**Московский авиационный институт**

**(национальный исследовательский университет) «МАИ»**

**Институт №3** **—** «Системы управления, информатика и электроэнергетика»

**Кафедра 316 —** «Системное моделирование и автоматизированное проектирование»

Курсовой проект

«**Деление нечётких чисел**»

**Выполнил:**

студент группы М3О-118М-19

Довгерд Игнат

**Принял:**

д.т.н., профессор кафедры 304

Судаков Владимир Анатольевич

Москва 2019

print("Введите первое нечёткое число(тройку его параметров{a, x, b}, где a <= x, b <= x, x - точка максимума графика функции), разделяя его пробелами: ")

a = input().split()

print("Введите второе нечёткое число(тройку его параметров{a, x, b}, где a <= x, b <= x, x - точка максимума графика функции), разделяя его пробелами: ")

b = input().split()

A = []

B = []

for i in a:

A.append(float(i))

for i in b:

B.append(float(i))

C = [round((A[0] \* B[2] + A[0] \* B[1]) / (B[1] \* B[1]), 2), round((A[1] / B[1]), 2), round((A[1]\*B[0]+B[1]\*A[2])/(B[1] \* B[1]), 2)]

print("Первое нечёткое число(А):", A)

print("Второе нечёткое число(В):", B)

print("Деление нечётких чисел:", A, "/", B)

print("Ответ:", C)

**Метод ближайших соседей при классификации станций метро по количеству людей которые пьют чай или кофе**

**Подготовка данных**

import os

# Текущий каталог где находятся файлы

FILES\_DIR = "C:\\Projects\\KNN"

# stations.txt содержит данные по станциям на линиях метро, а также соседние станции

# Сначала заполняем матрицу смежности для вершин на линиях метро и соседних станций

metro\_stations = {}

with open(os.path.join(FILES\_DIR, "stations.txt"), "r", encoding = "utf8") as st:

m\_lines = []

# Заполняем станции по линиям метро

for line in st:

if "\*\*\*ПЕРЕХОДЫ\*\*\*" in line:

break

if ("\*\*\*" in line):

if m\_lines:

# конец линии

for i in range(len(m\_lines)):

if m\_lines[i] not in metro\_stations.keys():

metro\_stations[m\_lines[i]] = {}

for j in range(len(m\_lines)):

metro\_stations[m\_lines[i]][m\_lines[j]] = abs(i-j)

m\_lines=[]

else:

m\_lines.append(line.split("(")[0].strip().lower())

# Заполняем смежные станции

for line in st:

line = line.split("(")[0].strip().lower().replace("+", "/")

if "/" in line:

m\_lines = line.split("/")

for i in range(len(m\_lines)):

if m\_lines[i] not in metro\_stations.keys():

metro\_stations[m\_lines[i]] = {}

for j in range(len(m\_lines)):

metro\_stations[m\_lines[i]][m\_lines[j]] = 0

with open(os.path.join(FILES\_DIR, "metro\_by\_line.py"), "w", encoding = "utf-8") as f:

f.write("METRO = {}".format(metro\_stations))

# Далее на основе данных по линиям метро и смежным станциям строим глобальную матрицу

# применяем алгоритм Дейкстры для поиска оптимального пути между станциями

script\_path = os.path.join(FILES\_DIR, "Calculate\_distances\_matrix.py")

%run $script\_path $FILES\_DIR

# результат записываем в файл full\_dist\_matrix.py

**Алгоритмы классификации KNN**

import sys

sys.path.append(FILES\_DIR)

from full\_dist\_matrix import METRO

def get\_neighbors(k, metro):

# Код получения соседних станций (включая саму станцию) по k и названию метро из словаря METRO

neighbours = {}

for i in METRO[metro]:

if METRO[metro][i] <= k:

neighbours.update( { i : METRO[metro][i] } )

return neighbours

from classes\_data import cl\_data

def knn(neighbors):

# В neighbors содержится словарь с данными по соседним станциям и самой станции

# Изначально используется взвешенное голосование для "чай" и "кофе" в зависимости от расстояния до станции

votes\_tea = 0

votes\_coffee = 0

for i in neighbors:

if i not in cl\_data.keys():

continue

else:

votes\_tea += ((1000 - neighbors[i]) / 1000) \* cl\_data[i]['чай']

votes\_coffee += ((1000 - neighbors[i]) / 1000) \* cl\_data[i]['кофе']

if votes\_tea > votes\_coffee:

return("чай", votes\_tea, votes\_coffee)

elif votes\_tea < votes\_coffee:

return("кофе", votes\_tea, votes\_coffee)

return ("", votes\_tea, votes\_coffee)

# Альтернативный алгоритм

def knn2(neighbors):

# В neighbors содержится словарь с данными по соседним станциям и самой станции

# Подсчитываем количества людей для классов "чай" и "кофе"

# Нужно проверить особый случай когда кол-во людей для "чай" и "кофе" одинаковое

# тогда используем взвешенное голосование (для каждого класса ["чай" и "кофе"] подсчитываем сумму по всем станциям 1/квадрат расстояния)

t=0

c=0

for i in neighbors:

if i not in cl\_data.keys():

continue

t += cl\_data[i]['чай']

c += cl\_data[i]['кофе']

if t == c:

t=0

c=0

for i in neighbors:

if i not in cl\_data.keys():

continue

t += 1/((1+neighbors[i])\*\*2) \* cl\_data[i]['чай']

c += 1/((1+neighbors[i])\*\*2) \* cl\_data[i]['кофе']

if t > c:

return("чай", t, c)

elif t < c:

return("кофе", t, c)

return ("", t, c)

**Ввод данных и выполнение классификации**

k = 0

metro = ''

print("Для выхода нажмите <ENTER>")

while True:

print("Введите k:")

k = input()

if k == '':

break

print("Введите название метро:")

metro = input().strip().lower()

if metro == '':

break

try:

result = knn2(get\_neighbors(int(k), metro))

except KeyError:

print("Неправильно указана станция метро")

continue

except ValueError:

print("Число k указано неправильно")

continue

if not result[0]:

if result[1] == result[2] == 0:

print("По станции метро {0} отсутствуют данные".format(metro))

else:

print("На станции метро {0} c k={1} чай и кофе пьют поровну (чай = {2}, кофе = {3})"

.format(metro, k, result[1], result[2]))

if result[0] == "чай":

print("На станции метро {0} с k={1} пьют чай (чай = {2}, кофе = {3})".format(metro, k, result[1], result[2]))

if result[0] == "кофе":

print("На станции метро {0} с k={1} пьют кофе (чай = {2}, кофе = {3})".format(metro, k, result[1], result[2]))

**Центральность по посредничеству.**

**Загрузка графа друзей через VK API**

import vk\_api

import networkx as nx # библиотека графов

MAX\_DEEP = 5 # Максимальная глубина графа

MAX\_FRIENDS = 30 # Максимальное число друзей которое может обработать vk\_api для данного узла

VK\_LOGIN = 'kirainluck@gmail.com'

VK\_TOKEN = 'ce373d45c3f69d93022b12d3e9af02916e22becd049aaeeb40de1c19e7a65a3e0e58945b1d547beab9e0b'

VK\_IDS = [

559640345, 29759351, 120435774, 78517271, 55564167,

202514864, 79818398, 18214699, 25689500, 322913600,

32418429, 32784038, 55355150, 65817487, 76637395,

31710423, 557916962, 44907648

]

GRAPH\_FILENAME = os.path.join(FILES\_DIR, 'friends\_graph\_full.yaml')

def fill\_graph(vk, graph, id, deep=1):

"""

Функция загрузки графа

"""

max\_cnt = MAX\_FRIENDS

# обычно все друзья на первом уровне

if deep == 1:

max\_cnt = 500

try:

response = vk.friends.get(user\_id=id, count=max\_cnt)

except vk\_api.exceptions.ApiError as e:

print(f"VK api error on id {id} and deep {deep}\n {e}")

return

fr\_in\_lst = False

for fr\_id in response['items']:

graph.add\_edge(id, fr\_id)

if fr\_id in VK\_IDS:

fr\_in\_lst = True

if (not fr\_in\_lst) and (deep < MAX\_DEEP):

deep += 1

for fr\_id in response['items']:

fill\_graph(vk, graph, fr\_id, deep)

friends\_graph = None

if not os.path.exists(GRAPH\_FILENAME):

vk\_session = vk\_api.VkApi(login=VK\_LOGIN, token=VK\_TOKEN)

vk = vk\_session.get\_api()

friends\_graph = nx.Graph()

for id in VK\_IDS:

fill\_graph(vk, friends\_graph, id)

# Записываем результат в файл

nx.readwrite.nx\_yaml.write\_yaml(friends\_graph, GRAPH\_FILENAME)

else:

# Загружаем граф из файла

friends\_graph = nx.readwrite.nx\_yaml.read\_yaml(GRAPH\_FILENAME)

print(f"Number of nodes: {friends\_graph.number\_of\_nodes()}") # долго считает

**Вычисление центральности по посредничеству**

between\_cent = {key:value for (key, value) in nx.betweenness\_centrality(friends\_graph, k=10).items() if key in VK\_IDS}

between\_cent

import operator

between\_cent\_id = max(between\_cent.items(), key=operator.itemgetter(1))

**print(**between\_cent\_id) # id и значение центральности

(202514864, 0.002597780659422589)

# Центральный по посредничеству - Артем Осин

**Модель Шеллинга**

import numpy as np

import matplotlib.pyplot as plt

import cell\_happines\_calc as mf

# Shaling algorithm params

world\_size = 20

p\_blue = 0.45

p\_red = 0.45

num\_for\_happy = 2

max\_num\_iter = 100

# Generate initial world

Color\_map\_flat = np.zeros((world\_size\*\*2))

num\_elements = world\_size \*\* 2

num\_blue = round(num\_elements \* p\_blue)

num\_red = round(num\_elements \* p\_red)

Color\_map\_flat[0:num\_blue] = -0.6 # blue

Color\_map\_flat[num\_blue: num\_red+num\_blue] = 0.6

np.random.shuffle(Color\_map\_flat)

Color\_map = Color\_map\_flat.reshape((world\_size, world\_size))

# Iteratively move random unhappy to random empty

fig, ax = plt.subplots()

happiness\_map = np.zeros\_like(Color\_map)

for iteration in range(max\_num\_iter):

# Calculate happiness map

for i, \_ in enumerate(happiness\_map):

for j, \_ in enumerate(happiness\_map[i]):

happiness\_map[i, j] = mf.cell\_happines(Color\_map, i, j, num\_for\_happy)

unhappy\_x, unhappy\_y = np.where((happiness\_map == 0) & (Color\_map != 0))

empty\_x, empty\_y = np.where(Color\_map == 0)

if len(unhappy\_x) != 0:

unhappy\_to\_move\_idx = np.random.randint(0, len(unhappy\_x))

empty\_to\_move\_idx = np.random.randint(0, len(empty\_x))

Color\_map[empty\_x[empty\_to\_move\_idx], empty\_y[empty\_to\_move\_idx]] =\

Color\_map[unhappy\_x[unhappy\_to\_move\_idx ], unhappy\_y[unhappy\_to\_move\_idx]]

Color\_map[unhappy\_x[unhappy\_to\_move\_idx ], unhappy\_y[unhappy\_to\_move\_idx]] = 0

# Plot

ax.cla()

ax.set\_title("Iteration {}".format(iteration + 1))

plt.imshow(Color\_map, cmap='bwr', vmin=-1, vmax=1)

plt.pause(0.5)

else:

print("счастливых клеток не осталось после", iteration)

print("No unhappy cells!!!")

break

**Дерево решений ID3**

import pandas as pd

import math

from graphviz import Digraph

TARGET\_COLUMN = 'os'

data = pd.read\_csv("editor\_corrected.csv")

print(data.head())

**Алгоритм ID3**

class Node(object):

"""

Простое дерево

"""

def \_\_init\_\_(self):

self.value = None

self.edge = None # Название ребра, которое ведёт в данный узел

self.childs = []

def getEntropy(counts):

"""

Вычисление энтропии по последовательности с количествами различных классов

"""

entropy = 0

n = sum(counts)

for cnt in counts:

if cnt != 0:

entropy += -cnt/n \* math.log(cnt/n, 2)

return entropy

def getInformationGain(dataset, target\_name, attr\_name):

"""

Вычисление information gain для конкретного атрибута

"""

counts = dataset[target\_name].value\_counts()

n = sum(counts)

entropy = getEntropy(counts)

values = dataset[attr\_name].unique()

av\_info\_entropy = 0

for value in values:

partitioned = dataset[dataset[attr\_name]==value]

attr\_entropy = getEntropy(partitioned[target\_name].value\_counts())

av\_info\_entropy += (partitioned.shape[0]/n) \* attr\_entropy

return entropy - av\_info\_entropy

def getAttributeMaxInformationGain(dataset, target\_name, attributes):

"""

Вычисление атрибута с максимальным приростом информации

"""

attributes = [attr for attr in dataset if attr != target\_name]

gains = [getInformationGain(dataset, target\_name, attr\_name) for attr\_name in attributes]

return attributes[gains.index(max(gains))]

def getID3Tree(iteration, dataset, target\_name, attributes, root=Node()):

"""

Функция генерации ID3 дерева

"""

if iteration > 100:

print(dataset)

print(attributes)

return

if dataset[target\_name].nunique() == 1:

root.value = dataset[target\_name].unique()[0]

return root

if len(attributes) == 0:

root.value = dataset[target\_name].value\_counts().idxmax()

return root

bestAttrName = getAttributeMaxInformationGain(dataset, target\_name, attributes)

root.value = bestAttrName

attr\_values = dataset[bestAttrName].unique()

for value in attr\_values:

child\_node = Node()

child\_node.edge = value

root.childs.append(child\_node)

childDataSet = dataset[dataset[bestAttrName]==value]

if childDataSet.shape[0] == 0:

child.value = dataset[target\_name].value\_counts().idxmax()

else:

next\_lev\_attrs = [attr for attr in attributes if attr != bestAttrName]

getID3Tree(iteration + 1, childDataSet, target\_name, next\_lev\_attrs, child\_node)

return root

attributes = [attr for attr in data.columns if attr != TARGET\_COLUMN]

root = getID3Tree(0, data, TARGET\_COLUMN, attributes)

**Рисуем дерево**

dot = Digraph(comment='ID3')

def fill\_visual\_graph(dot, root):

"""

Заполняем граф для отображения

"""

dot.node(str(id(root)), root.value)

for node in root.childs:

dot.edge(str(id(root)), str(id(node)), node.edge)

fill\_visual\_graph(dot, node)

dot = Digraph(comment='ID3')

fill\_visual\_graph(dot, root)

print(dot)

Приложение

1. stations.txt

Пятницкое шоссе

Митино

Волоколамская

Мякинино

Строгино

Крылатское

Молодёжная

Кунцевская(синяя)

Славянский бульвар

Парк Победы

Киевская(Синяя с пересадочной)

Смоленская(Синяя)

Арбатская(С пересадками)

Площадь Революции(пересадка)

Курская

Бауманская

Электрозаводская

Семёновская

Партизанская

Измайловская

Первомайская

Щёлковская

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Синяя ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Кунцевская(голубая)

Пионерская

Филёвский парк

Багратионовская

Фили

Кутузовская

Международная(пересадка на деловой центр МЦК)

Выставочная

Киевская(Голубая с пересадочной)

Смоленская(Голубая)

Арбатская

Александровский сад

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Голубая ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Коммунарка

Ольховая

Прокшино

Филатов Луг

Саларьево

Румянцево

Тропарёво

Юго-западная

Проспект Вернадского

Университет

Воробьёвы горы

Спортивная

Фрунзенская

Парк Культуры(пересадка на кольцо)

Кропоткинская

Библиотека имени Ленина(с пересадочной)

Охотный ряд(пересадка)

Лубянка(пересадка)

Чистые пруды

Красные ворота

Комсомольская(пересадка)

Красносельская

Сокольники

Преображенская площадь

Черкизовская

Бульвар Рокоссовского

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Красная ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Ховрино

Беломорская

Речной вокзал

Водный стадион

Войковская

Сокол

Аэропорт

Динамо

Белорусская(зелень)

Маяковская

Тверская

Театральная

Новокузнецкая

Павелецкая

Автозаводская

Технопарк

Коломенская

Каширская

Кантемировская

Царицыно

Орехово

Домодедовская

Красногвардейская

Алма-Атинская

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Зелёная ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Планерная

Сходненская

Тушинская

Спартак

Щукинская

Октябрьское поле

Полежаевская

Беговая

Улица 1905 года

Баррикадная

Пушкинская

Кузнецкий Мост

Китай-город

Таганская

Пролетарская

Волгоградский проспект

Текстильщики

Кузьминки

Рязанский проспект

Выхино

Лермонтовский проспект

Жулебино

Котельники

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Фиолетовая ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Третьяковская

Марксистская

Площадь Ильича

Авиамоторная

Шоссе Энтузиастов

Перово

Новогиреево

Новокосино

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Жёлтая ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Косино

Улица Дмитриевского

Лухмановская

Некрасовка

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Розовая ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Селигерская

Верхние Лихоборы

Окружная

Петровско-Разумовская

Фонвизинская

Бутырская

Марьина Роща

Достоевская

Трубная

Сретенский бульвар

Чкаловская

Римская

Крестьянская застава

Дубровка

Кожуховская

Печатники

Волжская

Люблино

Братиславская

Марьино

Борисово

Шипиловская

Зябликово

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Салатовая ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Каширская

Варшавская

Каховская

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*censored ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Бунинская аллея

Улица Горчакова

Бульвар Адмирала Ушакова

Улица Скобелевская

Улица Старокачаловская

Лесопарковая

Битцевский парк

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*ярчесерого ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Новоясеневская

Ясенево

Тёплый стан

Коньково

Беляево

Калужская

Новые Черёмушки

Профсоюзная

Академическая

Ленинский проспект

Шаболовская

Октябрьская

Третьяковская

Китай-город

Тургеневская

Сухаревская

Проспект Мира

Рижская

Алексеевская

ВДНХ

Ботанический сад

Свиблово

Бабушкинская

Медведково

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Оранжевая ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Павелецкая

Таганская

Курская

Комсомольская

Проспект Мира

Новослободская

Белорусская

Краснопресненская

Киевская

Парк Культуры

Октябрьская

Добрынинская

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Кольцевая ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Савёловская

Петровский Парк

ЦСКА

Хорошёвская

Шепелиха

Деловой центр

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*censored ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Савёловская

Петровский Парк

ЦСКА

Хорошёвская

Шепелиха

Парк Победы

Минская

Ломоносовский проспект

Раменки

Мичуринский проспект

Озёрная

Говорово

Солнцево

Боровское шоссе

Новопеределкино

Рассказовка

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Желтая ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Алтуфьево

Бибирево

Отрадное

Владыкино

Петровско-Разумовская

Тимирязевская

Дмитровская

Савёловская

Менделеевская

Цветной бульвар

Чеховская

Боровицкая

Полянка

Серпуховская

Тульская

Нагатинская

Нагорная

Нахимовский проспект

Севастопольская

Чертановская

Южная

Пражская

Улица Академика Янгеля

Аннино

Бульвар Дмитрия Донского

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*Серая ветка всё\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*ПЕРЕХОДЫ\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Кунцевская(3+4) - 2

Парк Победы(3+8А) - 2

Новоясеневская/Битцевский Парк(12+6) - 2

Бульвар Дмитрия Донского+улица Старокачаловская(3+12) - 2

Красногвардейская/Зябликово(2+10) - 2

Каховская+Севастопольская(11А+3) - 2

Каширская(11А+2) - 2

Лермонтовский проспект/Косино (15+7) - 2

Крестьянская застава/Пролетарская (10+7) - 2

Киевская(4+3+5) - 3

Парк Культуры (5+1) - 2

Октябрьская (6+5) - 2

Добрынинская/Серпуховская (9+5)

Павелецкая (2+5) - 2

Таганская/Марксистская (5+7+8) - 3

Курская/Чкаловская(10+5+3) - 3

Комсомольская (1+5) - 2

Проспект Мира (6+5) - 2

Менделеевская/Новослободская (3+5) - 2

Белорусская (2+5) - 2

Баррикадная/Краснопресненская (7+5) - 2

Китай-город (6+7) - 2

Кузнецкий Мост/Лубянка (7+1) - 2

Новокузнецкая/Третьяковская(2+8+6) - 3

Арбатская/Боровицкая/Александровский сад/Библиотека имени Ленина(3+1+3+4) - 4

Площадь Революции/Охотный ряд/Театральная (3+2+1) - 3

Пушкинская/Тверская/Чеховская (7+2+3) - 3

Цветной Бульвар/Трубная (3+10) - 2

Сретенский Бульвар/Тургеневская/Чистые пруды (6+10+1) - 3

Петровско-Разумовская (10+3) - 2

Римская/Площадь Ильича (8+10) - 2

Полежаевская/Хорошёвская (8А+11+7) - 3

Выставочная/Деловой центр (11+4) - 2

Динамо/Петровский парк (2+8А+11) - 3

Савёловская (8А+11+9) - 3

ЦСКА(8А+11) - 2

Шепелиха(8А+11) – 2

1. classes\_data.py

cl\_data = {

"академическая": {"чай": 0, "кофе": 1},

"царицыно": {"чай": 2, "кофе": 0},

"окружная": {"чай":0, "кофе": 1},

"тимирязевская": {"чай": 0, "кофе": 1},

"молодежная": {"чай": 0, "кофе": 1},

"текстильщики": {"чай": 1, "кофе": 0},

"войковская": {"чай": 2, "кофе": 2},

"бабушкинская": {"чай": 1,"кофе": 0},

"отрадное": {"чай": 1, "кофе": 0},

#"коптево": {"чай": 1, "кофе": 0},

"орехово": {"чай": 1, "кофе": 0},

"ховрино": {"чай":1, "кофе": 0},

"бутырская": {"чай":0, "кофе": 1},

"комсомольская": {"чай": 0, "кофе": 1},

"речной вокзал": {"чай":0, "кофе": 1},

"сокол": {"чай":0, "кофе": 1},

}