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»**

Кафедра «Математическая кибернетика и информационные технологии»

**Курсовая работа**

по дисциплине

**«Структуры и алгоритмы обработки данных»**

Выполнил:

студент группы БВТ2201 Овчинников Д. О.

Проверил:

преподаватель Симонов С. Е.

Москва, 2024 год

Оглавление

[Введение 3](#_Toc185086924)

[Цель работы 3](#_Toc185086925)

[Задачи 3](#_Toc185086926)

[Ход работы 5](#_Toc185086927)

[Алгоритм в лоб 5](#_Toc185086928)

[Генетический алгоритм 10](#_Toc185086929)

[Сравнение алгоритмов 14](#_Toc185086930)

[Заключение 15](#_Toc185086931)

[Вывод 15](#_Toc185086932)

[Список источников 16](#_Toc185086933)

[Приложения 17](#_Toc185086934)

[GitHub 17](#_Toc185086935)

[Алгоритм в лоб 17](#_Toc185086936)

[Генетический алгоритм 24](#_Toc185086937)

# Введение

## Цель работы

Создать два алгоритма (в лоб и генетический) для составления оптимального графика работы водителей и маршрутов автобусов. Необходимо учесть все условия и сравнить подходы между собой.

## Задачи

Исходные данные:

1) Автопарк: 4 автобуса (задаётся изначально).

2) Водители: как можно меньше.

3) Целевая функция: минимизация числа задействованных водителей и автобусов, учитывая заработную плату водителей, пассажиропоток и количество автобусов.

График работы водителей:

1) 8 часов в сутки, выходные – суббота и воскресенье, без ночных смен, начинают работать с 6:00 до 10:00 утра.

2) 12 часов в сутки, работа через 2 дня (1 день работает, 2 отдыхает), работа ночью и в субботу и воскресенье возможны только в этом варианте.

Долгие и короткие перерывы:

1) короткие перерывы по 10–15 минут каждые 2–4 часа на конечной станции. Водители не могут одновременно уходить на перерыв, чтобы автобусы продолжали движение по маршрутам.

2) долгий перерыв (1 час) берется на конечной станции в период с 13:00 до 15:00.

Остальные условия:

1) Автобусы должны круглосуточно курсировать.

2) Кольцевой маршрут.

3) Пересменка водителей длится 15 минут и происходит на конечной станции.

Выходные данные:

Итоговые таблицы, содержащие:

● Распределение водителей по автобусам.

● Распределение смен водителей по дням недели.

● Время начала и окончания смен каждого водителя.

● Время долгих и коротких перерывов.

# Ход работы

## Алгоритм в лоб

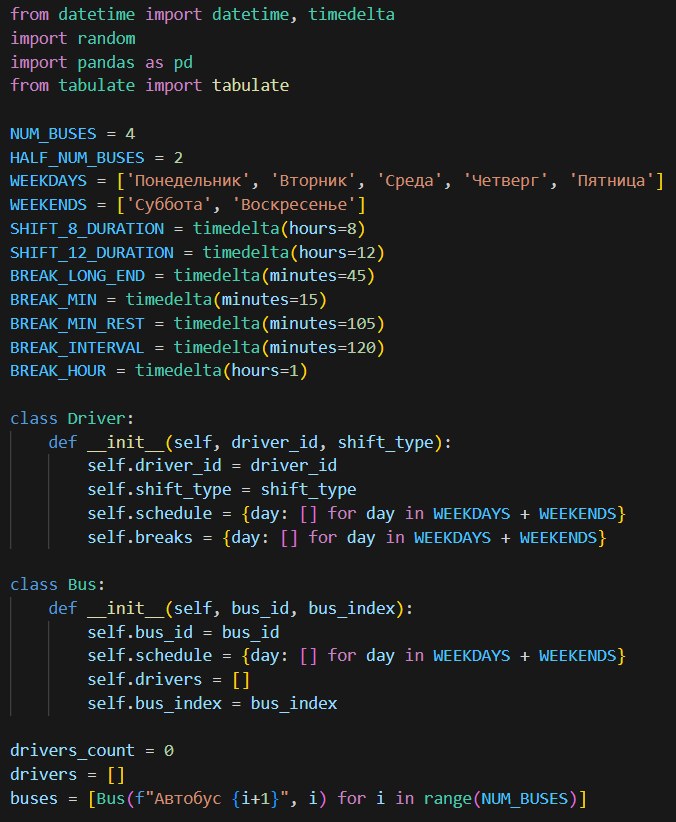


Рис. 1 – задаём глобальные переменные (кол-во автобусов, дни недели, длительности смен и перерывов), классы для водителей и автобусов, а также их счётчики.

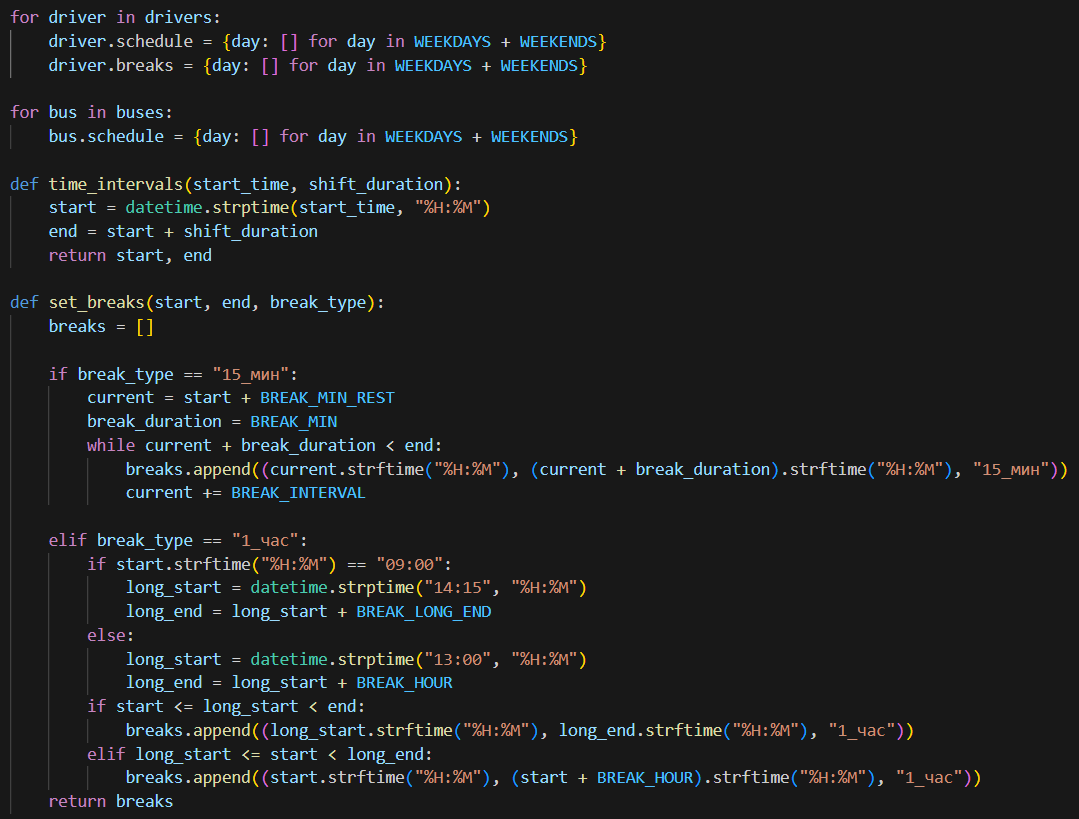
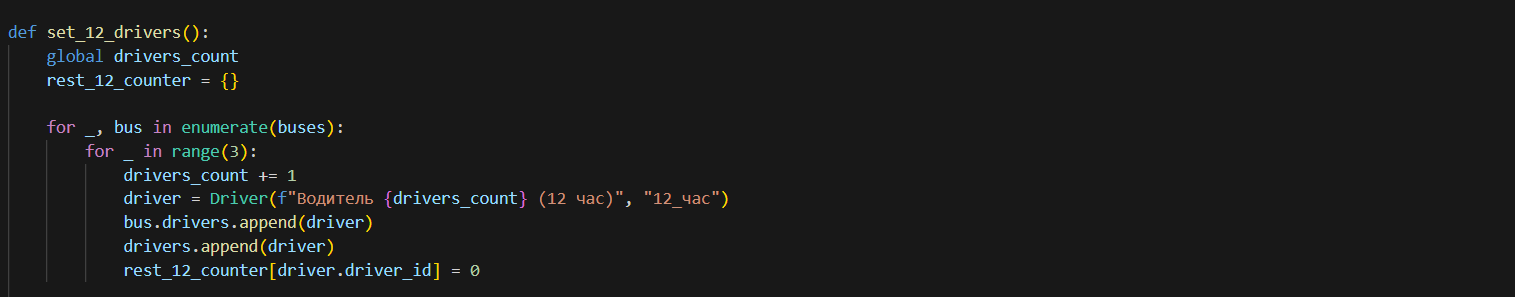


Рис. 2 – создаём временные интервалы (начало и конец смены), назначаем перерывы для водителей в зависимости от типа смены (по 15 минут или 1 часу).



Рис. 3 – распределяем 8-часовых водителей: первая группа автобусов начинает смену в 06:00, а вторая в 09:00; также включаем перерывы, зависящие от времени начала смены.



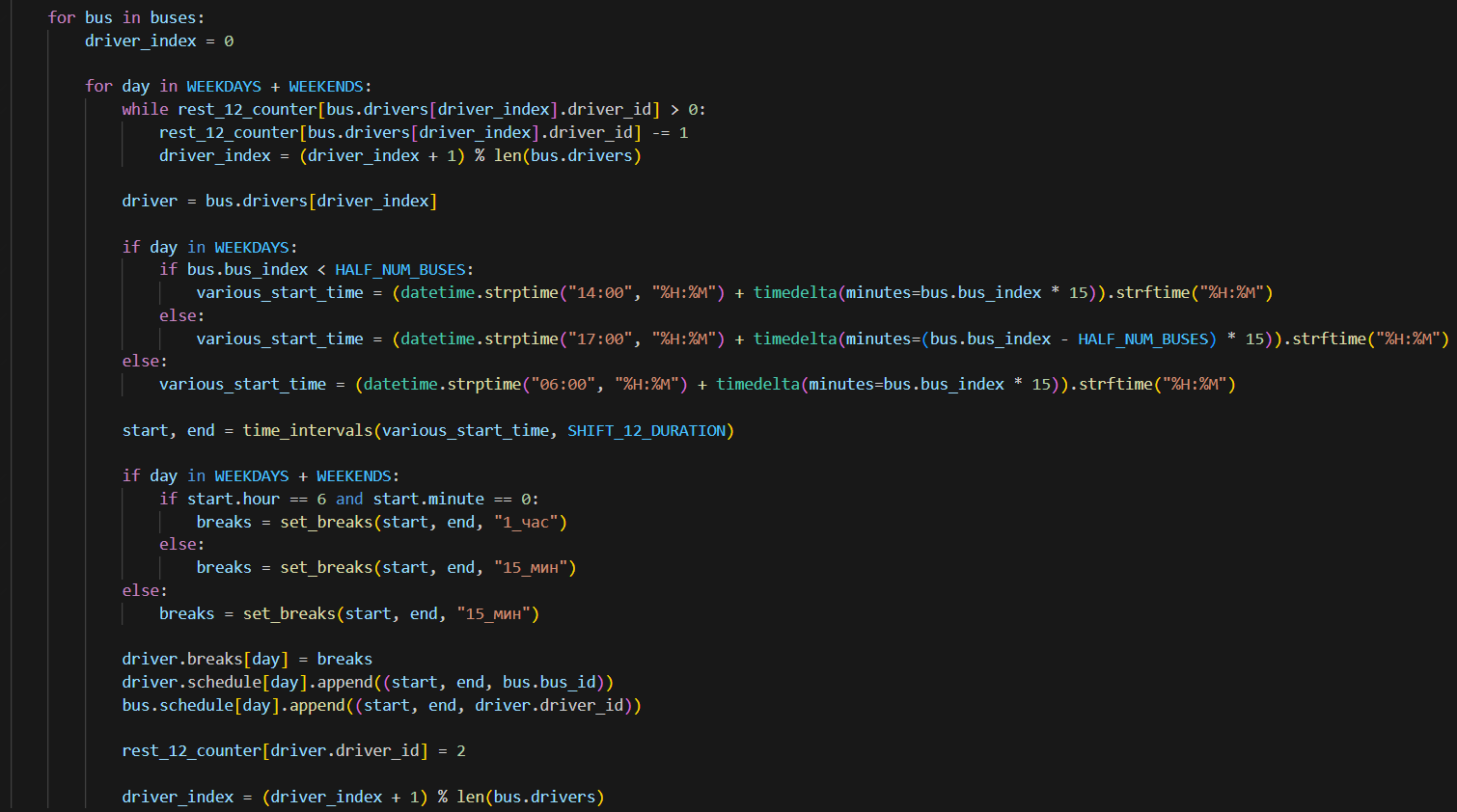


Рис. 4 – распределяем 12-часовых водителей: для каждой группы автобусов выделяем по 3 водителя на автобус; используем систему ротации с учетом разного времени начала смены 2 дней на отдых.

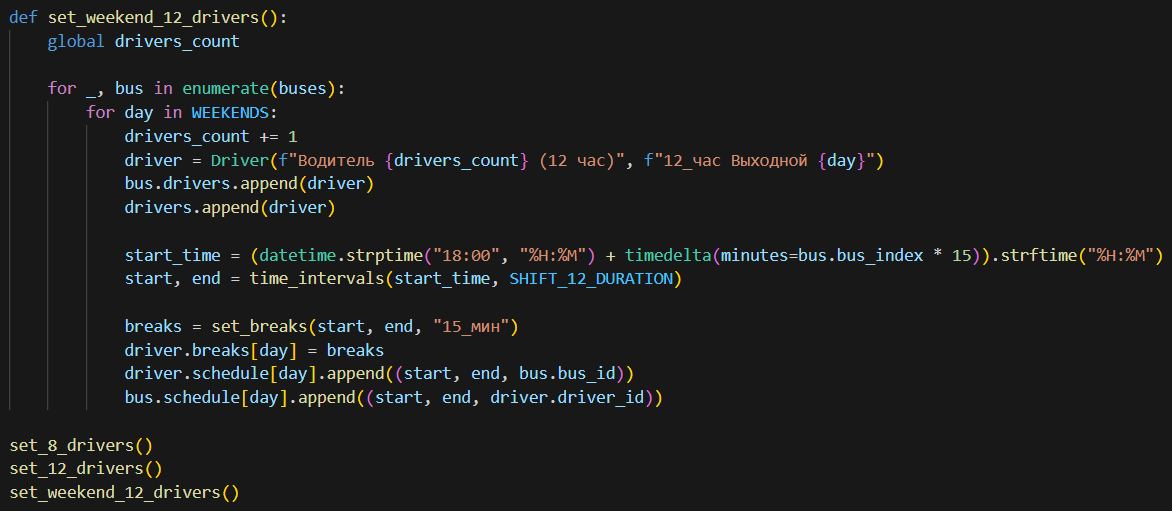


Рис. 5 – распределяем водителей, работающих в выходные, и вызываем последовательно функции для назначения водителей на все смены.

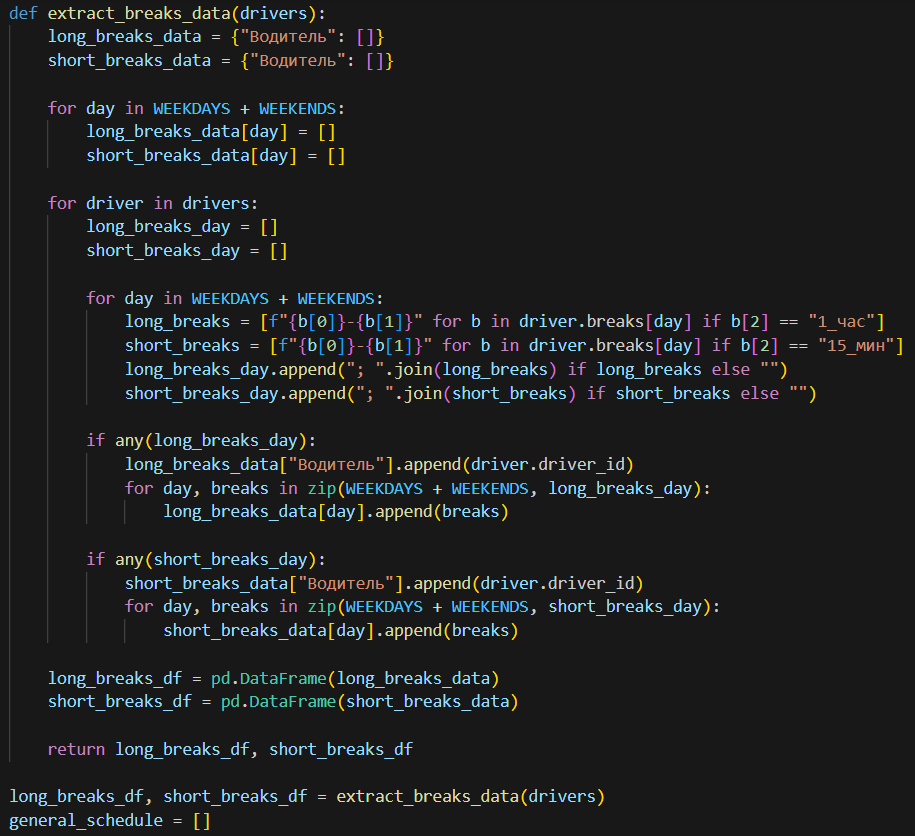


Рис. 6 – извлекаем данные о перерывах из расписаний водителей, группируем данные по дням недели для каждого водителя и возвращаем результат в виде двух DataFrame.

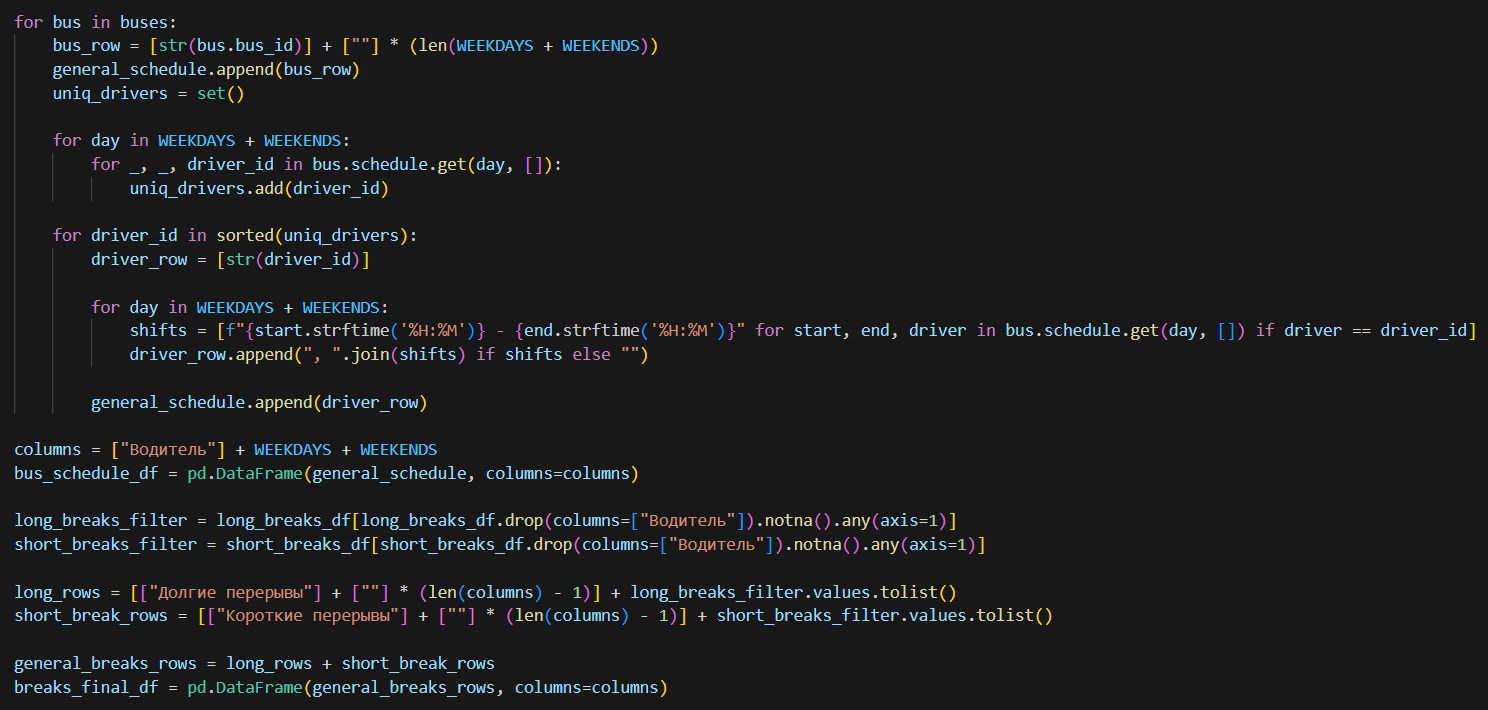


Рис. 7 – для каждого автобуса формируем строки с расписанием водителей.



Рис. 8 – кастомизируем индексы таблицы для более удобного восприятия и выводим результат.

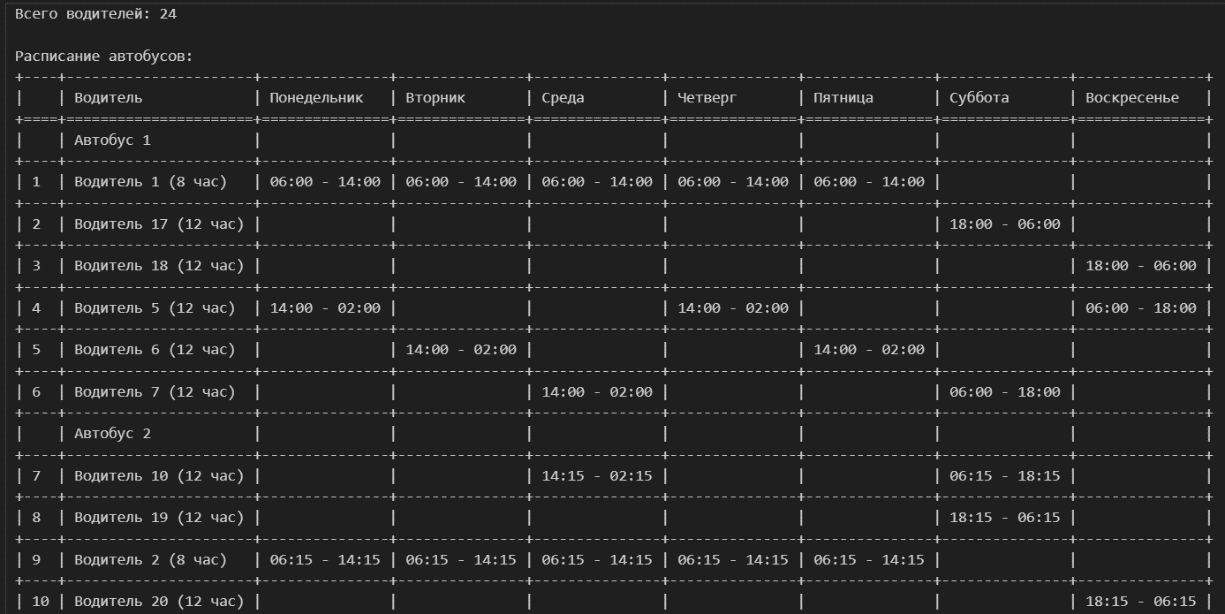


Рис. 9 – вывод общего количества водителей и их расписания.

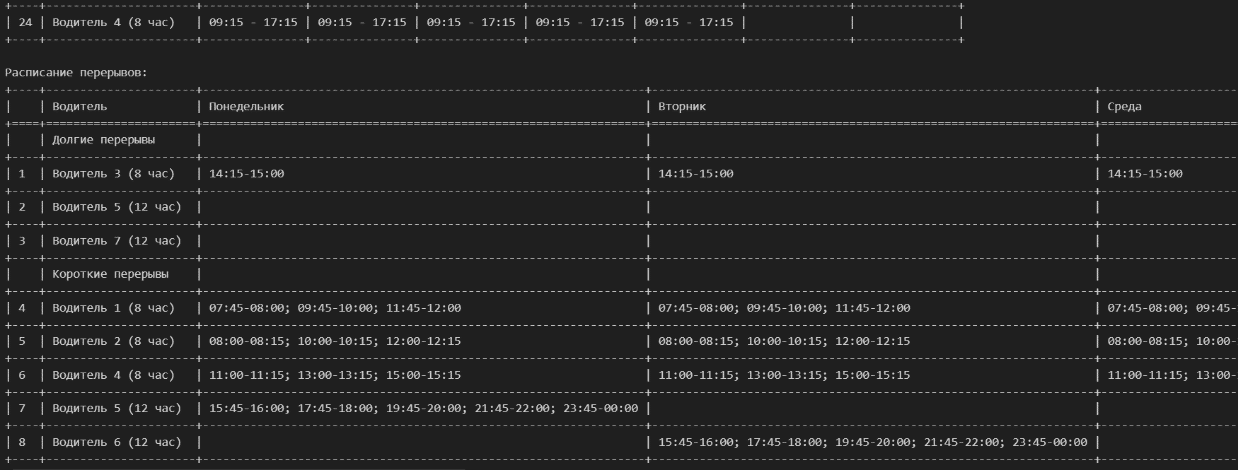
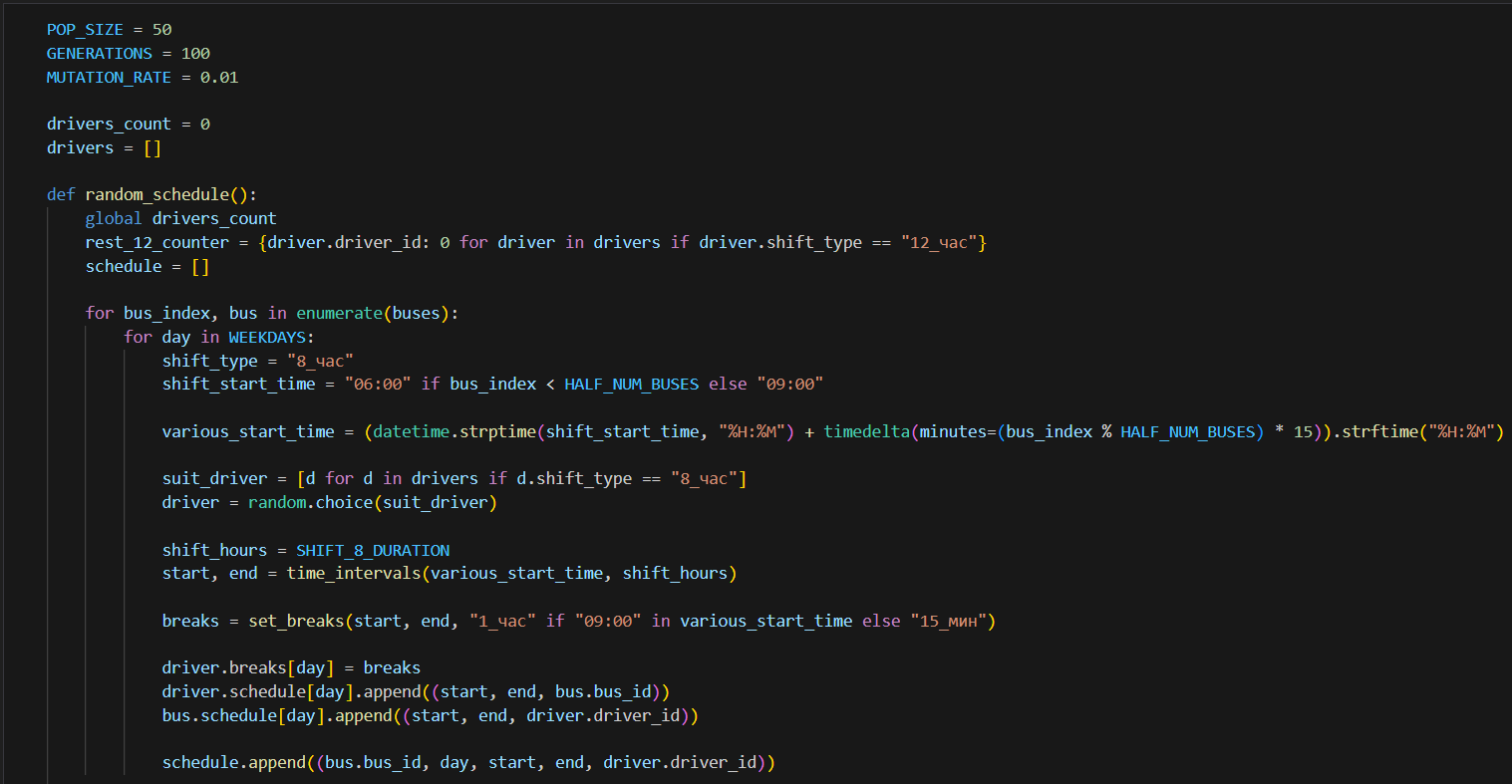


Рис. 10 – вывод расписания перерывов.

## Генетический алгоритм



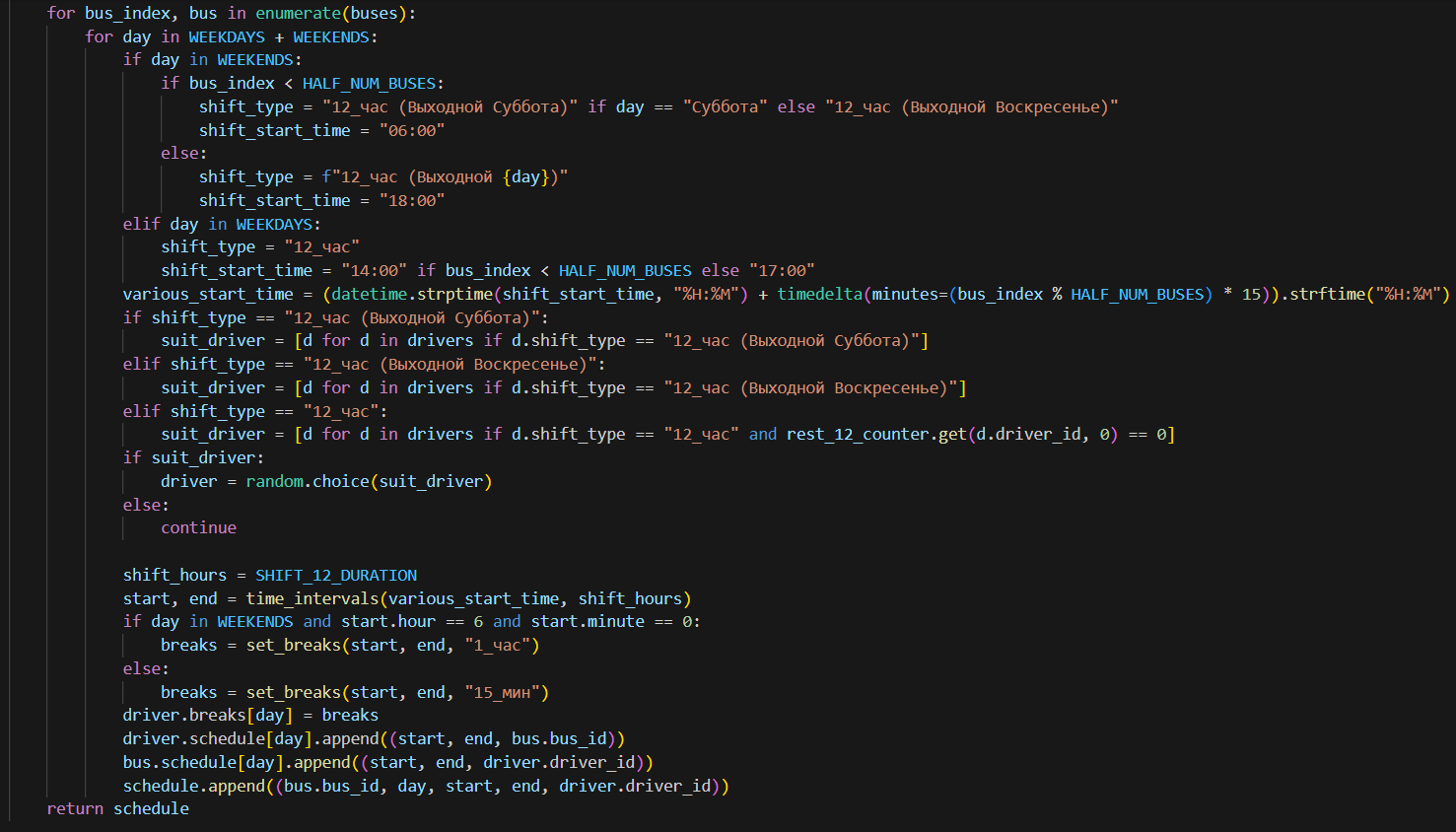


Рис. 11 – задаём новые глобальные переменные для генетического алгоритма (размер популяции, количество поколений, вероятность мутации); обнуляем счётчик водителей; создаём функцию для генерации случайных расписаний на основе имеющихся условий для будущей популяции.

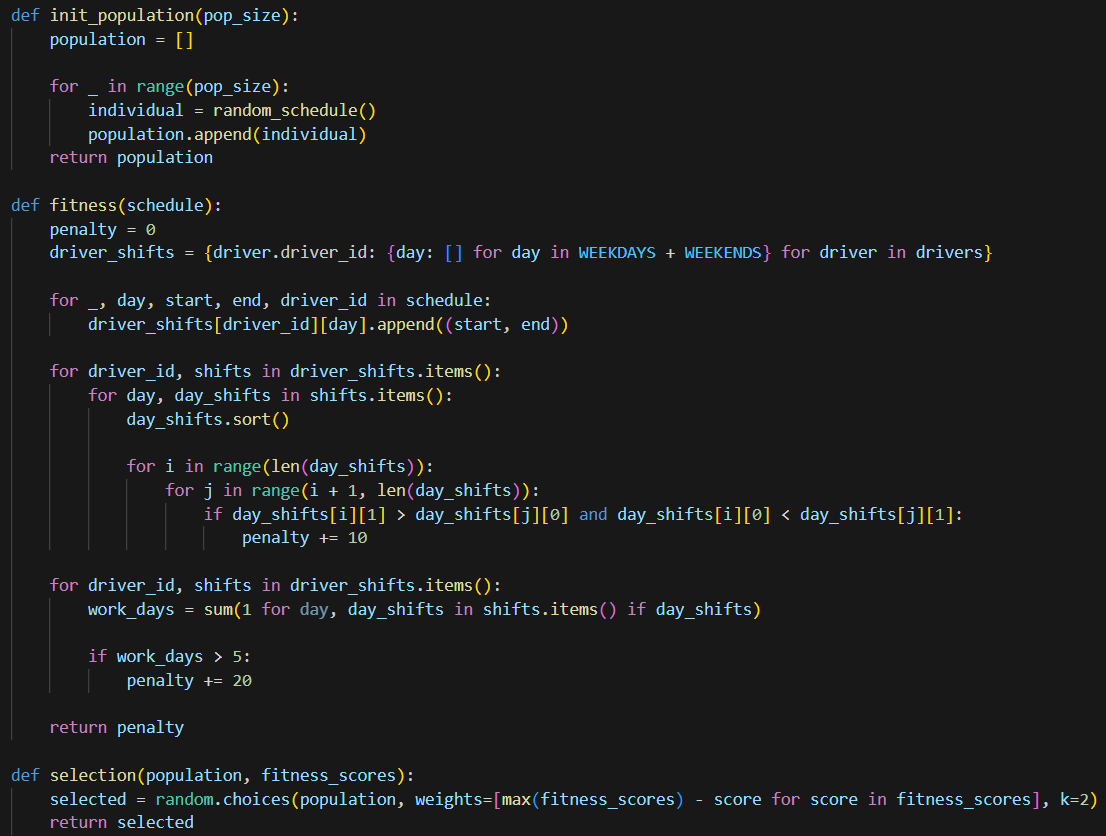


Рис. 12 – инициализируем нашу популяцию, после чего фитнесс-функция рассчитывает штрафы для расписания на основе перекрытия смен, нарушения отдыха и невыполнения требования к перерывам; далее selection выбирает два расписания («родителя») на основе их приспособленности и чем меньше штраф, тем больше шанс быть выбранным.

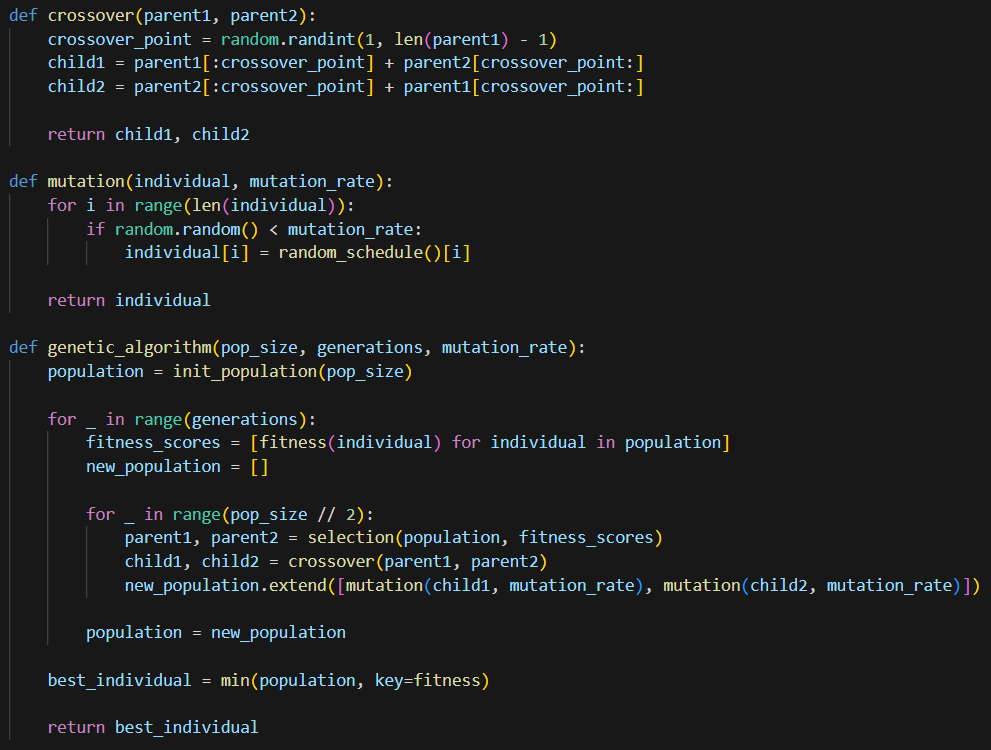


Рис. 13 – кроссовер объединяет расписания двух родителей, создавая новые комбинации; мутация случайно изменяет элементы расписания, чтобы внести разнообразие; в итоге создаём генетический алгоритм, где будут происходить все этапы.

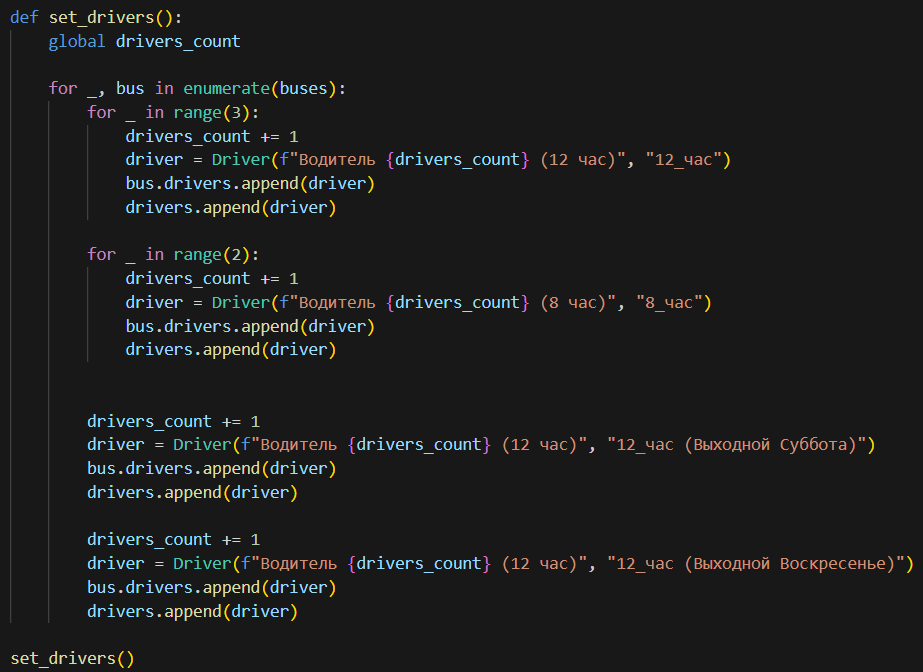


Рис. 14 – распределяем водителей по автобусам.

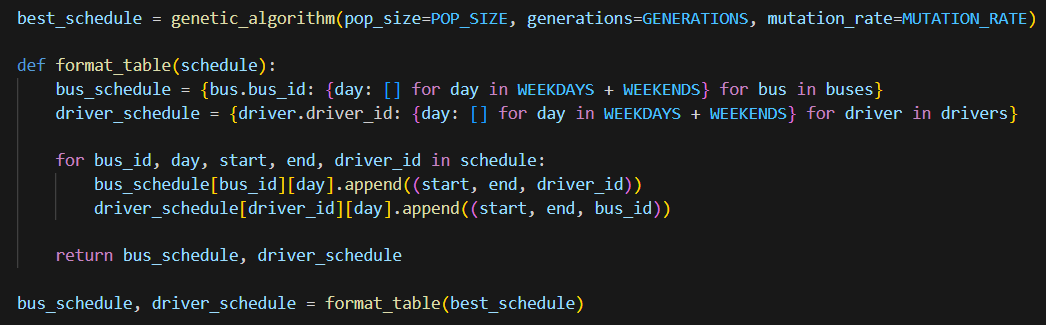


Рис. 15 – выявляем лучшее расписание с помощью генетического алгоритма и конвертируем его в подходящий формат (для вывода таблиц используются идентичные строки, как и в первом алгоритме).

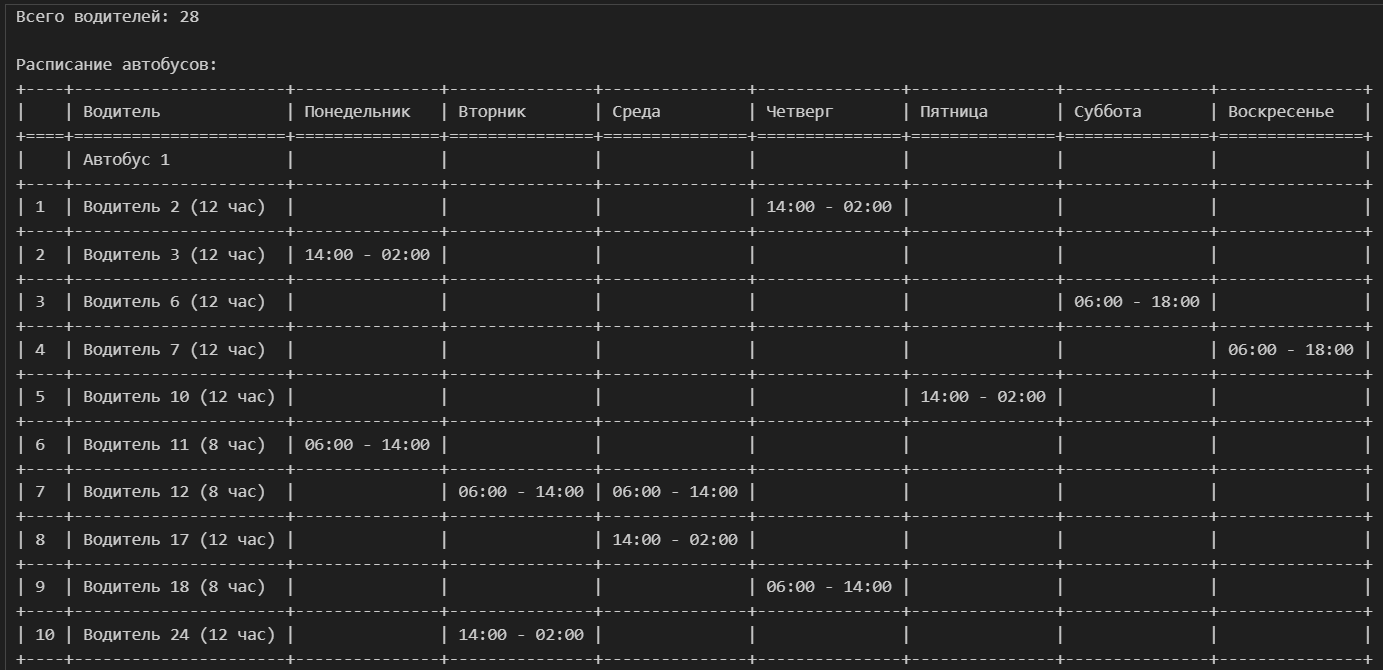


Рис. 16 – вывод общего количества водителей и их расписания.

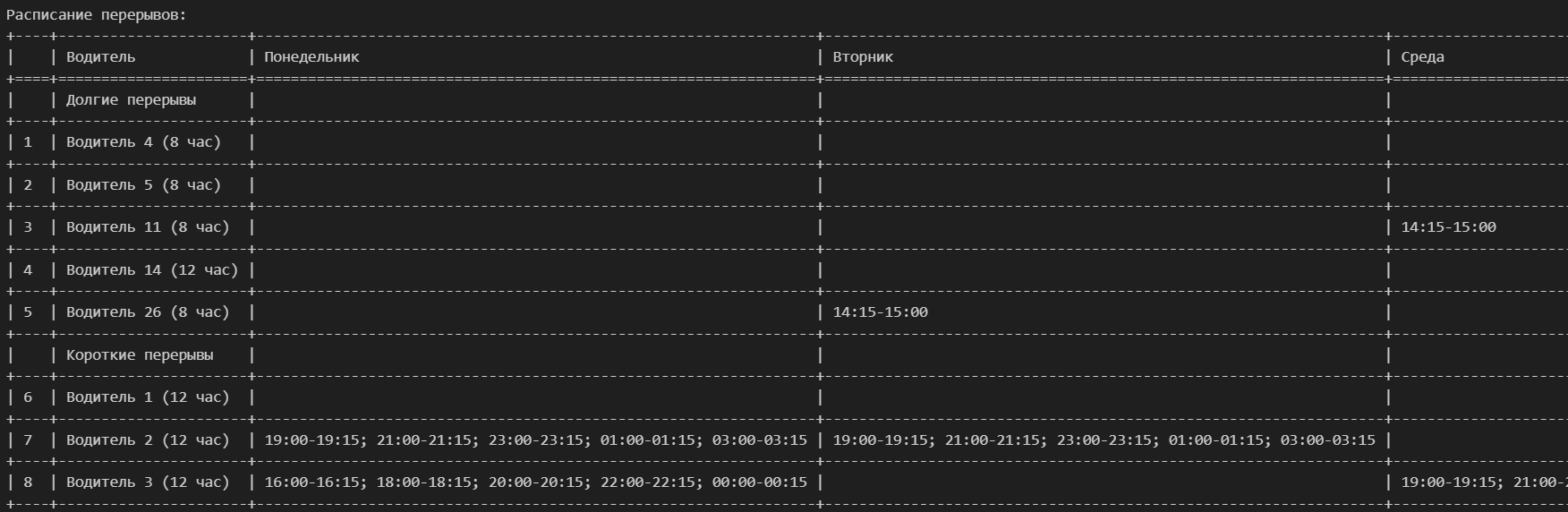


Рис. 17 – вывод расписания перерывов.

## Сравнение алгоритмов

|  |  |  |
| --- | --- | --- |
|  | Алгоритм в лоб | Генетический алгоритм |
| Временная сложность | O(D \* C ^ T); где D – водители, C – комбинации, T – интервалы времени. | O(G \* P \* T); где G – поколения, P – размер популяции, T – время оценивания. |
| Ресурсоёмкость | Потребует значительных ресурсов с большим количеством условий. | Эффективно использует ресурсы за счёт частичной оценки вариантов. |
| Гибкость | Непригоден для адаптации при изменении условий задачи. | Легко адаптируется к изменениям благодаря способности переоценивать симуляцию. |
| Точность | Обеспечивает одним верным решением. | Выдаёт приблизительный ответ задачи, однако это не всегда однозначный выбор. |
| Область применения | Подойдёт для задачи, если в ней критически важен точный результат. | Применим к широкому кругу задач, включая сложные и динамические проблемы. |
| Практичность | Непригоден для более сложных и масштабных задач. | Подходит для решения как малых, так и крупных задач. |
| Масштабируемость | Плохо справляется с увеличением количества условий из-за роста числа возможных вариантов. | Хорошо масштабируется при изменении задачи. |

# Заключение

## Вывод

Проделав работу, мы разработали два алгоритма, направленные на составление оптимального графика работы водителей и распределение маршрутов автобусов с учётом всех заданных условий и ограничений. Для каждого алгоритма была сформирована пара таблиц: одна с расписанием автобусов для каждого водителя, другая с графиком перерывов работников. В ходе сравнительного анализа были изучены временные сложности обоих подходов, а также были выявлены ключевые достоинства и недостатки каждого метода, включая их способность адаптироваться к изменяющимся условиям и ресурсным ограничениям.

# Список источников

1. 7 Ways to Loop Through a List in Python | LearnPython.com

<https://learnpython.com/blog/python-list-loop/>

1. Iterate over a list in Python – GeeksforGeeks

<https://www.geeksforgeeks.org/iterate-over-a-list-in-python/>

1. 4. More Control Flow Tools — Python 3.13.1 documentation

<https://docs.python.org/3/tutorial/controlflow.html>

1. Iterate Over a List of Lists in Python – GeeksforGeeks

<https://www.geeksforgeeks.org/iterate-over-a-list-of-lists-in-python/>

1. Задача о рюкзаке: разбор решения на Python

<https://pythonist.ru/zadacha-o-ryukzake-razbor-resheniya-na-python/>

1. Python и динамическое программирование на примере задачи о рюкзаке

<https://proglib.io/p/python-i-dinamicheskoe-programmirovanie-na-primere-zadachi-o-ryukzake-2020-02-04>

1. Genetic Algorithms – GeeksforGeeks

<https://www.geeksforgeeks.org/genetic-algorithms/>

1. Mastering Python Genetic Algorithms: A Complete Guide

<https://www.pythonpool.com/python-genetic-algorithm/>

1. Genetic Algorithm for Machine learning in Python – CodeSpeedy

<https://www.codespeedy.com/genetic-algorithm-for-machine-learning-in-python/>

1. Adaptive Mutation in Genetic Algorithm With Python Examples

<https://neptune.ai/blog/adaptive-mutation-in-genetic-algorithm-with-python-examples>

# Приложения

## GitHub

<https://github.com/dimarik1117/SIAOD_Kursovaya_5_semester>

## Алгоритм в лоб

from datetime import datetime, timedelta

import random

import pandas as pd

from tabulate import tabulate

NUM\_BUSES = 4

HALF\_NUM\_BUSES = 2

WEEKDAYS = ['Понедельник', 'Вторник', 'Среда', 'Четверг', 'Пятница']

WEEKENDS = ['Суббота', 'Воскресенье']

SHIFT\_8\_DURATION = timedelta(hours=8)

SHIFT\_12\_DURATION = timedelta(hours=12)

BREAK\_LONG\_END = timedelta(minutes=45)

BREAK\_MIN = timedelta(minutes=15)

BREAK\_MIN\_REST = timedelta(minutes=105)

BREAK\_INTERVAL = timedelta(minutes=120)

BREAK\_HOUR = timedelta(hours=1)

class Driver:

def \_\_init\_\_(self, driver\_id, shift\_type):

self.driver\_id = driver\_id

self.shift\_type = shift\_type

self.schedule = {day: [] for day in WEEKDAYS + WEEKENDS}

self.breaks = {day: [] for day in WEEKDAYS + WEEKENDS}

class Bus:

def \_\_init\_\_(self, bus\_id, bus\_index):

self.bus\_id = bus\_id

self.schedule = {day: [] for day in WEEKDAYS + WEEKENDS}

self.drivers = []

self.bus\_index = bus\_index

drivers\_count = 0

drivers = []

buses = [Bus(f"Автобус {i+1}", i) for i in range(NUM\_BUSES)]

for driver in drivers:

driver.schedule = {day: [] for day in WEEKDAYS + WEEKENDS}

driver.breaks = {day: [] for day in WEEKDAYS + WEEKENDS}

for bus in buses:

bus.schedule = {day: [] for day in WEEKDAYS + WEEKENDS}

def time\_intervals(start\_time, shift\_duration):

start = datetime.strptime(start\_time, "%H:%M")

end = start + shift\_duration

return start, end

def set\_breaks(start, end, break\_type):

breaks = []

if break\_type == "15\_мин":

current = start + BREAK\_MIN\_REST

break\_duration = BREAK\_MIN

while current + break\_duration < end:

breaks.append((current.strftime("%H:%M"), (current + break\_duration).strftime("%H:%M"), "15\_мин"))

current += BREAK\_INTERVAL

elif break\_type == "1\_час":

if start.strftime("%H:%M") == "09:00":

long\_start = datetime.strptime("14:15", "%H:%M")

long\_end = long\_start + BREAK\_LONG\_END

else:

long\_start = datetime.strptime("13:00", "%H:%M")

long\_end = long\_start + BREAK\_HOUR

if start <= long\_start < end:

breaks.append((long\_start.strftime("%H:%M"), long\_end.strftime("%H:%M"), "1\_час"))

elif long\_start <= start < long\_end:

breaks.append((start.strftime("%H:%M"), (start + BREAK\_HOUR).strftime("%H:%M"), "1\_час"))

return breaks

def set\_8\_drivers():

global drivers\_count

for bus\_index, bus in enumerate(buses):

if bus\_index < HALF\_NUM\_BUSES:

shift\_start\_time = "06:00"

else:

shift\_start\_time = "09:00"

drivers\_count += 1

driver = Driver(f"Водитель {drivers\_count} (8 час)", "8\_час")

drivers.append(driver)

for day in WEEKDAYS:

various\_start\_time = (datetime.strptime(shift\_start\_time, "%H:%M") + timedelta(minutes=(bus\_index % HALF\_NUM\_BUSES) \* 15)).strftime("%H:%M")

start, end = time\_intervals(various\_start\_time, SHIFT\_8\_DURATION)

if "09:00" in various\_start\_time:

breaks = set\_breaks(start, end, "1\_час")

else:

breaks = set\_breaks(start, end, "15\_мин")

driver.breaks[day] = breaks

driver.schedule[day].append((start, end, bus.bus\_id))

bus.schedule[day].append((start, end, driver.driver\_id))

def set\_12\_drivers():

global drivers\_count

rest\_12\_counter = {}

for \_, bus in enumerate(buses):

for \_ in range(3):

drivers\_count += 1

driver = Driver(f"Водитель {drivers\_count} (12 час)", "12\_час")

bus.drivers.append(driver)

drivers.append(driver)

rest\_12\_counter[driver.driver\_id] = 0

for bus in buses:

driver\_index = 0

for day in WEEKDAYS + WEEKENDS:

while rest\_12\_counter[bus.drivers[driver\_index].driver\_id] > 0:

rest\_12\_counter[bus.drivers[driver\_index].driver\_id] -= 1

driver\_index = (driver\_index + 1) % len(bus.drivers)

driver = bus.drivers[driver\_index]

if day in WEEKDAYS:

if bus.bus\_index < HALF\_NUM\_BUSES:

various\_start\_time = (datetime.strptime("14:00", "%H:%M") + timedelta(minutes=bus.bus\_index \* 15)).strftime("%H:%M")

else:

various\_start\_time = (datetime.strptime("17:00", "%H:%M") + timedelta(minutes=(bus.bus\_index - HALF\_NUM\_BUSES) \* 15)).strftime("%H:%M")

else:

various\_start\_time = (datetime.strptime("06:00", "%H:%M") + timedelta(minutes=bus.bus\_index \* 15)).strftime("%H:%M")

start, end = time\_intervals(various\_start\_time, SHIFT\_12\_DURATION)

if day in WEEKDAYS + WEEKENDS:

if start.hour == 6 and start.minute == 0:

breaks = set\_breaks(start, end, "1\_час")

else:

breaks = set\_breaks(start, end, "15\_мин")

else:

breaks = set\_breaks(start, end, "15\_мин")

driver.breaks[day] = breaks

driver.schedule[day].append((start, end, bus.bus\_id))

bus.schedule[day].append((start, end, driver.driver\_id))

rest\_12\_counter[driver.driver\_id] = 2

driver\_index = (driver\_index + 1) % len(bus.drivers)

def set\_weekend\_12\_drivers():

global drivers\_count

for \_, bus in enumerate(buses):

for day in WEEKENDS:

drivers\_count += 1

driver = Driver(f"Водитель {drivers\_count} (12 час)", f"12\_час Выходной {day}")

bus.drivers.append(driver)

drivers.append(driver)

start\_time = (datetime.strptime("18:00", "%H:%M") + timedelta(minutes=bus.bus\_index \* 15)).strftime("%H:%M")

start, end = time\_intervals(start\_time, SHIFT\_12\_DURATION)

breaks = set\_breaks(start, end, "15\_мин")

driver.breaks[day] = breaks

driver.schedule[day].append((start, end, bus.bus\_id))

bus.schedule[day].append((start, end, driver.driver\_id))

set\_8\_drivers()

set\_12\_drivers()

set\_weekend\_12\_drivers()

def extract\_breaks\_data(drivers):

long\_breaks\_data = {"Водитель": []}

short\_breaks\_data = {"Водитель": []}

for day in WEEKDAYS + WEEKENDS:

long\_breaks\_data[day] = []

short\_breaks\_data[day] = []

for driver in drivers:

long\_breaks\_day = []

short\_breaks\_day = []

for day in WEEKDAYS + WEEKENDS:

long\_breaks = [f"{b[0]}-{b[1]}" for b in driver.breaks[day] if b[2] == "1\_час"]

short\_breaks = [f"{b[0]}-{b[1]}" for b in driver.breaks[day] if b[2] == "15\_мин"]

long\_breaks\_day.append("; ".join(long\_breaks) if long\_breaks else "")

short\_breaks\_day.append("; ".join(short\_breaks) if short\_breaks else "")

if any(long\_breaks\_day):

long\_breaks\_data["Водитель"].append(driver.driver\_id)

for day, breaks in zip(WEEKDAYS + WEEKENDS, long\_breaks\_day):

long\_breaks\_data[day].append(breaks)

if any(short\_breaks\_day):

short\_breaks\_data["Водитель"].append(driver.driver\_id)

for day, breaks in zip(WEEKDAYS + WEEKENDS, short\_breaks\_day):

short\_breaks\_data[day].append(breaks)

long\_breaks\_df = pd.DataFrame(long\_breaks\_data)

short\_breaks\_df = pd.DataFrame(short\_breaks\_data)

return long\_breaks\_df, short\_breaks\_df

long\_breaks\_df, short\_breaks\_df = extract\_breaks\_data(drivers)

general\_schedule = []

for bus in buses:

bus\_row = [str(bus.bus\_id)] + [""] \* (len(WEEKDAYS + WEEKENDS))

general\_schedule.append(bus\_row)

uniq\_drivers = set()

for day in WEEKDAYS + WEEKENDS:

for \_, \_, driver\_id in bus.schedule.get(day, []):

uniq\_drivers.add(driver\_id)

for driver\_id in sorted(uniq\_drivers):

driver\_row = [str(driver\_id)]

for day in WEEKDAYS + WEEKENDS:

shifts = [f"{start.strftime('%H:%M')} - {end.strftime('%H:%M')}" for start, end, driver in bus.schedule.get(day, []) if driver == driver\_id]

driver\_row.append(", ".join(shifts) if shifts else "")

general\_schedule.append(driver\_row)

columns = ["Водитель"] + WEEKDAYS + WEEKENDS

bus\_schedule\_df = pd.DataFrame(general\_schedule, columns=columns)

long\_breaks\_filter = long\_breaks\_df[long\_breaks\_df.drop(columns=["Водитель"]).notna().any(axis=1)]

short\_breaks\_filter = short\_breaks\_df[short\_breaks\_df.drop(columns=["Водитель"]).notna().any(axis=1)]

long\_rows = [["Долгие перерывы"] + [""] \* (len(columns) - 1)] + long\_breaks\_filter.values.tolist()

short\_break\_rows = [["Короткие перерывы"] + [""] \* (len(columns) - 1)] + short\_breaks\_filter.values.tolist()

general\_breaks\_rows = long\_rows + short\_break\_rows

breaks\_final\_df = pd.DataFrame(general\_breaks\_rows, columns=columns)

def set\_custom\_index(df):

custom\_index = []

counter = 1

for row in df.itertuples(index=False):

if row[0] and (row[0].startswith("Автобус ") or row[0] in ["Долгие перерывы", "Короткие перерывы"]):

custom\_index.append("")

else:

custom\_index.append(counter)

counter += 1

df.index = custom\_index

return df

bus\_schedule\_df = set\_custom\_index(bus\_schedule\_df)

breaks\_final\_df = set\_custom\_index(breaks\_final\_df)

print(f"Всего водителей: {drivers\_count}")

print("\nРасписание автобусов:")

print(tabulate(bus\_schedule\_df, headers='keys', tablefmt='grid'))

print("\nРасписание перерывов:")

print(tabulate(breaks\_final\_df, headers='keys', tablefmt='grid'))

## Генетический алгоритм

POP\_SIZE = 50

GENERATIONS = 100

MUTATION\_RATE = 0.01

drivers\_count = 0

drivers = []

def random\_schedule():

global drivers\_count

rest\_12\_counter = {driver.driver\_id: 0 for driver in drivers if driver.shift\_type == "12\_час"}

schedule = []

for bus\_index, bus in enumerate(buses):

for day in WEEKDAYS:

shift\_type = "8\_час"

shift\_start\_time = "06:00" if bus\_index < HALF\_NUM\_BUSES else "09:00"

various\_start\_time = (datetime.strptime(shift\_start\_time, "%H:%M") + timedelta(minutes=(bus\_index % HALF\_NUM\_BUSES) \* 15)).strftime("%H:%M")

suit\_driver = [d for d in drivers if d.shift\_type == "8\_час"]

driver = random.choice(suit\_driver)

shift\_hours = SHIFT\_8\_DURATION

start, end = time\_intervals(various\_start\_time, shift\_hours)

breaks = set\_breaks(start, end, "1\_час" if "09:00" in various\_start\_time else "15\_мин")

driver.breaks[day] = breaks

driver.schedule[day].append((start, end, bus.bus\_id))

bus.schedule[day].append((start, end, driver.driver\_id))

schedule.append((bus.bus\_id, day, start, end, driver.driver\_id))

for bus\_index, bus in enumerate(buses):

for day in WEEKDAYS + WEEKENDS:

if day in WEEKENDS:

if bus\_index < HALF\_NUM\_BUSES:

shift\_type = "12\_час (Выходной Суббота)" if day == "Суббота" else "12\_час (Выходной Воскресенье)"

shift\_start\_time = "06:00"

else:

shift\_type = f"12\_час (Выходной {day})"

shift\_start\_time = "18:00"

elif day in WEEKDAYS:

shift\_type = "12\_час"

shift\_start\_time = "14:00" if bus\_index < HALF\_NUM\_BUSES else "17:00"

various\_start\_time = (datetime.strptime(shift\_start\_time, "%H:%M") + timedelta(minutes=(bus\_index % HALF\_NUM\_BUSES) \* 15)).strftime("%H:%M")

if shift\_type == "12\_час (Выходной Суббота)":

suit\_driver = [d for d in drivers if d.shift\_type == "12\_час (Выходной Суббота)"]

elif shift\_type == "12\_час (Выходной Воскресенье)":

suit\_driver = [d for d in drivers if d.shift\_type == "12\_час (Выходной Воскресенье)"]

elif shift\_type == "12\_час":

suit\_driver = [d for d in drivers if d.shift\_type == "12\_час" and rest\_12\_counter.get(d.driver\_id, 0) == 0]

if suit\_driver:

driver = random.choice(suit\_driver)

else:

continue

shift\_hours = SHIFT\_12\_DURATION

start, end = time\_intervals(various\_start\_time, shift\_hours)

if day in WEEKENDS and start.hour == 6 and start.minute == 0:

breaks = set\_breaks(start, end, "1\_час")

else:

breaks = set\_breaks(start, end, "15\_мин")

driver.breaks[day] = breaks

driver.schedule[day].append((start, end, bus.bus\_id))

bus.schedule[day].append((start, end, driver.driver\_id))

schedule.append((bus.bus\_id, day, start, end, driver.driver\_id))

return schedule

def init\_population(pop\_size):

population = []

for \_ in range(pop\_size):

individual = random\_schedule()

population.append(individual)

return population

def fitness(schedule):

penalty = 0

driver\_shifts = {driver.driver\_id: {day: [] for day in WEEKDAYS + WEEKENDS} for driver in drivers}

for \_, day, start, end, driver\_id in schedule:

driver\_shifts[driver\_id][day].append((start, end))

for driver\_id, shifts in driver\_shifts.items():

for day, day\_shifts in shifts.items():

day\_shifts.sort()

for i in range(len(day\_shifts)):

for j in range(i + 1, len(day\_shifts)):

if day\_shifts[i][1] > day\_shifts[j][0] and day\_shifts[i][0] < day\_shifts[j][1]:

penalty += 10

for driver\_id, shifts in driver\_shifts.items():

work\_days = sum(1 for day, day\_shifts in shifts.items() if day\_shifts)

if work\_days > 5:

penalty += 20

return penalty

def selection(population, fitness\_scores):

selected = random.choices(population, weights=[max(fitness\_scores) - score for score in fitness\_scores], k=2)

return selected

def crossover(parent1, parent2):

crossover\_point = random.randint(1, len(parent1) - 1)

child1 = parent1[:crossover\_point] + parent2[crossover\_point:]

child2 = parent2[:crossover\_point] + parent1[crossover\_point:]

return child1, child2

def mutation(individual, mutation\_rate):

for i in range(len(individual)):

if random.random() < mutation\_rate:

individual[i] = random\_schedule()[i]

return individual

def genetic\_algorithm(pop\_size, generations, mutation\_rate):

population = init\_population(pop\_size)

for \_ in range(generations):

fitness\_scores = [fitness(individual) for individual in population]

new\_population = []

for \_ in range(pop\_size // 2):

parent1, parent2 = selection(population, fitness\_scores)

child1, child2 = crossover(parent1, parent2)

new\_population.extend([mutation(child1, mutation\_rate), mutation(child2, mutation\_rate)])

population = new\_population

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

return best\_individual

def set\_drivers():

global drivers\_count

for \_, bus in enumerate(buses):

for \_ in range(3):

drivers\_count += 1

driver = Driver(f"Водитель {drivers\_count} (12 час)", "12\_час")

bus.drivers.append(driver)

drivers.append(driver)

for \_ in range(2):

drivers\_count += 1

driver = Driver(f"Водитель {drivers\_count} (8 час)", "8\_час")

bus.drivers.append(driver)

drivers.append(driver)

drivers\_count += 1

driver = Driver(f"Водитель {drivers\_count} (12 час)", "12\_час (Выходной Суббота)")

bus.drivers.append(driver)

drivers.append(driver)

drivers\_count += 1

driver = Driver(f"Водитель {drivers\_count} (12 час)", "12\_час (Выходной Воскресенье)")

bus.drivers.append(driver)

drivers.append(driver)

set\_drivers()

best\_schedule = genetic\_algorithm(pop\_size=POP\_SIZE, generations=GENERATIONS, mutation\_rate=MUTATION\_RATE)

def format\_table(schedule):

bus\_schedule = {bus.bus\_id: {day: [] for day in WEEKDAYS + WEEKENDS} for bus in buses}

driver\_schedule = {driver.driver\_id: {day: [] for day in WEEKDAYS + WEEKENDS} for driver in drivers}

for bus\_id, day, start, end, driver\_id in schedule:

bus\_schedule[bus\_id][day].append((start, end, driver\_id))

driver\_schedule[driver\_id][day].append((start, end, bus\_id))

return bus\_schedule, driver\_schedule

bus\_schedule, driver\_schedule = format\_table(best\_schedule)

def extract\_breaks\_data(drivers):

long\_breaks\_data = {"Водитель": []}

short\_breaks\_data = {"Водитель": []}

for day in WEEKDAYS + WEEKENDS:

long\_breaks\_data[day] = []

short\_breaks\_data[day] = []

for driver in drivers:

long\_breaks\_day = []

short\_breaks\_day = []

for day in WEEKDAYS + WEEKENDS:

if driver\_schedule[driver.driver\_id][day]:

long\_breaks = [f"{b[0]}-{b[1]}" for b in driver.breaks[day] if b[2] == "1\_час"]

short\_breaks = [f"{b[0]}-{b[1]}" for b in driver.breaks[day] if b[2] == "15\_мин"]

long\_breaks\_day.append("; ".join(long\_breaks) if long\_breaks else "")

short\_breaks\_day.append("; ".join(short\_breaks) if short\_breaks else "")

else:

long\_breaks\_day.append("")

short\_breaks\_day.append("")

if any(long\_breaks\_day):

long\_breaks\_data["Водитель"].append(driver.driver\_id)

for day, breaks in zip(WEEKDAYS + WEEKENDS, long\_breaks\_day):

long\_breaks\_data[day].append(breaks)

if any(short\_breaks\_day):

short\_breaks\_data["Водитель"].append(driver.driver\_id)

for day, breaks in zip(WEEKDAYS + WEEKENDS, short\_breaks\_day):

short\_breaks\_data[day].append(breaks)

long\_breaks\_df = pd.DataFrame(long\_breaks\_data)

short\_breaks\_df = pd.DataFrame(short\_breaks\_data)

return long\_breaks\_df, short\_breaks\_df

long\_breaks\_df, short\_breaks\_df = extract\_breaks\_data(drivers)

general\_schedule = []

for bus in buses:

bus\_id = bus.bus\_id

bus\_row = [str(bus\_id)] + [""] \* len(WEEKDAYS + WEEKENDS)

general\_schedule.append(bus\_row)

for driver\_id, driver\_shifts in driver\_schedule.items():

driver\_row = [str(driver\_id)]

include\_driver = False

for day in WEEKDAYS + WEEKENDS:

if day in driver\_shifts:

shifts = [f"{start.strftime('%H:%M')} - {end.strftime('%H:%M')}" for start, end, bus in driver\_shifts[day] if bus == bus\_id]

if shifts:

include\_driver = True

driver\_row.append(", ".join(shifts))

else:

driver\_row.append("")

else:

driver\_row.append("")

if include\_driver:

general\_schedule.append(driver\_row)

columns = ["Водитель"] + WEEKDAYS + WEEKENDS

bus\_schedule\_df = pd.DataFrame(general\_schedule, columns=columns)

long\_breaks\_filter = long\_breaks\_df[long\_breaks\_df.drop(columns=["Водитель"]).notna().any(axis=1)]

short\_breaks\_filter = short\_breaks\_df[short\_breaks\_df.drop(columns=["Водитель"]).notna().any(axis=1)]

long\_rows = [["Долгие перерывы"] + [""] \* (len(columns) - 1)] + long\_breaks\_filter.values.tolist()

short\_break\_rows = [["Короткие перерывы"] + [""] \* (len(columns) - 1)] + short\_breaks\_filter.values.tolist()

general\_breaks\_rows = long\_rows + short\_break\_rows

breaks\_final\_df = pd.DataFrame(general\_breaks\_rows, columns=columns)

def set\_custom\_index(df):

custom\_index = []

counter = 1

for row in df.itertuples(index=False):

if row[0] and (row[0].startswith("Автобус ") or row[0] in ["Долгие перерывы", "Короткие перерывы"]):

custom\_index.append("")

else:

custom\_index.append(counter)

counter += 1

df.index = custom\_index

return df

bus\_schedule\_df = set\_custom\_index(bus\_schedule\_df)

breaks\_final\_df = set\_custom\_index(breaks\_final\_df)

print(f"Всего водителей: {drivers\_count}")

print("\nРасписание автобусов:")

print(tabulate(bus\_schedule\_df, headers='keys', tablefmt='grid'))

print("\nРасписание перерывов:")

print(tabulate(breaks\_final\_df, headers='keys', tablefmt='grid'))