import tkinter as tk  
from tkinter import ttk  
import math  
from collections import defaultdict  
from sklearn.cluster import KMeans  
import numpy as np  
  
# İlaç koordinatları ve eczane isimleri  
coordinates = {  
 "A": (-3, 0),  
 "B": (-1, 0),  
 "C": (-4, 0),  
 "D": (4, 0),  
 "E": (-3, 0),  
 "F": (3, 0),  
 "G": (-2, 0),  
 "H": (-1, 0),  
 "K": (5, 0),  
 "L": (-2, 0),  
 "Depo": (0, 0)  
}  
  
eczane\_isimleri = ["somun", "buyruk", "müşkül", "sevgi", "faruk", "derdest", "tarumar", "zeytin", "kibar", "orhun"]  
hafta\_gunleri = ["Pazartesi", "Salı", "Çarşamba", "Perşembe", "Cuma", "Cumartesi", "Pazar"]  
  
# Orijinal siparişler  
original\_orders = [  
 ["2A-2C-3B", "4K-5A-6H", "3C-4B-2K-3A", "4H-5L", "2B-2D-7K", "6A-7B-3G-1L", "1L-4G"],  
 ["3L-5C-9A", "3A", "2B-2D-7K", "2B-4K", "1C-5A", "5D", "3B-8H"],  
 ["6F-2A-3C", "4H-5L", "2D-5B-6A-7G", "5C-6L", "5L", "2G-3D", "4B-5L"],  
 ["4K-5A-3B", "3L-5C-9A", "3C-4H", "1L-6B-4C", "1B-9A", "2H-3F-1A", "3D-5G"],  
 ["3B-3F-7G", "4B-5A", "1L-4G", "2B-2D-5K", "3A", "3K-4L", "4K-5A-6H"],  
 ["6F-2A-3C", "4H-5L", "4H-5L", "2B-4A-8K", "3L-5G-9C-3A", "7G-5C-6A", "1L-6B-4C"],  
 ["2B-2D-5K", "3D-5G", "4C-5L", "4K-5A-6H", "5F", "3B-8H", "7F-5L"],  
 ["6H-6K", "2G-3D", "4K-5A-3B", "2B-4K", "1L-4B-4C", "2B-2D-7K", "5C-7L"],  
 ["4B-2A-5H", "2G-3F", "4H-5L", "6A-7B-3G-1L", "3A", "5C-6L", "4B-5L"],  
 ["5L", "2G-4F", "1B-5F-9A", "3B-3F-7G", "4K-5A-6H", "3L-5C", "2D-5B-6A-7G"]  
]  
  
# Yeni siparişler  
eczane\_orders = [  
 ["a1", "a2", "a3", "a4", "a5", "a6", "a7"],  
 ["b1", "b2", "b3", "b4", "b5", "b6", "b7"],  
 ["c1", "c2", "c3", "c4", "c5", "c6", "c7"],  
 ["d1", "d2", "d3", "d4", "d5", "d6", "d7"],  
 ["e1", "e2", "e3", "e4", "e5", "e6", "e7"],  
 ["f1", "f2", "f3", "f4", "f5", "f6", "f7"],  
 ["g1", "g2", "g3", "g4", "g5", "g6", "g7"],  
 ["h1", "h2", "h3", "h4", "h5", "h6", "h7"],  
 ["i1", "i2", "i3", "i4", "i5", "i6", "i7"],  
 ["j1", "j2", "j3", "j4", "j5", "j6", "j7"]  
]  
  
# Öklid mesafesi hesaplama fonksiyonu  
def calculate\_distance(point1, point2):  
 return math.sqrt((point1[0] - point2[0]) \*\* 2 + (point1[1] - point2[1]) \*\* 2)  
  
# Forkliftin sipariş toplarken kat edeceği mesafeyi hesaplayan fonksiyon  
def calculate\_total\_distance\_for\_order(order\_name):  
 for i, day\_orders in enumerate(eczane\_orders):  
 for j, order in enumerate(day\_orders):  
 if order == order\_name:  
 original\_order = original\_orders[i][j]  
 items = original\_order.split('-')  
 points = [coordinates["Depo"]]  
 for item in items:  
 qty = int(item[:-1])  
 medicine = item[-1]  
 points.extend([coordinates[medicine]] \* qty)  
 points.append(coordinates["Depo"])  
 total\_distance = 0  
 for k in range(len(points) - 1):  
 total\_distance += calculate\_distance(points[k], points[k + 1])  
 return total\_distance, points  
 return 0, []  
  
# FIFO sırasına göre kapasite kontrolü ve mesafe hesaplama fonksiyonu  
def calculate\_routes\_with\_capacity\_fifo(day\_index):  
 orders = eczane\_orders  
 capacity = 30  
 current\_capacity = 0  
 current\_route = []  
 total\_distance = 0  
 routes = []  
 collected\_orders = []  
 deliveries = []  
  
 for i in range(len(orders)):  
 order\_name = orders[i][day\_index]  
 order\_size = sum(int(item[:-1]) for item in original\_orders[i][day\_index].split('-'))  
  
 if current\_capacity + order\_size <= capacity:  
 current\_capacity += order\_size  
 collected\_orders.append(order\_name)  
 \_, points = calculate\_total\_distance\_for\_order(order\_name)  
 current\_route.extend(points[1:-1])  
 else:  
 current\_route = [coordinates["Depo"]] + current\_route + [coordinates["Depo"]]  
 routes.append(current\_route)  
 deliveries.append(collected\_orders)  
 for k in range(len(current\_route) - 1):  
 total\_distance += calculate\_distance(current\_route[k], current\_route[k + 1])  
 current\_route = []  
 collected\_orders = []  
 current\_capacity = 0  
 \_, points = calculate\_total\_distance\_for\_order(order\_name)  
 current\_route.extend(points[1:-1])  
 collected\_orders.append(order\_name)  
 current\_capacity = order\_size  
  
 if current\_route:  
 current\_route = [coordinates["Depo"]] + current\_route + [coordinates["Depo"]]  
 routes.append(current\_route)  
 deliveries.append(collected\_orders)  
 for k in range(len(current\_route) - 1):  
 total\_distance += calculate\_distance(current\_route[k], current\_route[k + 1])  
  
 return total\_distance, routes, deliveries  
  
# LIFO sırasına göre kapasite kontrolü ve mesafe hesaplama fonksiyonu  
def calculate\_routes\_with\_capacity\_lifo(day\_index):  
 orders = eczane\_orders  
 capacity = 30  
 current\_capacity = 0  
 current\_route = []  
 total\_distance = 0  
 routes = []  
 collected\_orders = []  
 deliveries = []  
  
 for i in range(len(orders) - 1, -1, -1):  
 order\_name = orders[i][day\_index]  
 order\_size = sum(int(item[:-1]) for item in original\_orders[i][day\_index].split('-'))  
  
 if current\_capacity + order\_size <= capacity:  
 current\_capacity += order\_size  
 collected\_orders.append(order\_name)  
 \_, points = calculate\_total\_distance\_for\_order(order\_name)  
 current\_route.extend(points[1:-1])  
 else:  
 current\_route = [coordinates["Depo"]] + current\_route + [coordinates["Depo"]]  
 routes.append(current\_route)  
 deliveries.append(collected\_orders)  
 for k in range(len(current\_route) - 1):  
 total\_distance += calculate\_distance(current\_route[k], current\_route[k + 1])  
 current\_route = []  
 collected\_orders = []  
 current\_capacity = 0  
 \_, points = calculate\_total\_distance\_for\_order(order\_name)  
 current\_route.extend(points[1:-1])  
 collected\_orders.append(order\_name)  
 current\_capacity = order\_size  
  
 if current\_route:  
 current\_route = [coordinates["Depo"]] + current\_route + [coordinates["Depo"]]  
 routes.append(current\_route)  
 deliveries.append(collected\_orders)  
 for k in range(len(current\_route) - 1):  
 total\_distance += calculate\_distance(current\_route[k], current\_route[k + 1])  
  
 return total\_distance, routes, deliveries  
  
# Paketlerin ağırlık merkezini hesaplayan fonksiyon  
def calculate\_centroid(order\_name):  
 for i, day\_orders in enumerate(eczane\_orders):  
 for j, order in enumerate(day\_orders):  
 if order == order\_name:  
 original\_order = original\_orders[i][j]  
 items = original\_order.split('-')  
 x\_sum = 0  
 y\_sum = 0  
 total\_qty = 0  
 for item in items:  
 qty = int(item[:-1])  
 medicine = item[-1]  
 x\_sum += coordinates[medicine][0] \* qty  
 y\_sum += coordinates[medicine][1] \* qty  
 total\_qty += qty  
 centroid\_x = x\_sum / total\_qty  
 centroid\_y = y\_sum / total\_qty  
 return (centroid\_x, centroid\_y)  
 return (0, 0)  
  
# KMeans ile paketleri sınıflandırma fonksiyonu  
def perform\_kmeans\_clustering(day\_index):  
 centroids = []  
 for i in range(len(eczane\_orders)):  
 centroid = calculate\_centroid(eczane\_orders[i][day\_index])  
 centroids.append(centroid)  
  
 data = np.array(centroids)  
 kmeans = KMeans(n\_clusters=3)  
 kmeans.fit(data)  
 labels = kmeans.labels\_  
 centers = kmeans.cluster\_centers\_  
  
 return labels, centers  
  
# KMeans sınıflarına göre kapasite kontrolü ve mesafe hesaplama fonksiyonu  
def calculate\_routes\_with\_kmeans(day\_index):  
 labels, \_ = perform\_kmeans\_clustering(day\_index)  
 clusters = defaultdict(list)  
  
 for i, label in enumerate(labels):  
 clusters[label].append(eczane\_orders[i][day\_index])  
  
 orders = eczane\_orders  
 capacity = 30  
 routes = []  
 total\_distance = 0  
 deliveries = []  
  
 for cluster in clusters.values():  
 current\_capacity = 0  
 current\_route = []  
 collected\_orders = []  
 for order\_name in cluster:  
 i = next(i for i, orders in enumerate(eczane\_orders) if order\_name in orders)  
 order\_size = sum(int(item[:-1]) for item in original\_orders[i][day\_index].split('-'))  
  
 if current\_capacity + order\_size <= capacity:  
 current\_capacity += order\_size  
 collected\_orders.append(order\_name)  
 \_, points = calculate\_total\_distance\_for\_order(order\_name)  
 current\_route.extend(points[1:-1])  
 else:  
 current\_route = [coordinates["Depo"]] + current\_route + [coordinates["Depo"]]  
 routes.append(current\_route)  
 deliveries.append(collected\_orders)  
 for k in range(len(current\_route) - 1):  
 total\_distance += calculate\_distance(current\_route[k], current\_route[k + 1])  
 current\_route = []  
 collected\_orders = []  
 current\_capacity = 0  
 \_, points = calculate\_total\_distance\_for\_order(order\_name)  
 current\_route.extend(points[1:-1])  
 collected\_orders.append(order\_name)  
 current\_capacity = order\_size  
  
 if current\_route:  
 current\_route = [coordinates["Depo"]] + current\_route + [coordinates["Depo"]]  
 routes.append(current\_route)  
 deliveries.append(collected\_orders)  
 for k in range(len(current\_route) - 1):  
 total\_distance += calculate\_distance(current\_route[k], current\_route[k + 1])  
  
 return total\_distance, routes, deliveries  
  
# Tkinter GUI kısmı  
class ForkliftApp:  
 def \_\_init\_\_(self, root):  
 self.root = root  
 self.root.title("Forklift Sipariş Toplama")  
  
 self.tab\_control = ttk.Notebook(root)  
 self.summary\_tab = ttk.Frame(self.tab\_control)  
 self.details\_tab = ttk.Frame(self.tab\_control)  
 self.distance\_tab = ttk.Frame(self.tab\_control)  
 self.fifo\_tab = ttk.Frame(self.tab\_control)  
 self.lifo\_tab = ttk.Frame(self.tab\_control)  
 self.kmeans\_combined\_tab = ttk.Frame(self.tab\_control)  
  
 self.tab\_control.add(self.summary\_tab, text='Özet')  
 self.tab\_control.add(self.details\_tab, text='Detaylar')  
 self.tab\_control.add(self.distance\_tab, text='Mesafe Hesaplama')  
 self.tab\_control.add(self.fifo\_tab, text='FIFO')  
 self.tab\_control.add(self.lifo\_tab, text='LIFO')  
 self.tab\_control.add(self.kmeans\_combined\_tab, text='KMeans')  
  
 self.tab\_control.pack(expand=1, fill="both")  
  
 self.create\_summary\_tab()  
 self.create\_details\_tab()  
 self.create\_distance\_tab()  
 self.create\_fifo\_tab()  
 self.create\_lifo\_tab()  
 self.create\_kmeans\_combined\_tab()  
  
 def create\_summary\_tab(self):  
 summary\_label = tk.Label(self.summary\_tab, text="Özet")  
 summary\_label.pack(padx=10, pady=10)  
  
 tree = ttk.Treeview(self.summary\_tab, columns=("Eczane", "Pazartesi", "Salı", "Çarşamba", "Perşembe", "Cuma", "Cumartesi", "Pazar"), show='headings')  
 tree.heading("Eczane", text="Eczane")  
 tree.heading("Pazartesi", text="Pazartesi")  
 tree.heading("Salı", text="Salı")  
 tree.heading("Çarşamba", text="Çarşamba")  
 tree.heading("Perşembe", text="Perşembe")  
 tree.heading("Cuma", text="Cuma")  
 tree.heading("Cumartesi", text="Cumartesi")  
 tree.heading("Pazar", text="Pazar")  
  
 for i, eczane in enumerate(eczane\_isimleri):  
 tree.insert("", "end", values=(eczane, \*eczane\_orders[i]))  
  
 tree.pack(expand=True, fill="both")  
  
 def create\_details\_tab(self):  
 details\_label = tk.Label(self.details\_tab, text="Detaylar")  
 details\_label.pack(padx=10, pady=10)  
  
 tree = ttk.Treeview(self.details\_tab, columns=("Eczane", "Pazartesi", "Salı", "Çarşamba", "Perşembe", "Cuma", "Cumartesi", "Pazar"), show='headings')  
 tree.heading("Eczane", text="Eczane")  
 tree.heading("Pazartesi", text="Pazartesi")  
 tree.heading("Salı", text="Salı")  
 tree.heading("Çarşamba", text="Çarşamba")  
 tree.heading("Perşembe", text="Perşembe")  
 tree.heading("Cuma", text="Cuma")  
 tree.heading("Cumartesi", text="Cumartesi")  
 tree.heading("Pazar", text="Pazar")  
  
 for i, eczane in enumerate(eczane\_isimleri):  
 tree.insert("", "end", values=(eczane, \*original\_orders[i]))  
  
 tree.pack(expand=True, fill="both")  
  
 map\_label = tk.Label(self.details\_tab, text="İlaç ve Eczane Haritası")  
 map\_label.pack(padx=10, pady=10)  
  
 map\_frame = ttk.Frame(self.details\_tab)  
 map\_frame.pack(padx=10, pady=10)  
  
 for i, (key, value) in enumerate(coordinates.items()):  
 label\_key = tk.Label(map\_frame, text=f"{key}:")  
 label\_key.grid(row=i // 2, column=(i % 2) \* 2, sticky="e")  
 label\_value = tk.Label(map\_frame, text=f"{value}")  
 label\_value.grid(row=i // 2, column=(i % 2) \* 2 + 1, sticky="w")  
  
 def create\_kmeans\_combined\_tab(self):  
 kmeans\_combined\_label = tk.Label(self.kmeans\_combined\_tab, text="KMeans")  
 kmeans\_combined\_label.pack(padx=10, pady=10)  
  
 self.kmeans\_day\_var = tk.StringVar(value="Pazartesi")  
 kmeans\_day\_menu = tk.OptionMenu(self.kmeans\_combined\_tab, self.kmeans\_day\_var, \*hafta\_gunleri)  
 kmeans\_day\_menu.pack(pady=5)  
  
 kmeans\_button = tk.Button(self.kmeans\_combined\_tab, text="KMeans Kümeleme Yap", command=self.calculate\_kmeans\_clustering)  
 kmeans\_button.pack(pady=10)  
  
 self.kmeans\_result = tk.Text(self.kmeans\_combined\_tab, width=50, height=15)  
 self.kmeans\_result.pack()  
  
 kmeansapp\_button = tk.Button(self.kmeans\_combined\_tab, text="KMeansApp Rotaları Hesapla", command=self.calculate\_kmeansapp\_routes)  
 kmeansapp\_button.pack(pady=10)  
  
 self.kmeansapp\_result = tk.Text(self.kmeans\_combined\_tab, width=50, height=15)  
 self.kmeansapp\_result.pack()  
  
 def create\_distance\_tab(self):  
 distance\_label = tk.Label(self.distance\_tab, text="Mesafe Hesaplama")  
 distance\_label.pack(padx=10, pady=10)  
  
 self.order\_name\_entry = tk.Entry(self.distance\_tab, width=20)  
 self.order\_name\_entry.pack(pady=5)  
  
 calculate\_distance\_button = tk.Button(self.distance\_tab, text="Mesafeyi Hesapla",  
 command=self.calculate\_distance)  
 calculate\_distance\_button.pack(pady=5)  
  
 self.distance\_result = tk.Text(self.distance\_tab, width=50, height=15)  
 self.distance\_result.pack()  
  
 def create\_fifo\_tab(self):  
 fifo\_label = tk.Label(self.fifo\_tab, text="FIFO: Kapasite ve Mesafe Hesaplama")  
 fifo\_label.pack(padx=10, pady=10)  
  
 self.fifo\_day\_var = tk.StringVar(value="Pazartesi")  
 fifo\_day\_menu = tk.OptionMenu(self.fifo\_tab, self.fifo\_day\_var, \*hafta\_gunleri)  
 fifo\_day\_menu.pack(pady=5)  
  
 calculate\_fifo\_routes\_button = tk.Button(self.fifo\_tab, text="Rotaları Hesapla",  
 command=self.calculate\_fifo\_routes)  
 calculate\_fifo\_routes\_button.pack(pady=5)  
  
 self.fifo\_result = tk.Text(self.fifo\_tab, width=50, height=15)  
 self.fifo\_result.pack()  
  
 def create\_lifo\_tab(self):  
 lifo\_label = tk.Label(self.lifo\_tab, text="LIFO: Kapasite ve Mesafe Hesaplama")  
 lifo\_label.pack(padx=10, pady=10)  
  
 self.lifo\_day\_var = tk.StringVar(value="Pazartesi")  
 lifo\_day\_menu = tk.OptionMenu(self.lifo\_tab, self.lifo\_day\_var, \*hafta\_gunleri)  
 lifo\_day\_menu.pack(pady=5)  
  
 calculate\_lifo\_routes\_button = tk.Button(self.lifo\_tab, text="Rotaları Hesapla",  
 command=self.calculate\_lifo\_routes)  
 calculate\_lifo\_routes\_button.pack(pady=5)  
  
 self.lifo\_result = tk.Text(self.lifo\_tab, width=50, height=15)  
 self.lifo\_result.pack()  
  
 def calculate\_distance(self):  
 order\_name = self.order\_name\_entry.get()  
 total\_distance, points = calculate\_total\_distance\_for\_order(order\_name)  
 self.distance\_result.delete('1.0', tk.END)  
 self.distance\_result.insert(tk.END, f"Toplam Mesafe: {total\_distance}\n")  
 self.distance\_result.insert(tk.END, f"Rota: {points}")  
  
 def calculate\_fifo\_routes(self):  
 selected\_day = self.fifo\_day\_var.get()  
 day\_index = hafta\_gunleri.index(selected\_day)  
 total\_distance, routes, deliveries = calculate\_routes\_with\_capacity\_fifo(day\_index)  
 self.fifo\_result.delete('1.0', tk.END)  
 self.fifo\_result.insert(tk.END, f"Toplam Mesafe: {total\_distance}\n")  
 for route, delivery in zip(routes, deliveries):  
 self.fifo\_result.insert(tk.END, f"Rota: {route}\nTeslimatlar: {delivery}\n")  
  
 def calculate\_lifo\_routes(self):  
 selected\_day = self.lifo\_day\_var.get()  
 day\_index = hafta\_gunleri.index(selected\_day)  
 total\_distance, routes, deliveries = calculate\_routes\_with\_capacity\_lifo(day\_index)  
 self.lifo\_result.delete('1.0', tk.END)  
 self.lifo\_result.insert(tk.END, f"Toplam Mesafe: {total\_distance}\n")  
 for route, delivery in zip(routes, deliveries):  
 self.lifo\_result.insert(tk.END, f"Rota: {route}\nTeslimatlar: {delivery}\n")  
  
 def calculate\_kmeans\_clustering(self):  
 selected\_day = self.kmeans\_day\_var.get()  
 day\_index = hafta\_gunleri.index(selected\_day)  
 labels, centers = perform\_kmeans\_clustering(day\_index)  
 result\_text = "KMeans Kümeleme Sonuçları:\n"  
 for i, label in enumerate(labels):  
 result\_text += f"Order {eczane\_orders[i][day\_index]} -> Cluster {label} at Centroid {centers[label]}\n"  
 result\_text += f"\nCluster Centers:\n{centers}"  
 self.kmeans\_result.delete('1.0', tk.END)  
 self.kmeans\_result.insert(tk.END, result\_text)  
  
 def calculate\_kmeansapp\_routes(self):  
 selected\_day = self.kmeans\_day\_var.get()  
 day\_index = hafta\_gunleri.index(selected\_day)  
 total\_distance, routes, deliveries = calculate\_routes\_with\_kmeans(day\_index)  
 self.kmeansapp\_result.delete('1.0', tk.END)  
 self.kmeansapp\_result.insert(tk.END, f"Toplam Mesafe: {total\_distance}\n")  
 for route, delivery in zip(routes, deliveries):  
 self.kmeansapp\_result.insert(tk.END, f"Rota: {route}\nTeslimatlar: {delivery}\n")  
  
  
root = tk.Tk()  
app = ForkliftApp(root)  
root.mainloop()