PV DE LIVRAISON

Date : 29/01/2024

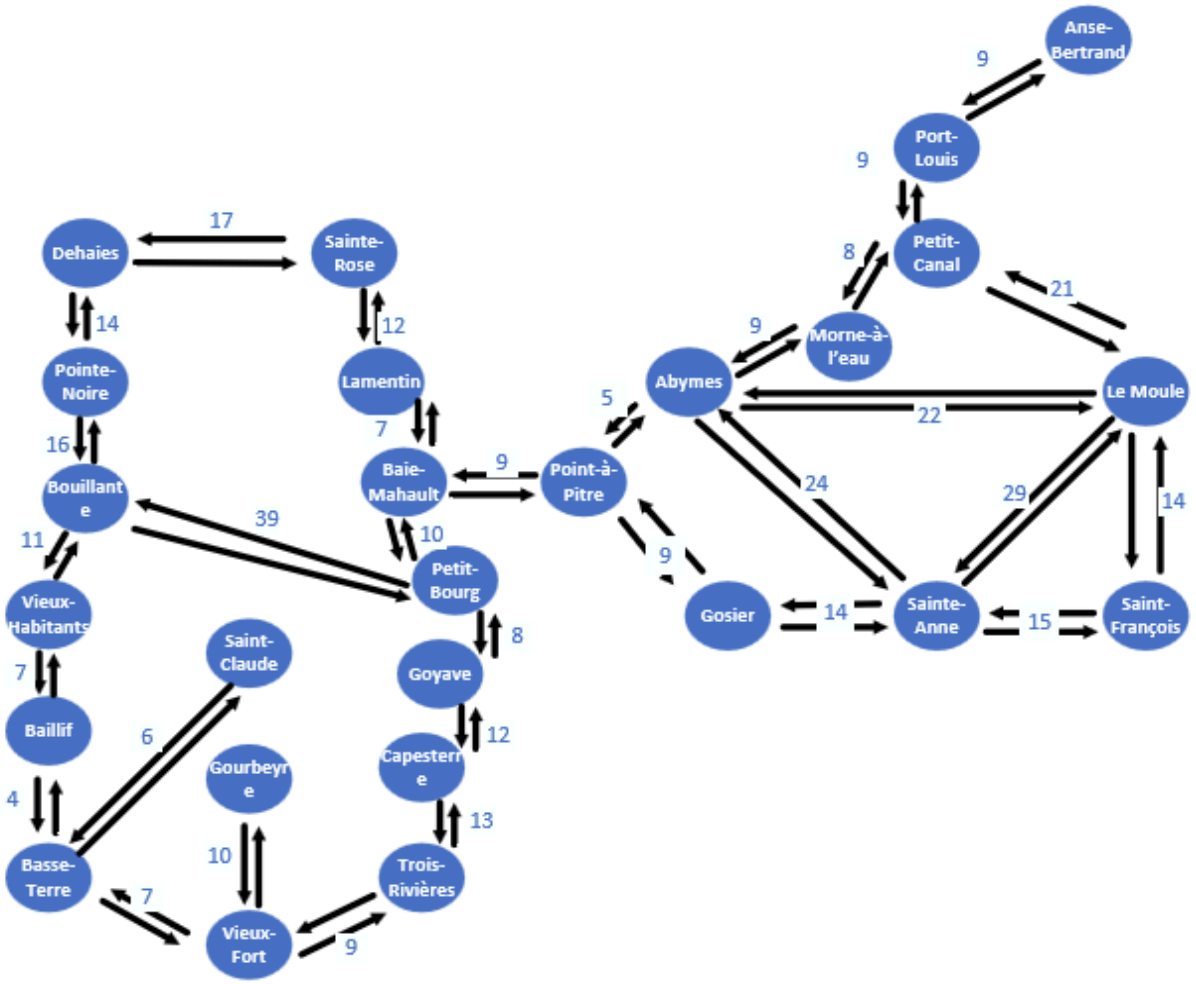
Version : 1.0.0

Libellé de livraison : Résoudre un problème de distribution de colis en calculant le chemin optimum (le plus court) dans le cadre d'une distribution de colis. Vous avez une liste de communes à distribuer, votre programme détermine la tournée optimale.

Description de la livraison : On a pour mission de résoudre un problème de distribution de colis, en cherchant le chemin optimal (le plus court) pour une distribution efficace entre différentes communes. En utilisant Python avec les bibliothèques NetworkX et Tkinter, on peut mettre en place un programme qui calcule la tournée optimale pour la distribution des colis. En exploitant NetworkX, on peut modéliser les relations entre les communes et calculer les chemins les plus courts, tandis que Tkinter permet de créer une interface utilisateur pour interagir avec le programme de manière conviviale. Avec cette approche, le programme sera capable d'optimiser la logistique de la distribution de colis en déterminant les itinéraires les plus efficaces entre les différentes destinations.

Contenu :

LE GRAPHE :



Modélisation python :

Pour travailler avec des graphes orientés dont les arêtes ont un coût en Python, la bibliothèque la plus couramment utilisée est NetworkX. NetworkX est une bibliothèque Python dédiée à la création, la manipulation et la visualisation de structures de réseau complexes.

MultiDiGraph (Graphe Orienté Multigraphe) :

MultiDiGraph est une extension de DiGraph qui permet d'avoir plusieurs arêtes entre les mêmes paires de nœuds.

Chaque arête peut avoir des attributs distincts, ce qui est utile lorsque des relations multiples entre les mêmes nœuds sont nécessaires.

ORIGINES	DESTINATIONS	COÛTS
PAP	Abymes	5
Abymes	PAP	5
PAP	BaieMahault	9
BaieMahault	PAP	9
PAP	Gosier	9
Gosier	PAP	9
Gosier	SainteAnne	14
SainteAnne	Gosier	14
Abymes	SainteAnne	24
SainteAnne	Abymes	24
SainteAnne	SaintFrancois	15
SaintFrancois	SainteAnne	15
SaintFrancois	LeMoule	14
LeMoule	SaintFrancois	14
LeMoule	SainteAnne	28
SainteAnne	LeMoule	28
LeMoule	MorneAleau	13
MorneAleau	LeMoule	13
LeMoule	AnseBertrand	28
AnseBertrand	LeMoule	28
LeMoule	Abymes	22
Abymes	LeMoule	22
LeMoule	PetitCanal	21
PetitCanal	LeMoule	21
PetitCanal	MorneAleau	8
MorneAleau	PetitCanal	8
PetitCanal	PortLouis	9
PortLouis	PetitCanal	9
MorneAleau	Abymes	9
Abymes	MorneAleau	9
PortLouis	AnseBertrand	9
AnseBertrand	PortLouis	9
BaieMahault	Lamentin	7
Lamentin	BaieMahault	7
SainteRose	Lamentin	12
Lamentin	SainteRose	12
SainteRose	Deshaies	17
Deshaies	SainteRose	17
BaieMahault	PetitBourg	10
PetitBourg	BaieMahault	10
PetitBourg	Bouillante	39
Bouillante	PetitBourg	39
Goyave	PetitBourg	8
PetitBourg	Goyave	8
Goyave	Capesterre	12
Capesterre	Goyave	12
Capesterre	TroisRivieres	13
TroisRivieres	Capesterre	13
TroisRivieres	Gourbeyre	8

Gourbeyre	TroisRivieres	8
BasseTerre	Gourbeyre	6
Gourbeyre	BasseTerre	6
BasseTerre	VieuxFort	7
VieuxFort	BasseTerre	7
BasseTerre	SaintClaude	6
SaintClaude	BasseTerre	6
Baillif	BasseTerre	4
BasseTerre	Baillif	4
TroisRivieres	VieuxFort	9
VieuxFort	TroisRivieres	9
Baillif	VieuxHabitants	7
VieuxHabitants	Baillif	7
Bouillante	VieuxHabitants	11
VieuxHabitants	Bouillante	11
Bouillante	PointeNoire	16
PointeNoire	Bouillante	16
Deshaies	PointeNoire	14
PointeNoire	Deshaies	14

Ex 2 : Rechercher les bibliothèques de manipulation de graphes disponible en pyhton et gratuite et justifier votre choix

- Recherche des Bibliothèques Python pour les Graphes Orientés avec Coûts

voici un tableau comparatif :

Bibliothèque	**Avantages**	**Inconvénients**
igraph	- Facile à utiliser - Prise en charge des graphes orientés et non orientés - Gestion des coûts sur les arcs	- Documentation limitée - Fonctionnalités avancées limitées
PyGraphviz	- Utilise Graphviz pour la visualisation - Prise en charge des graphes orientés et non orientés - Gestion des coûts sur les arcs	- Installation complexe - Documentation limitée
Graph-tool	- Prise en charge des graphes orientés et non orientés - Gestion des coûts sur les arcs - Nombreuses fonctionnalités avancées	- Installation complexe - Documentation limitée
NetworkX	-Facilité d'utilisation : Syntaxe simple et conviviale. -Documentation complète : Ressources bien documentées. Flexibilité : Convient pour une variété de tâches liées aux graphes.	-Performances : Peut être moins performant pour des graphes massifs par rapport à des bibliothèques spécialisées.
Cytoscape	-Visualisation : Excellentes fonctionnalités de visualisation avec une interface graphique conviviale. -Analyse spécifique des réseaux biologiques : Adapté pour l'analyse des réseaux biologiques.	-Manipulation programmatique limitée : Principalement axé sur la visualisation et l'analyse spécifique des réseaux biologiques.

Gephi	-Visualisation : Excellentes fonctionnalités de visualisation avec une interface graphique conviviale.	-Manipulation programmatique limitée : Principalement axé sur la visualisation, moins sur la manipulation programmatique complexe.
iGraph	-Multi-langage : Disponible dans plusieurs langages, y compris Python. -Fonctionnalités avancées : Offre des fonctionnalités avancées pour l'analyse de graphes.	-Certaines fonctionnalités spécifiques : Certaines fonctionnalités peuvent être spécifiques à un langage particulier.

Sachant que par la suite, il faudra coder des algorithmes permettant de donner l'ordre du graphe, de parcourir ce graphe orienté et un algorithme permettant de déterminer le chemin le plus court entre 2 sommets. Quelle bibliothèque permettra de faire toutes ces choses, Pour la plupart de ces tâches (ordre du graphe, parcours du graphe orienté, algorithme pour déterminer le chemin le plus court entre deux sommets), NetworkX en Python serait un choix approprié. NetworkX est une bibliothèque complète et bien établie qui offre une large gamme de fonctionnalités pour travailler avec des graphes.

Charger le graphe à partir d'un fichier :

Méthode : nx.read_weighted_edgelist

Exemple : G = nx.read_weighted_edgelist('nom_fichier.txt', create_using=nx.DiGraph())
Déterminer le nombre de nœuds (ordre) du graphe :

Méthode : G.order()

Parcourir le graphe : order_of_graph = G.order()

Parcours en largeur (BFS) :

Méthode : nx.bfs_edges

Exemple : bfs_order = list(nx.bfs_edges(G, source='nom_noeud_depart'))

Parcours en profondeur (DFS) :

Méthode : nx.dfs_edges

Exemple : dfs_order = list(nx.dfs_edges(G, source='nom_noeud_depart'))

Exemple :

```python
import networkx as nx

# Charger le graphe à partir d'un fichier texte
G = nx.read_weighted_edgelist('votre_fichier.txt', create_using=nx.DiGraph())

# Déterminer le nombre de nœuds (ordre) du graphe
order = G.order()
print(f"L'ordre du graphe est : {order}")

# Parcourir le graphe en largeur (BFS)
bfs_order = list(nx.bfs_edges(G, source='A'))
print(f"Parcours en largeur du graphe : {bfs_order}")

# Parcourir le graphe en profondeur (DFS)
dfs_order = list(nx.dfs_edges(G, source='A'))
print(f"Parcours en profondeur du graphe : {dfs_order}")

# Algorithme de Dijkstra pour le chemin le plus court
shortest_path = nx.shortest_path(G, source='A', target='C', weight='weight')
shortest_distance = nx.shortest_path_length(G, source='A', target='C', weight='weight'
print(f"Chemin le plus court de A à C : {shortest_path}")
print(f"Distance la plus courte de A à C : {shortest_distance}")
```

Notre code en python :

```python
import networkx as nx
import matplotlib.pyplot as plt

# Fonction pour lire un graphe à partir d'un fichier d'arêtes pondérées
def read_weighted_edge_list(file_path):
    G = nx.DiGraph()  # Utiliser un graphe orienté
    with open(file_path, 'r') as file:
        for line in file:
            elements = line.split()
            if len(elements) == 3:
                source, destination, weight = elements
                weight = int(weight)
                G.add_edge(source, destination, weight=weight)
    return G

# Spécifiez le chemin de votre fichier
file_path = r'C:\Users\lolo7\OneDrive\Documents\Informatique\3eme année\TH_Arbres_Graphes\TP\graphe.txt'

# Appel de la fonction pour lire le graphe
graph = read_weighted_edge_list(file_path)

# Calcul des positions avec une disposition en cercle
pos = nx.circular_layout(graph)

# Affichage du graphe
nx.draw(graph, pos, with_labels=True, node_size=700, node_color="skyblue", font_size=8, font_color="black", font_weight="bold", edge_color="gray", linewidths=1

# Affichage des poids des arêtes
edge_labels = nx.get_edge_attributes(graph, 'weight')
nx.draw_networkx_edge_labels(graph, pos, edge_labels=edge_labels)

# Affichage du graphe
plt.show()
```

```
C:\Users\lolo7>pip install networkx
Requirement already satisfied: networkx in c:\users\lolo7\appdata\local\programs\python\python311\lib\site-packages (3.2
.1)
```

Contexte :

Lors d'une tournée de distribution de colis, il nous faut dans un premier temps saisir le point de départ. Une fois ce point de départ saisi, il nous faudra ensuite saisir les différents points de livraison, sachant que le total des points de livraison doit être supérieur inférieur ou égal à 6. Enfin nous devrons afficher le circuit comportant le coût le moins élever qui nous permettra d'effectuer cette tournée de distribution de colis.

Problème posé : Algorithme permettant de réalisés une tournée de distribution de colis à l'aide d'un graphe orienté en effectuant le circuit de poids le plus court.

Solution :

Pour résoudre ce problème, vous pouvez utiliser l'algorithme du "voyageur de commerce" (TSP - Traveling Salesman Problem), qui consiste à trouver le circuit hamiltonien (un cycle qui passe par chaque nœud exactement une fois) de poids minimum dans un graphe orienté pondéré. Pour rendre cette algorithme plus performant, nous utiliserons les méthodes intégrées à la bibliothèque Networkx telles que « nx.shortest_path» et «nx.shortest_path_lenght » pour avoir le chemin le plus court et le cout associé.

Lien github : https://github.com/Raidover/TP_TH_ARBRES_Les3Phenix