PV DE LIVRAISON

Date : 30/01/2024 **Version :** 1.0.0

Libellé de Livraison : TP THEORIE DES ARBRES (GROUPE BASIOU)

Description de la Livraison : Résolution d'un problème de livraison de colis par le biais du calcul d'un chemin optimal (c'est-à-dire le plus court possible) entre les points de livraisons choisis.

Cette application a été développée en utilisant le langage de programmation Python. Elle permet aux utilisateurs de spécifier une commune de départ, de sélectionner jusqu'à six communes de destination, et d'obtenir le trajet le plus court entre ces points.

Ce programme à été mené à bien grâce aux bibliothèques NetworkX, Tkinter et Matplotlib:

NetworkX est une bibliothèque permettant la modélisation de graphes, elle nous a permis de relationner les différentes communes ainsi que les distances les séparant, permettant donc le calcul optimal du chemin le plus court.

Tkinter quant à elle nous fournit une interface utilisateur agréable qui permettra la sélection de la commune de départ ainsi que celles qui seront livrées.

Matplotlib sera utilisée pour créer une représentation visuelle du graphe de réseau des communes en Guadeloupe, facilitant ainsi la compréhension et l'interaction de l'utilisateur avec les données géographiques.

Contenu de la Livraison :

Lien GitHub : https://github.com/TaliBSP/TP_TH_ARBRES

GRAPHE :

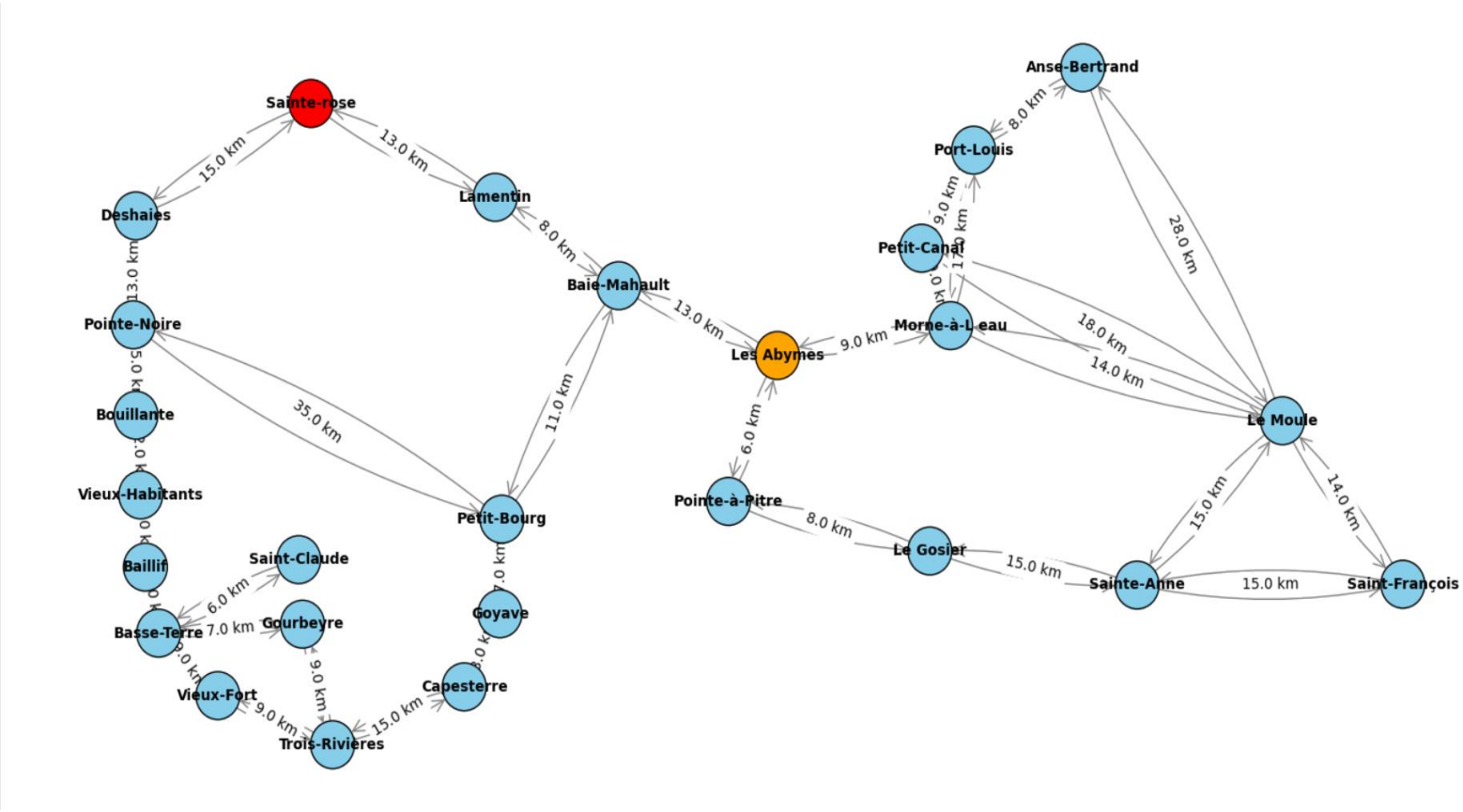

Sainte-rose
Deshaies
Lamentin
Baie-Mahault
Pointe-Noire
Les Abymes
Morne-à-L eau
Anse-Bertrand
Port-Louis
Petit-Canal
Le Moule
Bouillante
Vieux-Habitants
Petit-Bourg
Pointe-à-Pitre
Le Gosier
Sainte-Anne
Saint-François
Baillif
Saint-Claude
Basse-Terre
Gourbeyre
Goyave
Capesterre
Vieux-Fort
Trois-Rivières
15.0 km
13.0 km
8.0 km
13.0 km
9.0 km
8.0 km
9.0 km
11.0 km
28.0 km
18.0 km
14.0 km
35.0 km
11.0 km
6.0 km
8.0 km
15.0 km
14.0 km
15.0 km
15.0 km
6.0 km
7.0 km
9.0 km
9.0 km
15.0 km

TABLEAU DES DISTANCES :

	Sainte-Rose	Deshaies	Pointe-Noire	Bouillante	Vieux-Habitants	Baillif	Basse-Terre	Saint-Claude	Gourbeyre	Vieux-Fort	Trois-Rivières	Capesterre	Goyave	Petit-Bourg	Pointe-à-Pitre	Gosier	Sainte-Anne	Saint-François	Le Moule	Abymes	Morne-à-L'eau	Anse Bertrand	Port-Louis	Petit-Canal	Baie-Mahault	Lamentin
Sainte-Rose	0	15																								13
Deshaies	15		13																							
Pointe-Noire		13		15										35												
Bouillante			15		12																					
Vieux-Habitants				12		7																				
Baillif					7		4																			
Basse-Terre						4		6	7	9																
Saint-Claude							6																			
Gourbeyre							7				9															
Vieux-Fort							9				9															
Trois-Rivières									9	9		15														
Capesterre											15		13													
Goyave												13		7												
Petit-Bourg			35										7												11	
Pointe-à-Pitre																8				6						
Gosier															8			15	30							
Sainte-Anne																15			15	15						
Saint-François																30		15		14						
Le Moule																		15	14			14	28		18	
Abymes																6						9				13
Morne-à-L'eau																				14	9			17	9	
Anse Bertrand																				28				8		
Port-Louis																						17	8		9	
Petit-Canal																				18		9		9		
Baie-Mahault														11							13					8
Lamentin	13																								8	

Présentation et justification du choix des bibliothèques :

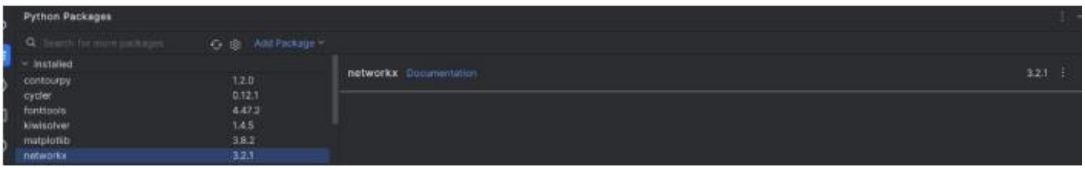
- NetworkX

NetworkX est une bibliothèque pour le langage de programmation Python qui est utilisé pour créer, manipuler et étudier la structure, la dynamique et les fonctions des réseaux de graphes complexes.

NetworkX fournit un moyen standardisé pour les scientifiques des données et les autres utilisateurs des mathématiques des graphes de collaborer, de construire, de concevoir, d'analyser et de partager des modèles de réseaux de graphes.

En tant que logiciel libre remarquable pour son évolutivité et sa portabilité, NetworkX a été largement adopté par les passionnés de Python.

C'est également le cadre graphique le plus populaire utilisé par les scientifiques des données, qui contribuent à un écosystème dynamique de paquets Python qui étendent NetworkX avec des fonctionnalités telles que l'algèbre linéaire numérique et le dessin.



- Pandas

Pandas est une bibliothèque écrite pour le langage de programmation Python permettant la manipulation et l'analyse des données.

Elle propose en particulier des structures de données et des opérations de manipulation de tableaux numériques et de séries temporelles.

- Matplotlib

Matplotlib est une bibliothèque du langage de programmation Python destinée à tracer et visualiser des données sous forme de graphiques.

Elle fournit également une API orientée objet, permettant d'intégrer des graphiques dans des applications, utilisant des outils d'interface graphique polyvalents tels que Tkinter, wxPython, Qt ou GTK.

CODE PYTHON :

```python
import networkx as nx
import matplotlib.pyplot as plt
from tkinter import Tk, Label, Listbox, Button, StringVar, messagebox, Toplevel, ttk
import tkinter as tk

1 usage
class GrapheGuadeloupe:
    def __init__(self):
        self.graph = nx.DiGraph()

    1 usage
    def ajouter_communes(self, communes):
        for commune, coordonnees in communes.items():
            self.graph.add_node(commune, pos=coordonnees)

    1 usage
    def ajouter_liaisons(self, liaisons):
        for liaison in liaisons:
            commune1, commune2, distance = liaison
            self.graph.add_edge(commune1, commune2, distance=float(distance))
            self.graph.add_edge(commune2, commune1, distance=float(distance))

    1 usage (1 dynamic)
    def afficher_graphe(self, commune_selectionnee=None, communes_selectionnees=None):
        pos = nx.get_node_attributes(self.graph, name: 'pos')
        edge_labels = {(edge[0], edge[1]): f"{edge[2]['distance']} km" for edge in self.graph.edges(data=True)}

        node_colors = ['red' if node == commune_selectionnee else 'orange' if node in communes_selectionnees else 'skyblue'
                       for node in self.graph.nodes()]
```

```python
        plt.figure(figsize=(12, 9))
        nx.draw(self.graph, pos, with_labels=True, node_size=1000, node_color=node_colors, font_size=10,
                font_color='black', font_weight='bold', edge_color='gray', font_family='sans-serif',
                connectionstyle="arc3,rad=0.1", arrows=True, arrowsize=20, arrowstyle='->', edgecolors='black')

        nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels)
        plt.show()

1 usage
class InterfaceUtilisateur:
    def __init__(self, fenetre, graphe):
        self.fenetre = fenetre
        self.fenetre.title("Sélection de Commune")
        self.graphe = graphe

        self.label = Label(fenetre, text="Sélectionnez une commune comme point de départ:", font=("Helvetica", 14))
        self.label.pack(pady=10)

        self.liste_communes = Listbox(fenetre, selectmode="single", font=("Helvetica", 12), height=15)
        for commune in graphe.graph.nodes():
            self.liste_communes.insert( index: 'end', *elements: commune)
        self.liste_communes.pack(pady=10)

        self.commune_selectionnee = None
        self.bouton_valider = Button(fenetre, text="Valider", command=self.valider_commune, font=("Helvetica", 12))
        self.bouton_valider.pack(pady=10)
```

```
1 usage
def valider_commune(self):
    self.commune_selectionnee = self.liste_communes.get(self.liste_communes.curselection())
    if self.commune_selectionnee:
        self.ouvrir_fenetre_selection()
    else:
        messagebox.showwarning( title: "Erreur",  message: "Veuillez sélectionner une commune.")

1 usage
def ouvrir_fenetre_selection(self):
    fenetre_selection = Toplevel(self.fenetre)
    fenetre_selection.title("Sélection des communes suivantes")

    label_selection = Label(fenetre_selection, text="Sélectionnez les autres communes:", font=("Helvetica", 14))
    label_selection.pack(pady=10)

    liste_communes_selection = Listbox(fenetre_selection, selectmode="multiple", font=("Helvetica", 12), height=15)
    for commune in self.graphe.graph.nodes():
        if commune != self.commune_selectionnee:
            liste_communes_selection.insert( index: 'end',  *elements: commune)
    liste_communes_selection.pack(pady=10)

    bouton_valider_selection = Button(fenetre_selection, text="Valider",
                                      command=lambda: self.valider_selection(liste_communes_selection),
                                      font=("Helvetica", 12))
    bouton_valider_selection.pack(pady=10)
```

```
1 usage
def valider_selection(self, liste_communes_selection):
    communes_selectionnees = [liste_communes_selection.get(i) for i in liste_communes_selection.curselection()]

    if len(communes_selectionnees) <= 6:
        if len(communes_selectionnees) == 0:
            messagebox.showwarning( title: "Erreur",  message: "Veuillez sélectionner au moins une commune.")
        else:
            self.graphe.afficher_graphe(self.commune_selectionnee, communes_selectionnees)

            # Utilisez la méthode de l'objet InterfaceUtilisateur
            self.afficher_tableau_chemin(self.commune_selectionnee, communes_selectionnees)
    else:
        messagebox.showwarning( title: "Erreur",  message: "Veuillez sélectionner au maximum 6 communes.")

1 usage
def afficher_tableau_chemin(self, commune_depart, communes_selectionnees):
    fenetre_tableau = Toplevel(self.fenetre)
    fenetre_tableau.title("Tableau du chemin le plus court")

    tableau = ttk.Treeview(fenetre_tableau)
    tableau["columns"] = ("Communes Visitées", "Distance")

    tableau.heading("Communes Visitées", text="Communes Visitées", anchor="center")
    tableau.heading("Distance", text="Distance", anchor="center")

    # Initialiser le chemin avec la commune de départ
    chemin_aller = [commune_depart]
    distance_totale_aller = 0
```

```
        # Itérer sur chaque commune sélectionnée pour construire le chemin aller
        for commune_arrivee in communes_selectionnees:
            if commune_arrivee != commune_depart:
                # Utiliser l'algorithme de Dijkstra pour trouver le chemin le plus court aller
                path_aller = nx.shortest_path(self.graphe.graph, source=chemin_aller[-1], target=commune_arrivee,
                                              weight='distance')

                # Ajouter le chemin aller et la distance aller à la liste
                for i in range(len(path_aller) - 1):
                    depart = path_aller[i]
                    arrivee = path_aller[i + 1]
                    distance = self.graphe.graph[depart][arrivee]['distance']
                    tableau.insert( parent: "",  index: "end", values=(depart, f"{distance} km"))
                    chemin_aller.extend(path_aller[1:])

                # Ajouter la distance du chemin aller
                distance_totale_aller += sum(
                    self.graphe.graph[path_aller[i]][path_aller[i + 1]]['distance'] for i in range(len(path_aller) - 1))

        # Utiliser l'algorithme de Dijkstra pour trouver le chemin le plus court retour
        path_retour = nx.shortest_path(self.graphe.graph, source=chemin_aller[-1], target=commune_depart,
                                       weight='distance')

        # Ajouter le chemin retour et la distance retour à la liste
        for j in range(len(path_retour) - 1):
            depart = path_retour[j]
            arrivee = path_retour[j + 1]
            distance_retour = self.graphe.graph[depart][arrivee]['distance']
            tableau.insert( parent: "",  index: "end", values=(depart, f"{distance_retour} km"))
```

```
            chemin_aller.extend(path_retour[1:])

        # Ajouter la distance du chemin retour
        distance_totale_retour = sum(
            self.graphe.graph[path_retour[j]][path_retour[j + 1]]['distance'] for j in range(len(path_retour) - 1))

        # Insérer le chemin complet aller-retour et la distance totale
        tableau.insert( parent: "",  index: "end", values=("Chemin Aller-Retour", ""))
        tableau.insert( parent: "",  index: "end", values=("Distance totale Aller", f"{distance_totale_aller} km"))
        tableau.insert( parent: "",  index: "end", values=("Distance totale Retour", f"{distance_totale_retour} km"))

        # Ajustements visuels
        tableau.column("#0", stretch=tk.NO, width=1)  # Désactive la première colonne (inutile)
        tableau.column("Communes Visitées", anchor="center", width=200)
        tableau.column("Distance", anchor="center", width=120)

        tableau.pack(padx=20, pady=20, fill="both", expand=True)

if __name__ == "__main__":
    guadeloupe = GrapheGuadeloupe()

    coordonnees_communes = {
        'Sainte-rose': (15.999, -61.1655),
        'Deshaies': (15.8544, -61.2756),
        'Pointe-Noire': (15.8522, -61.3823),
        'Bouillante': (15.8544, -61.4709),
        'Vieux-Habitants': (15.8583, -61.5490),
        'Baillif': (15.8622, -61.6197),
        'Basse-Terre': (15.8731, -61.6844),
        'Saint-Claude': (15.9888, -61.6124),
```

```
    'Gourbeyre': (15.9919, -61.6756),
    'Vieux-Fort': (15.9220, -61.7456),
    'Trois-Rivières': (16.0167, -61.7937),
    'Capesterre': (16.1254, -61.7370),
    'Goyave': (16.1549, -61.6654),
    'Petit-Bourg': (16.1565, -61.5727),
    'Baie-Mahault': (16.253, -61.3440),
    'Les Abymes': (16.384, -61.4124),
    'Pointe-à-Pitre': (16.3436, -61.5553),
    'Le Gosier': (16.5098, -61.6039),
    'Sainte-Anne': (16.6810, -61.6373),
    'Saint-François': (16.9003, -61.6365),
    'Le Moule': (16.801, -61.4763),
    'Anse-Bertrand': (16.636, -61.1305),
    'Port-Louis': (16.546, -61.2112),
    'Petit-Canal': (16.503, -61.3073),
    'Morne-à-L eau': (16.527, -61.3831),
    'Lamentin': (16.151, -61.2574),
}

liaisons = [
    ('Sainte-rose', 'Lamentin', 13),
    ('Lamentin', 'Baie-Mahault', 8),
    ('Les Abymes', 'Morne-à-L eau', 9),
    ('Morne-à-L eau', 'Port-Louis', 17),
    ('Port-Louis', 'Anse-Bertrand', 8),
    ('Anse-Bertrand', 'Le Moule', 28),
    ('Le Moule', 'Saint-François', 14),
    ('Saint-François', 'Sainte-Anne', 15),
    ('Sainte-Anne', 'Le Gosier', 15),
    ('Le Gosier', 'Pointe-à-Pitre', 8),
    ('Petit-Bourg', 'Goyave', 7),
    ('Goyave', 'Capesterre', 13),
    ('Capesterre', 'Trois-Rivières', 15),
    ('Trois-Rivières', 'Vieux-Fort', 9),
    ('Saint-Claude', 'Basse-Terre', 6),
    ('Basse-Terre', 'Baillif', 4),
    ('Baillif', 'Vieux-Habitants', 7),
    ('Vieux-Habitants', 'Bouillante', 12),
    ('Bouillante', 'Pointe-Noire', 15),
    ('Pointe-Noire', 'Deshaies', 13),
    ('Deshaies', 'Sainte-rose', 15),
    ('Pointe-Noire', 'Petit-Bourg', 35),
    ('Basse-Terre', 'Vieux-Fort', 9),
    ('Basse-Terre', 'Gourbeyre', 7),
    ('Baie-Mahault', 'Les Abymes', 13),
    ('Les Abymes', 'Pointe-à-Pitre', 6),
    ('Sainte-Anne', 'Le Moule', 15),
    ('Le Moule', 'Morne-à-L eau', 14),
    ('Petit-Canal', 'Morne-à-L eau', 9),
    ('Petit-Canal', 'Port-Louis', 9),
    ('Petit-Canal', 'Le Moule', 18),
    ('Gourbeyre', 'Trois-Rivières', 9),
    ('Petit-Bourg', 'Baie-Mahault', 11),
]
```

```
guadeloupe.ajouter_communes(coordonnees_communes)
guadeloupe.ajouter_liaisons(liaisons)

fenetre_principale = Tk()
fenetre_principale.geometry("1200x500")

# Passez l'objet guadeloupe à la classe InterfaceUtilisateur
interface = InterfaceUtilisateur(fenetre_principale, guadeloupe)

fenetre_principale.mainloop()
```

SITUATION :

Une entreprise de grande distribution effectue des livraisons de colis en Guadeloupe.

Lors de ses distributions, il lui faut trouver son point de départ, ainsi que les différentes communes où seront effectuées ces livraisons, tout en ayant un maximum de six communes par livraison.

Faisant attention à sa rentabilité, elle souhaite pouvoir calculer, pour chaque livraison, le chemin le plus court possible permettant de rallier les communes devant être livrées.

PROBLEME :

- Saisie du point de départ par l'utilisateur
- Saisie des points de livraisons en tenant compte du maximum de 6 communes
- Calcul du cycle optimal

SOLUTIONS :

Algorithme développé en langage Python représentant un graphe orienté de la Guadeloupe incluant toutes ses communes, leurs liaisons et leurs distances en KM.

Interface utilisateur permettant la sélection de la commune de départ, puis des communes de livraison.

- Affichage en surbrillance des communes sélectionnées
- Calcul du chemin optimal à l'aide de l'algorithme du Voyageur de commerce (TSP), cet algorithme consiste à trouver le circuit hamiltonien de plus faible poids dans un graphe orienté.

Lien GitHub : https://github.com/TaliBSP/TP_TH_ARBRES