-01 19:50: Выезд в "в сторону улицы Маршала Тухачевского" на автобусе A003AA
2024-10-01 20:50: Начало перерыва
2024-10-01 21:00: Конец перерыва
2024-10-01 21:40: Выезд в "в сторону МЦД Панфиловская" на автобусе A003AA
2024-10-01 22:40: Выезд в "в сторону улицы Маршала Тухачевского" на автобусе A003AA
2024-10-01 23:40: Конец смены

Рисунок 23 – вывод расписания для водителя Василий (2 тип)

Проверка статуса водителей на заданное время

Если пользователь выбирает "да", то потом он вводит время, после чего вызывается функция `print_driver_status_at_hour`, и в итоге мы видим список статусов водителей на конкретный момент времени.



```
print("Выход из программы.")
✓ 44.6s

2024-10-01 14:00: Конец смены

Хотите увидеть статус водителей в конкретное время? (да/нет)
Статус на 2024-10-01 20:20:

Водители на работе:
- Виктор
- Владимир
- Владислав
- Вячеслав
- Виталий
- Валентин
- Венедикт
- Вилен
- Витольд
- Валдар
- Варлам

Водители в перерыве:
- Велор

Водители не на смене:
- Павел
- Петр
- Панкрат
- Прокл
- Платон
- Полина
- Попфилий
```

Рисунок 24 - статус водителей на 01.10. 2024г на 20:20

ГЛАВА 2 ГЕНЕТИЧЕСКИЙ АЛГОРИТМ

ОПИСАНИЕ СТРУКТУРЫ И ЛОГИКИ РАБОТЫ ПРОГРАММЫ

Инициализация данных

За основу я взяла классы, которые определяла в наивном способе оптимизации расписания. Их описание и документацию можно найти на стр 7-10

```
import random

class Stop:
    def __init__(self, name):
        self.name = name

class Bus:
    def __init__(self, bus_number):
        self.bus_number = bus_number
        self.current_trip = None

class Driver:
    def __init__(self, name, driver_type, bus, shift_start, shift_end, breaks):
        self.name = name
        self.driver_type = driver_type
        self.bus = bus
        self.shift_start = shift_start
        self.shift_end = shift_end
        self.breaks = breaks
        self.trips = []

    def is_available(self, trip_start, trip_duration):
        trip_end = trip_start + trip_duration
        if trip_start < self.shift_start or trip_end > self.shift_end:
            return False
        for break_start, break_end in self.breaks:
            if trip_start < break_end and trip_end > break_start:
                return False
        for trip in self.trips:
            if trip_start < trip.departure_time + trip.route.duration and trip_end > trip.departure_time:
                return False
        if self.bus.current_trip:
            bus_trip_end = self.bus.current_trip.departure_time + self.bus.current_trip.route.duration
            if trip_start < bus_trip_end:
                return False
        return True

class Route:
    def __init__(self, stops, direction):
        self.stops = stops
        self.direction = direction
        self.duration = timedelta(minutes=60)

class Trip:
    def __init__(self, departure_time, bus, driver, route):
        self.departure_time = departure_time
        self.bus = bus
        self.driver = driver
        self.route = route
        self.arrival_times = []
```

Рисунок 25 – определяю базовые классы Stop, Bus, Driver, Route, Trip


```

class Schedule:
    def __init__(self):
        self.trips = []

    def add_trip(self, trip):
        self.trips.append(trip)

    def generate_arrival_times(self):
        for trip in self.trips:
            current_time = trip.departure_time
            for stop in trip.route.stops:
                trip.arrival_times.append((stop.name, current_time))
                current_time += timedelta(minutes=20)

# Data initialization
stop_names = [
    "Улица маршала Тухачевского",
    "Университет связи и информатики",
    "Метро Октябрьское поле",
    "МЦК Панфиловская"
]
stops = [Stop(name) for name in stop_names]

route_there = Route(stops, "в сторону МЦД Панфиловская")
route_back = Route(stops[::-1], "в сторону улицы Маршала Тухачевского")

buses = [Bus(f"A00{i}AA") for i in range(1, 31)]

DRIVERS_TYPE1 = [
    "Павел", "Петр", "Панкрат", "Прокл", "Платон",
    "Полина", "Порфирий", "Пересвет", "Пимен", "Прохор",
    "Потап", "Пиона", "Протас", "Президий", "Преслав",
    "Платини", "Праксис", "Поликарп", "Палладий", "Парамон",
]

DRIVERS_TYPE2 = [
    "Вадим", "Валерий", "Василий", "Виктор", "Владимир",
    "Владислав", "Вячеслав", "Виталий", "Валентин", "Венедикт",
    "Велор", "Вилен", "Витольд", "Валдар", "Варлам",
    "Варфоломей", "Василевс", "Велимир", "Вероний", "Вильгельм"
]

start_date = datetime.strptime("2024-10-01 06:00", "%Y-%m-%d %H:%M")
end_date = datetime.strptime("2024-10-02 03:00", "%Y-%m-%d %H:%M")

```

Рисунок 26 – определяю класс Schedule, название остановок, направления маршрутов, имена водителей, начало и конец работы транспорта

```

drivers_type1 = []
for i, name in enumerate(DRIVERS_TYPE1):
    shift_start = start_date + timedelta(hours=i)
    shift_end = min(shift_start + timedelta(hours=8), end_date)
    break_start = shift_start + timedelta(hours=4)
    break_end = break_start + timedelta(hours=1)
    breaks = [(break_start, break_end)]
    drivers_type1.append(Driver(name, 1, buses[i % len(buses)], shift_start, shift_end, breaks))

drivers_type2 = []
shift_start_times = [start_date + timedelta(hours=i) for i in range(24)]
for i, name in enumerate(DRIVERS_TYPE2):
    shift_start = shift_start_times[i % len(shift_start_times)]
    shift_end = min(shift_start + timedelta(hours=12), end_date)
    breaks = []
    break_time = shift_start + timedelta(hours=2, minutes=10)
    while break_time < shift_end:
        break_start = break_time
        break_end = break_start + timedelta(minutes=10)
        breaks.append((break_start, break_end))
        break_time += timedelta(hours=2, minutes=10)
    drivers_type2.append(Driver(name, 2, buses[i % len(buses)], shift_start, shift_end, breaks))

all_drivers = drivers_type1 + drivers_type2

```

Рисунок 27 – генерирую объекты водителей по аналогии с наивным способом (смотреть на стр. 12)

Планировщик рейсов и работа с расписанием

Функция `get_interval` определяет интервал между рейсами в зависимости от текущего времени.

Параметры:

- `current_time`: `datetime` время, для которого требуется определить интервал.

Возвращает:

- `timedelta`: интервал между рейсами.
 - В утренние и вечерние часы пик (7:00–9:00 и 18:00–20:00) интервал составляет 10 минут.
 - В остальное время интервал составляет 20 минут

Потом я генерирую рейсы для создания набора данных, которые в дальнейшем будут использоваться в генетическом алгоритме. Так как

генетический алгоритм решает задачу оптимального распределения ресурсов (водителей и автобусов) для выполнения рейсов. Для этого ему нужен набор рейсов, который определяет, какие поездки должны быть выполнены.

Этот блок кода отвечает за создание рейсов в течение заданного временного интервала, используя интервалы, рассчитанные функцией `get_interval`. Маршруты чередуются между двумя направлениями, автобусы и водители определяются рандомно.

```
def get_interval(current_time):
    if (current_time.hour >= 7 and current_time.hour < 9) or (current_time.hour >= 18 and current_time.hour < 20):
        return timedelta(minutes=10)
    else:
        return timedelta(minutes=20)

trips = []
current_time = start_date
while current_time < end_date:
    interval = get_interval(current_time)
    route = route_there if len(trips) % 2 == 0 else route_back
    trip = Trip(current_time, random.choice(buses), random.choice(all_drivers), route)
    trips.append(trip)
    current_time += interval
```

Рисунок 28 — создаю рейсы для дальнейшей оптимизации

Генетический алгоритм для оптимизации

Функция `generate_initial_population` создает начальную популяцию для генетического алгоритма, представляя каждое расписание в виде хромосомы, где каждая запись хромосомы связывает рейс с водителем и автобусом.

Параметры:

`pop_size`: int размер начальной популяции, то есть количество хромосом.

`trips`: list список рейсов, которые необходимо распределить между водителями и автобусами.

`drivers`: list список доступных водителей.

`buses`: list список доступных автобусов.

Возвращает:

population: list список хромосом, где каждая хромосома — это список кортежей (trip, driver, bus).

Логика работы:

1. Инициализируется пустой список population.
2. Для каждой хромосомы (в количестве pop_size):
 - Создается пустая хромосома chromosome.
 - Для каждого рейса из trips:
 1. Формируется список доступных водителей (available_drivers) и автобусов (available_buses), основываясь на их доступности для текущего рейса.
 2. Если есть доступные водитель и автобус, то случайным образом выбираются водитель и автобус, которые добавляются в хромосому в виде кортежа (trip, driver, bus).
3. Хромосома добавляется в популяцию.
4. Возвращается полный список хромосом (population).

Также если для рейса невозможно найти подходящего водителя или автобус, он пропускается.

```
def generate_initial_population(pop_size, trips, drivers, buses):
    population = []
    for _ in range(pop_size):
        chromosome = []
        for trip in trips:
            available_drivers = [d for d in drivers if d.is_available(trip.departure_time, trip.route.duration)]
            available_buses = [b for b in buses if not b.current_trip or b.current_trip.departure_time + b.current_trip.route.duration <= trip.departure_time]
            if available_drivers and available_buses:
                driver = random.choice(available_drivers)
                bus = random.choice(available_buses)
                chromosome.append((trip, driver, bus))
            else:
                continue
        population.append(chromosome)
    return population
```

Рисунок 29 - функция generate_initial_population для создания начальной популяции

После создания начальной популяции нужно оценить данные, для этого создается функция `fitness_function`, которая оценивает качество расписания (хромосомы) с точки зрения соответствия заданным ограничениям и времени, если какие-то требования не выполняются назначаются штрафы.

Параметры:

`chromosome: list` хромосома, представляющая расписание в виде списка кортежей (`trip, driver, bus`).

`start_time: datetime` начальное время расчетного интервала.

`end_time: datetime` конечное время расчетного интервала.

Возвращает:

`fitness: int` значение фитнес-функции, оценивающее хромосому. Чем выше значение, тем лучше расписание.

Алгоритм:

1. Инициализируется переменная `fitness = 0`.
2. Для каждого рейса в хромосоме проверяются:
 - Входит ли рейс в расчетный интервал (`start_time <= trip.departure_time < end_time`).
 - Доступен ли водитель для выполнения рейса.
 - Если все условия выполнены, начисляется положительный вклад в фитнес.
 - Если условия не выполнены (например, водитель недоступен), вычитается штраф (-10 или -100).
3. Проверяется наличие пересекающихся рейсов:
 - Для каждого водителя и автобуса рейсы проверяются на временные конфликты. За каждое пересечение штрафуются -10.

4. Возвращается итоговое значение фитнеса.

```
def fitness_function(chromosome, start_time, end_time):
    fitness = 0
    driver_trips = {}
    bus_trips = {}
    for trip, driver, bus in chromosome:
        if start_time <= trip.departure_time < end_time:
            if driver.is_available(trip.departure_time, trip.route.duration):
                if driver.name not in driver_trips:
                    driver_trips[driver.name] = []
                driver_trips[driver.name].append((trip.departure_time, trip.route.duration))
                if bus.bus_number not in bus_trips:
                    bus_trips[bus.bus_number] = []
                bus_trips[bus.bus_number].append((trip.departure_time, trip.route.duration))
                fitness += 1 # за успешное назначение
            else:
                fitness -= 10 # назначение водителя невозможно, штраф
        else:
            fitness -= 100 # поездка выходит за границы временного окна, сильно штрафует

    # проверка на наличие пересекающихся поездок у водителей
    for trips_list in driver_trips.values():
        for i in range(len(trips_list)):
            for j in range(i+1, len(trips_list)):
                if trips_list[i][0] < trips_list[j][0] + trips_list[j][1] and trips_list[i][0] + trips_list[i][1] > trips_list[j][0]:
                    fitness -= 10 # поездки пересекаются, штраф

    # проверка на наличие пересекающихся поездок у автобусов
    for trips_list in bus_trips.values():
        for i in range(len(trips_list)):
            for j in range(i+1, len(trips_list)):
                if trips_list[i][0] < trips_list[j][0] + trips_list[j][1] and trips_list[i][0] + trips_list[i][1] > trips_list[j][0]:
                    fitness -= 10 # поездки пересекаются

    return fitness
```

Рисунок 30 - fitness_function для оценки качества расписания

Функция selection выбирает лучших родителей из популяции для создания следующего поколения.

Параметры:

population: list список текущей популяции (хромосомы).

fitness_scores: list список значений фитнес-функции для каждой хромосомы.

num_parents: int количество родителей для отбора.

Возвращает:

parents: list список отобранных хромосом-родителей.

Алгоритм:

1. Инициализируется пустой список parents.

2. Повторяется num_parents раз:

- Находится индекс хромосомы с максимальным значением фитнеса.
- Хромосома добавляется в список parents, а её значение фитнеса временно заменяется на -9999, чтобы избежать повторного выбора.

3. Возвращается список родителей.

```
def selection(population, fitness_scores, num_parents):  
    parents = []  
    for _ in range(num_parents):  
        max_fitness_idx = fitness_scores.index(max(fitness_scores))  
        parents.append(population[max_fitness_idx])  
        fitness_scores[max_fitness_idx] = -9999  
    return parents
```

Рисунок 31 - функция selection для выбора лучших родителей для следующего поколения

Функция crossover выполняет скрещивание двух хромосом, создавая потомка с элементами от каждого родителя.

Параметры:

parent1: list первая хромосома-родитель.

parent2: list вторая хромосома-родитель.

Возвращает:

child: list хромосома-потомок, созданная из генов родителей.

Алгоритм:

1. Определяется точка скрещивания случайным образом.
2. Потомок формируется как объединение первой части от parent1 и второй части от parent2.
3. Проверяются уникальность и корректность назначений:

— Если водитель уже назначен на другой рейс, производится замена на доступного водителя.

— Аналогично проверяется доступность автобуса.

4. Возвращается корректная хромосома-потомок.

```
def crossover(parent1, parent2):
    crossover_point = random.randint(0, len(parent1)-1)
    child = parent1[:crossover_point] + parent2[crossover_point:]
    driver_assigned = {}
    bus_assigned = {}
    for i, (trip, driver, bus) in enumerate(child):
        if driver.name in driver_assigned:
            available_drivers = [d for d in all_drivers if d.is_available(trip.departure_time, trip.route.duration)]
            if available_drivers:
                child[i] = (trip, random.choice(available_drivers), bus)
            else:
                child[i] = (trip, None, bus)
        else:
            driver_assigned[driver.name] = True
            if bus.bus_number in bus_assigned:
                available_buses = [b for b in buses if not b.current_trip or b.current_trip.departure_time + b.current_trip.route.duration <= trip.departure_time]
                if available_buses:
                    child[i] = (trip, driver, random.choice(available_buses))
                else:
                    child[i] = (trip, driver, None)
            else:
                bus_assigned[bus.bus_number] = True
    return child
```

Рисунок 32 – функция crossover для скрещивания

Функция `mutation` выполняет мутацию хромосомы, случайно изменяя одного из её генов.

Параметры:

`child`: list хромосома для мутации.

`trips`: list список рейсов.

`drivers`: list список доступных водителей.

`buses`: list список доступных автобусов.

Возвращает:

`child`: list модифицированная хромосома после мутации.

Алгоритм:

1. Случайным образом выбирается ген (индекс хромосомы).
2. Для выбранного рейса определяется новый доступный водитель и автобус.
3. Ген хромосомы обновляется новым кортежем (trip, new_driver, new_bus).
4. Возвращается обновленная хромосома.

```
def mutation(child, trips, drivers, buses):  
    mutation_index = random.randint(0, len(child)-1)  
    trip, driver, bus = child[mutation_index]  
    available_drivers = [d for d in drivers if d.is_available(trip.departure_time, trip.route.duration)]  
    available_buses = [b for b in buses if not b.current_trip or b.current_trip.departure_time + b.current_trip.route  
    if available_drivers and available_buses:  
        new_driver = random.choice(available_drivers)  
        new_bus = random.choice(available_buses)  
        child[mutation_index] = (trip, new_driver, new_bus)  
    return child
```

Рисунок 33 – функция mutation

Формирование итогового расписания

Параметры

- pop_size = 400: размер начальной популяции хромосом (решений).
- num_generation = 300: количество поколений, которые будет проработано алгоритмом.
- num_parents = 80: количество родителей, которые выбираются для генерации следующего поколения
- start_date, end_date: временные рамки, в которых должно находиться расписание рейсов.
- trips: список всех доступных рейсов.
- all_drivers, buses, stops: списки доступных водителей и автобусов, остановок

Алгоритм

1. Инициализация популяции:

- Вызывается функция `generate_initial_population`, которая создает начальную популяцию решений на основе рейсов, водителей и автобусов.

2. Эволюция популяции:

Для каждого поколения (из 300):

- Вычисляются фитнес-оценки всех хромосом текущей популяции через `fitness_function`.
- Из популяции отбираются родители с наивысшими фитнес-оценками (через `selection`).
- Из родителей создаются дети (новые решения):
 - Пары родителей выбираются случайно.
 - Дети формируются через операторы кроссовера (`crossover`) и мутации (`mutation`).
- Новая популяция состоит из родителей и их детей.

```
pop_size = 400
num_generations = 300
num_parents = 80

population = generate_initial_population(pop_size, trips, all_drivers, buses)

for generation in range(num_generations):
    fitness_scores = [fitness_function(chromosome, start_date, end_date) for chromosome in population]
    parents = selection(population, fitness_scores, num_parents)
    children = []
    for _ in range(pop_size - num_parents):
        parent1 = random.choice(parents)
        parent2 = random.choice(parents)
        child = crossover(parent1, parent2)
        child = mutation(child, trips, all_drivers, buses)
        children.append(child)
    population = parents + children
```

Рисунок 34 – параметры, инициализация популяции и ее эволюция

3. Выбор лучшего решения:

- Хромосома с максимальной фитнес-оценкой выбирается как лучшее решение.
- Формируется объект `Schedule`, в который добавляются рейсы лучшей хромосомы.
- Водители и автобусы обновляются в соответствии с выбранными рейсами.

```
best_chromosome = population[fitness_scores.index(max(fitness_scores))]  
schedule = Schedule()  
for trip, driver, bus in best_chromosome:  
    schedule.add_trip(trip)  
    driver.trips.append(trip)  
    bus.current_trip = trip  
  
schedule.generate_arrival_times()
```

Рисунок 35 – выбор лучшего решения

4. Генерация расписания остановок:

- Формируется структура `stop_schedules`, которая содержит расписания для всех остановок в обе стороны маршрута.
- Расписания сортируются по времени прибытия, и данные о каждом рейсе (время прибытия, водитель, номер рейса) выводятся для каждой остановки.

```
stop_schedules = {stop.name: {"в сторону МЦД Панфиловская": [], "в сторону улицы Маршала Тухачевского": []} for stop in stops}  
  
for trip in schedule.trips:  
    for stop_name, arrival_time in trip.arrival_times:  
        direction = trip.route.direction  
        stop_schedules[stop_name][direction].append({  
            "Время прибытия": arrival_time,  
            "Водитель": trip.driver.name,  
            "Номер рейса": trip.bus.bus_number,  
            "Направление": direction  
        })
```

Рисунок 36 – генерация расписания для каждой остановки

5. Вывод итогового расписания:

- Для каждой остановки и каждого направления печатается список рейсов с деталями.
- Выводится общее количество сгенерированных рейсов.

```
for stop_name, directions in stop_schedules.items():
    print(f"Остановка: {stop_name}")
    for direction in ["в сторону МЦД Панфиловская", "в сторону улицы Маршала Тухачевского"]:
        print(f"    {direction}:")
        sorted_entries = sorted(directions[direction], key=lambda x: x["Время прибытия"])
        for entry in sorted_entries:
            print(f"        Время: {entry['Время прибытия'].strftime('%H:%M')}, Водитель: {entry['Водитель']], Номер рейса: {entry['Номер рейса']}")
print(f"Всего сгенерированных рейсов: {len(schedule.trips)}")
```

Рисунок 34 – формирование расписания

ИТОГОВЫЙ ВЫВОД ПРОГРАММЫ

Расписание остановок генерировалось около 5 минут, из-за того, что я задавала большое количество поколений, чтоб добиться наилучшего результата. И результат действительно оказался лучше, это можно отследить даже по большому количеству сгенерированных рейсов. Функционал для пользователя я решила не делать, чтоб не перегружать программу, которая и без того долго работает. Визуально оценивая работу водителей, можно сказать, что простоев у водителей нет практически, после перерыва водитель сразу идет на работу катать рейсы. Новую Excel таблицу не вижу надобности делать, так как результат работы практически не отличается от наивного способа, кроме того, что вечером нет больших окон в расписании длиной в час, и в генетическом алгоритме каждый раз новый результат, с разным порядком выхода на работу водителей.

```
print(f"    Время: {entry['Время прибытия'].strftime('%H:%M')}, Водитель: {entry['Водитель']}, Номер рейса: {entry['Номер рейса']}\n")

print(f"Всего сгенерированных рейсов: {len(schedule.trips)}")
```

✓ 4m 49.8s

Остановка: Улица маршала Тухачевского
в сторону МЦД Панфиловская:

Время:	06:00,	Водитель:	Вилен,	Номер рейса:	A0010AA
Время:	06:20,	Водитель:	Валерий,	Номер рейса:	A003AA
Время:	06:40,	Водитель:	Василевс,	Номер рейса:	A008AA
Время:	07:00,	Водитель:	Прохор,	Номер рейса:	A001AA
Время:	07:10,	Водитель:	Пересвет,	Номер рейса:	A001AA
Время:	07:20,	Водитель:	Виталий,	Номер рейса:	A0021AA
Время:	07:30,	Водитель:	Петр,	Номер рейса:	A0028AA
Время:	07:40,	Водитель:	Платини,	Номер рейса:	A0022AA
Время:	07:50,	Водитель:	Вадим,	Номер рейса:	A0013AA
Время:	08:00,	Водитель:	Виктор,	Номер рейса:	A0020AA
Время:	08:10,	Водитель:	Велимир,	Номер рейса:	A0018AA
Время:	08:20,	Водитель:	Венедикт,	Номер рейса:	A0015AA
Время:	08:30,	Водитель:	Вячеслав,	Номер рейса:	A001AA
Время:	08:40,	Водитель:	Платон,	Номер рейса:	A0014AA
Время:	08:50,	Водитель:	Президий,	Номер рейса:	A004AA
Время:	09:00,	Водитель:	Прокл,	Номер рейса:	A007AA
Время:	09:20,	Водитель:	Поликарп,	Номер рейса:	A0015AA
Время:	09:40,	Водитель:	Петр,	Номер рейса:	A0013AA
Время:	10:00,	Водитель:	Парамон,	Номер рейса:	A0019AA
Время:	10:20,	Водитель:	Валентин,	Номер рейса:	A009AA
Время:	10:40,	Водитель:	Вадим,	Номер рейса:	A0015AA
Время:	11:00,	Водитель:	Пересвет,	Номер рейса:	A007AA
Время:	11:20,	Водитель:	Парамон,	Номер рейса:	A0024AA
Время:	11:40,	Водитель:	Президий,	Номер рейса:	A004AA
Время:	12:00,	Водитель:	Поликарп,	Номер рейса:	A0023AA
Время:	12:20,	Водитель:	Пересвет,	Номер рейса:	A009AA
Время:	12:40,	Водитель:	Валентин,	Номер рейса:	A0018AA
Время:	13:00,	Водитель:	Варлам,	Номер рейса:	A0026AA

Рисунок 35 – вывод расписания для всех остановок

```
Время: 21:10, Водитель: Платон, Номер рейса: A008AA
Время: 21:30, Водитель: Панкрат, Номер рейса: A0021AA
Время: 21:50, Водитель: Венедикт, Номер рейса: A0010AA
Время: 22:10, Водитель: Пиона, Номер рейса: A0022AA
Время: 22:30, Водитель: Велимир, Номер рейса: A0018AA
Время: 22:50, Водитель: Преслав, Номер рейса: A0027AA
Время: 23:10, Водитель: Варлам, Номер рейса: A0025AA
Время: 23:30, Водитель: Вильгельм, Номер рейса: A0014AA
Время: 23:50, Водитель: Владислав, Номер рейса: A001AA
Время: 00:10, Водитель: Протас, Номер рейса: A0020AA
Время: 00:30, Водитель: Панкрат, Номер рейса: A0023AA
Время: 00:50, Водитель: Велимир, Номер рейса: A002AA
Время: 01:10, Водитель: Велимир, Номер рейса: A002AA
Время: 01:30, Водитель: Владислав, Номер рейса: A006AA
Время: 01:50, Водитель: Палладий, Номер рейса: A0026AA
Всего сгенерированных рейсов: 145
```

Рисунок 36 – всего сгенерировано 145 рейсов

ОБЩИЙ ВЫВОД И СРАВНЕНИЕ ОПТИМАЛЬНОСТИ

В результате проведенного исследования оптимизации расписания автобусов двумя методами – наивного способа и генетического алгоритма – мне удалось выявить ключевые различия в их эффективности, удобстве использования и корректности.

Генетический алгоритм продемонстрировал значительное преимущество в плане оптимальности решения. За счет учета каждого интервала времени в процессе моделирования удалось минимизировать возникновение пропусков, характерных для наивного метода. Наивный, в свою очередь, страдает от серьезного недостатка: ближе к концу рабочего дня наблюдаются временные окна, в которые водители не выходят на смену, что существенно снижает общую эффективность работы системы. Конечно, эту проблему можно решить, заставив водителей работать неполный рабочий день, но это не совсем оптимально и выгодно компаниям, которые выплачивают зарплаты таким работникам.

Несмотря на более сложную реализацию генетического алгоритма он более приспособлен, и легко адаптируемый под изменяющиеся требования. В то же время, метод "влоб" требует значительных доработок и ручных корректировок для достижения приемлемого результата. При изменении даты, в моем коде образуются окна даже в начале дня, особенно в час пик. Генетический алгоритм выглядит более надежным в контексте дальнейшего масштабирования и модификации.

Ограничения генетического алгоритма связаны с высокой вычислительной сложностью, что может потребовать дополнительных ресурсов при значительном увеличении объема данных. Тем не менее, учитывая достигнутый уровень оптимальности и удобства, именно этот подход мне представляется наиболее перспективным для решения задач, требующих высокой точности и адаптивности.

Таким образом, генетический алгоритм подтверждает свое преимущество как современный и более эффективный инструмент оптимизации в сравнении с наивным методом.

Полный код и вывод программы можно посмотреть в моем репозитории

<https://github.com/ShedovaNastya/Coursework-optimization-of-bus-schedule/tree/main>