

PROCES-VERBAL DE LIVRAISON

Date : 30 Janvier 2024

Version : 1.0

Libellé de livraison : Projet Optimisation du chemin groupe ZASAR (SARMEZAN, ZAMI)

Description de la livraison :

- | | | |
|----|-----------------------------|---------|
| 1. | Lien GitHub du projet | Page 1 |
| 2. | TP numéro 1..... | Page 2 |
| 3. | TP numéro 2..... | Page 5 |
| 4. | TP numéro 3 | Page 8 |
| 5. | Code python du Graphe | Page 10 |

Contenu :

Lien GitHub :

<https://github.com/NoemiSmz/Projet-d-optimisation-du-chemin.git>

TP1 de Théorie des arbres

Groupe ZASAR : ZAMI Christophe, SARMEZAN Noémi

Graphe de représentation des différents chemins de la Guadeloupe :

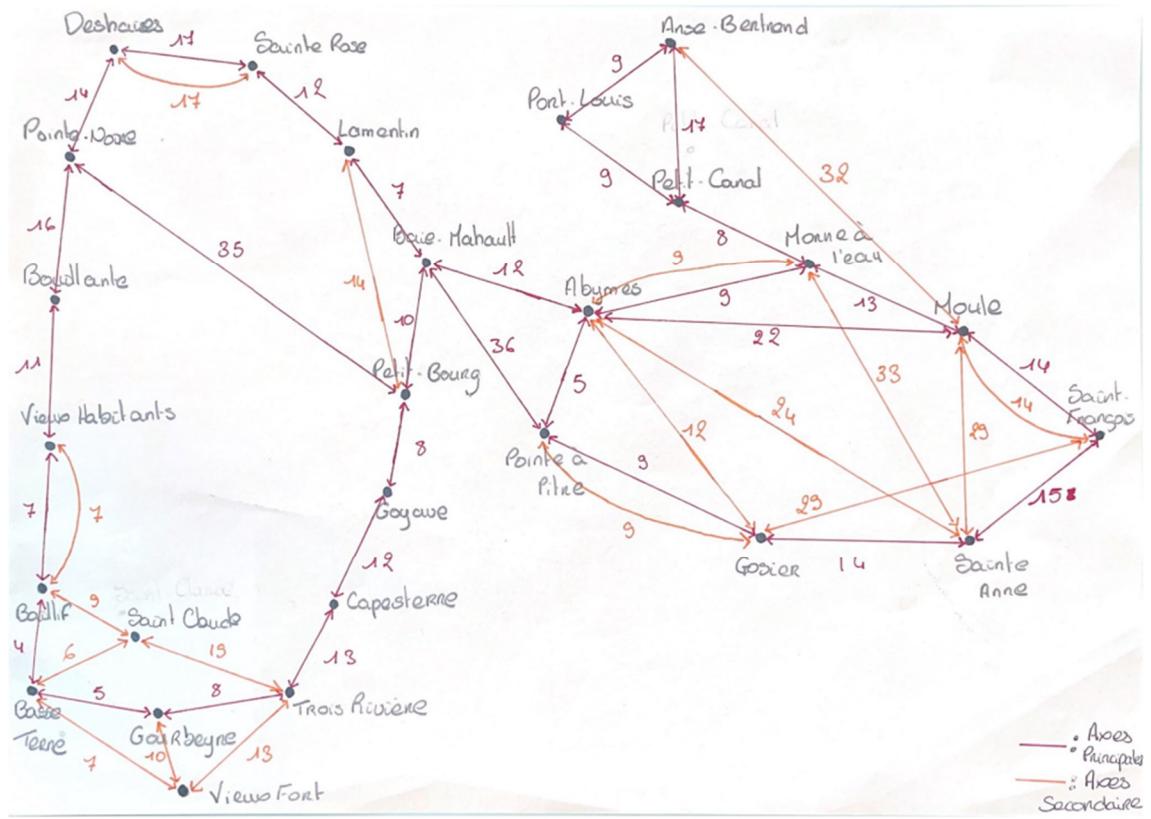


Tableau correspondant au graphe :

Abymes	Morme-a-l'eau loule st-anne Gosier	9 22 24 12
Anse-BERTRAND	Moule port-louis petit-canal	32 9 17
BAIE-MAHAULT	abymes petit-bourg lamentin point-a-pitre	12 10 7 36
BAILLIF	st claudie vieux-habitants basse terre	9 7 4
BASSE-TERRRE	bailif vieux port	4 7

BOUILLANTE	vieux-habitants point-noir	11 16
Capesterre-belle-eau	goyave Trois rivière	12 13
Le moule	saint-françois saint-anne morme-a-l'eau	14 29 13
Deshales	st-rose pointe-noir	17 14
Goyave	capesterre petit bourg	12 8

LAMENTIN	bais-mahaut saint-rose petit bourg	7 12 14
M-A-L	moule petit-canal saint-anne abymes	13 8 33 9

petit-bourg	lamentin bais-mahaut goyave point-noir	14 10 8 35
port-louis	anse-bertrand petit-canal	9 9
pointe-noire	petit-bourg dehail bouillante	35 14 16
saint-anne	saint françois goyave moule abymes morme-a-l'eau	15 14 29 24 33
saint-rose	lamentin dehail	12 17
trois-rivière	capesterre vieux port goubeure st-claudie	13 13 8 19
vieux-fort	Trois rivière goubeure basse terre	13 10 7
vieux-habitants	bailif bouillante	7 11
gosier	pointe-a-pitre abymes saint-anne saint-françois	9 12 14 29
goubeure	Trois rivière vieux habitant basse terre	8 10 5
petit-canal	m-a-l port lout anse bertrand	8 9 11
P-A-P	abymes bima gosier	5 36 9

saint-françois	saint-anne moule gosier	15 14 14

Modélisation Python Graphe :

```
> Users > sarme > OneDrive > Bureau > code.py > ...
1 import networkx as nx
2 import matplotlib.pyplot as plt
3
4 # Cr ation d'un graphe dirig 
5 graph = nx.DiGraph()
6
7 # Ajout des axes principaux
8 main_axes = [
9     ("Port-Louis", "Anse-Bertrand"),
10    ("Port-Louis", "Petit-Canal"),
11    ("Petit-Canal", "Anse-Bertrand"),
12    ("Petit-Canal", "Morne à l'eau"),
13    ("Morne à l'eau", "Moule"),
14    ("Moule", "Saint-Fran ois"),
15    ("Saint-Fran ois", "Sainte-Anne"),
16    ("Sainte-anne", "Gosier"),
17    ("Gosier", "Pointe à pitre"),
18    ("Pointe à pitre", "Abymes"),
19    ("Abymes", "Baie Mahault"),
20    ("Pointe à pitre", "Baie Mahault"),
21    ("Baie mahault", "Petit Bourg"),
22    ("Petit Bourg", "Goyave"),
23    ("Goyave", "Capesterre"),
24    ("Capesterre", "Trois rivi re"),
25    ("Trois rivi re", "Gourbeyre"),
26    ("Gourbeyre", "Basse Terre"),
27    ("Basse Terre", "Baillif"),
28    ("Baillif", "Vieux Habitants"),
29    ("Vieux Habitants", "Bouillante"),
30    ("Bouillante", "Pointe Noire"),
31    ("Pointe Noire", "Deshaises"),
32    ("Deshaises", "Sainte rose"),
33    ("Sainte rose", "Lamentin"),
34    ("Lamentin", "Baie Mahault"),
35
36 ]
37
38 secondary_axes = [
39     ("Anse Bertrand", "la Moule"),
40     ("Moule", "Saint-Fran ois"),
41     ("Saint-Fran ois", "Gosier"),
42     ("Moule", "Sainte Anne"),
43     ("Sainte Anne", "Morne à l'eau"),
44     ("Sainte anne", "Abymes"),
45     ("Abymes", "Gosier"),
46     ("Gosier", "Pointe à pitre"),
47     ("Pointe à pitre", "Baie Bourg"),
48     ("Baie Bourg", "Deshaises"),
49     ("Deshaises", "Sainte Rose"),
50     ("Vieux Habitants", "Baillif"),
51     ("Saint Claude", "Basse Terre"),
52     ("Saint Claude", "Trois Rivi re"),
53     ("Trois Rivi re", "Vieux Fort"),
54     ("Basse Terre", "Vieux Fort"),
55     ("Vieux Fort", "Gourbeyre"),
56     ("Gourbeyre", "Vieux Fort"),
57 ]
58
59 # Ajout des routes au graphe
60 graph.add_edges_from(main_axes, weight=2) # Poids plus  lev  pour les axes principaux
61 graph.add_edges_from(secondary_axes, weight=1)
62
63 # Affichage du graphe
64 pos = nx.spring_layout(graph) # Ajustez la disposition selon vos besoins
65 nx.draw(graph, pos, with_labels=True, font_weight='bold', node_size=700, node_color='skyblue', font_size=8)
66
67 # Affichage des poids des routes
68 labels = nx.get_edge_attributes(graph, 'weight')
69 nx.draw_networkx_edge_labels(graph, pos, edge_labels=labels)
70
71 plt.show()
72
```

TP2 : Théorie des Arbres

Groupe ZASAR : ZAMI Christophe, SARMEZAN Noémi

Question 2 :

Les bibliothèques Python sont utilisées pour modéliser les données souhaitées sous forme de graphe. Le code va construire le graphe en lui ajoutant ses nœuds et ses arêtes grâce à des fonctions.

Python possède de nombreux Framework différents et propose un large choix d'options pour ceux-ci. Parmi eux, on retrouve :

- ❖ **Tkinter** : Une bibliothèque pour l'interface graphique utilisateur en Open Source. La prise en main est très simple et il à la spécificité d'être préinstallé en Python. Cependant, Tkinter est une ancienne bibliothèque qui présente des limites.
- ❖ **Kivy** : A été conçu pour la création d'interfaces utilisateurs avec des applications multi-touch. Elle permet aux concepteurs de coder et de déployer sur plusieurs plates-formes. Cependant le prise en main est un peu compliquée et nécessite du temps.
- ❖ Nous avons trouvé ces deux bibliothèques Python mais avons préféré utiliser **NetworkX** pour sa simplicité d'installation et les options utiles qu'elle propose pour une utilisation large dans le domaine des graphes. La version 2.2 de NetworkX permet de créer, manipuler et analyser les réseaux de façon optimal. Elle dispose aussi d'algorithmes classiques de théorie des graphes tel que Dijkstra, PageRank et SlmRank...

A) Installation de la bibliothèque NetworkX sur PyCharm

```
C:\Users\chris\OneDrive\Documents\Universiter\L3\semestre 1\Projet rinaldo> pip install Networkx
Collecting Networkx
  Using cached networkx-3.2.1-py3-none-any.whl (1.6 MB)
Installing collected packages: Networkx
Successfully installed Networkx-3.2.1

[notice] A new release of pip available: 22.3 -> 23.3.2
[notice] To update, run: python.exe -m pip install --upgrade pip
C:\Users\chris\OneDrive\Documents\Universiter\L3\semestre 1\Projet rinaldo>
```

B) Chargement du graphe

```
activate_this.py
    sommets = re.findall(r'Nom Sommet: (\w+)', texte_page)
    poids_arretes = re.findall(r'Poids Arête: (\d+)', texte_page)

    donnees_graphe['sommets'].extend(sommets)

    # Construction des arêtes avec les poids
    if len(sommets) > 1 and len(poids_arretes) > 0:
        arretes = [(sommets[i], sommets[i + 1], int(poids_arretes[i])) for i in range(len(sommets) - 1)]
        donnees_graphe['arretes'].extend(arretes)

    return donnees_graphe

def regarde():
    g = nx.DiGraph()
    return g

graphe.py
import networkx as nx
data = []
def initialisation(chemin_fichier):
    donnees_graphe = {'sommets': [], 'arretes': []}
    with open(chemin_fichier, 'rb') as fichier_pdf:
        lecteur_pdf = PyPDF2.PdfFileReader(fichier_pdf)
        nombre_pages = lecteur_pdf.numPages

    for numero_page in range(nombre_pages):
        page = lecteur_pdf.getPage(numero_page)
        texte_page = page.extractText()

        # Utilisation d'expressions régulières pour extraire les informations
        sommets = re.findall(r'Nom Sommet: (\w+)', texte_page)
        poids_arretes = re.findall(r'Poids Arête: (\d+)', texte_page)

        donnees_graphe['sommets'].extend(sommets)

        # Construction des arêtes avec les poids
        if len(sommets) > 1 and len(poids_arretes) > 0:
            arretes = [(sommets[i], sommets[i + 1], int(poids_arretes[i])) for i in range(len(sommets) - 1)]
            donnees_graphe['arretes'].extend(arretes)
```

C) Parcourir le graphe

Ce code Python réalise un parcours en profondeur (DFS) d'un graphe créé à l'aide de la bibliothèque NetworkX. En spécifiant un nœud de départ, le code explore récursivement les voisins de chaque nœud, affichant le nom du nœud en cours de visite. Cette approche permet de parcourir le graphe de manière profonde avant de revenir en arrière, assurant ainsi que tous les chemins sont explorés en profondeur avant d'explorer d'autres branches.

D) Utilisation de la bibliothèque

I- Création du graphe

- ❖ Premièrement, après avoir importé NetworkX, on crée le graphe à l'aide de la fonction : ‘nx.DiGraph’ (graphe orienté).

- ❖ On déclare le nombre de sommets de ce graphe avec la commande « **nomGraphe.add_nodes_from(range(6))** »
- ❖ Ensuite, on y ajoute les nœuds et les arrêtes en utilisant « **nomGraphe.add_edge('Moule', 'Petit-Canal')** » pour les liens (arrêtes) et « **nomGraphe.node('Moule', 'Petit-Canal')** » pour les nœuds avec lien.
- ❖ Pour ajouter un poids à un lien on utilise « **nomGraphe.add_edge('Moule', 'Petit-Canal', poids = 6)** »

II- Visualisation du graphe

Pour afficher le graphe il faut utiliser la commande « **nx.draw_networkx(nomGraphe)** »

TP3 : Théorie Arbres

Groupe ZASAR : ZAMI Christophe, SARMEZAN Noémie

Analyse du problème.

I- Expression des besoins

Nous cherchons à réaliser un programme permettant à une société de livraison de mettre en place de façon informatisée, les parcours de livraison de ses chauffeurs. En s'appuyant sur le graphe orienté des différents axes routiers de la Guadeloupe, le programme devra trouver le chemin le plus favorable et le plus court en fonction des points de livraison rentrés par le chauffeur-livreur. Le programme fournira en sortie le graphe des axes en mettant en avant le chemin que devra suivre le chauffeur ainsi que l'ordre dans lequel il devra se rendre aux différents points de livraison.

II- Analyse de l'existant

Précédemment, lors du TP1, nous avions déjà réalisé le code Python qui, en s'appuyant sur la bibliothèque Network, affiche le graphe orienté des axes routiers principaux et secondaires reliant les 26 communes de la Guadeloupe. Le graphe affiche également la distance de l'axe reliant chaque commune.

Description du programme :

- 1) Importation de NetworkX
- 2) Création du graphe
- 3) Création d'un premier tableau regroupant tous les axes routiers principaux et leurs distances en kilomètres
- 4) Création d'un second tableau regroupant tous les axes routiers secondaires et leurs distances en kilomètres
- 5) Ajout des routes au graphe
- 6) Affichage du graphe
- 7) Affichage des poids des routes

III- Choix de la solution Informatique

Afin de réaliser le projet souhaité, en nous appuyant sur le programme déjà créé, nous allons créer un algorithme de calcul afin de trouver le chemin le plus court.

Description du programme :

- 1) Demande à l'utilisateur de saisir son point de départ
- 2) Demande à l'utilisateur de saisir le nombre de villes à livrer
- 3) Demande à l'utilisateur de saisir les différentes villes de livraison (inférieurs ou égales à 6 villes)
- 4) Création d'une structure permettant à l'utilisateur de saisir les villes
- 5) Utilisation d'un algorithme permettant de calculer le chemin le plus court en additionnant la distance de chaque axe des villes demandées
- 6) Affichage du chemin sur le graphe

Code Python du Graphe

```
1 import networkx as nx
2 import matplotlib.pyplot as plt
3 from scipy.optimize import linear_sum_assignment
4 G= nx.DiGraph()
5
6 main_axes = [
7     ("Port-Louis", "Anse-Bertrand", 9),
8     ("Anse-Bertrand", "Port-Louis", 9),
9     ("Port-Louis", "Petit-Canal", 9),
10    ("Petit-Canal", "Port-Louis", 9),
11    ("Petit-Canal", "Anse-Bertrand", 17),
12    ("Anse-Bertrand", "Petit-Canal", 17),
13    ("Petit-Canal", "Morne à l'eau", 8),
14    ("Morne à l'eau", "Petit-Canal", 8),
15    ("Morne à l'eau", "Moule", 13),
16    ("Moule", "Morne à l'eau", 13),
17    ("Moule", "Saint-François", 14),
18    ("Saint-François", "Moule", 14),
19    ("Saint-François", "Sainte-Anne", 15),
20    ("Sainte-Anne", "Saint-François", 15),
21    ("Sainte-Anne", "Gosier", 14),
22    ("Gosier", "Sainte-Anne", 14),
23    ("Gosier", "Pointe à pitre", 9),
24    ("Pointe à pitre", "Gosier", 9),
25    ("Pointe à pitre", "Abymes", 5),
26    ("Abymes", "Pointe à pitre", 5),
27    ("Abymes", "Baie-Mahault", 12),
28    ("Baie-Mahault", "Abymes", 12),
29    ("Pointe à pitre", "Baie-Mahault", 9),
30    ("Baie-Mahault", "Pointe à pitre", 9),
31    ("Baie-Mahault", "Petit Bourg", 10),
32    ("Baie-Mahault", "Petit Bourg", 10),
33    ("Petit Bourg", "Goyave", 8),
34    ("Goyave", "Petit Bourg", 8),
35    ("Goyave", "Capesterre", 12),
36    ("Capesterre", "Goyave", 12),
37    ("Capesterre", "Trois rivières", 13),
38    ("Trois rivières", "Capesterre", 13),
39    ("Trois rivières", "Gourbeyre", 8),
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
    ("Trois rivières", "Capesterre", 13),
    ("Trois rivières", "Gourbeyre", 8),
    ("Gourbeyre", "Trois rivières", 8),
    ("Gourbeyre", "Basse Terre", 5),
    ("Basse Terre", "Gourbeyre", 5),
    ("Basse Terre", "Baillif", 4),
    ("Baillif", "Basse Terre", 4),
    ("Baillif", "Vieux Habitants", 7),
    ("Vieux Habitants", "Bouillante", 11),
    ("Bouillante", "Vieux Habitants", 11),
    ("Bouillante", "Pointe Noire", 16),
    ("Pointe Noire", "Bouillante", 16),
    ("Pointe Noire", "Deshaies", 14),
    ("Deshaies", "Pointe Noire", 14),
    ("Deshaies", "Sainte rose", 17),
    ("Sainte rose", "Deshaies", 17),
    ("Lamentin", "Sainte rose", 12),
    ("Lamentin", "Baie-Mahault", 7),
    ("Lamentin", "Baie-Mahault", 7),
    ("Anse-Bertrand", "Moule", 32),
    ("Moule", "Anse-Bertrand", 32),
    ("Saint-François", "Moule", 17),
    ("Moule", "Saint-François", 17),
    ("Saint-François", "Gosier", 31),
    ("Gosier", "Saint-François", 31),
    ("Moule", "Sainte-Anne", 29),
    ("Sainte-Anne", "Moule", 29),
    ("Sainte-Anne", "Morne à l'eau", 33),
    ("Morne à l'eau", "Sainte-Anne", 33),
    ("Sainte-anne", "Abymes", 24),
    ("Abymes", "Sainte-anne", 24),
    ("Abymes", "Gosier", 12),
    ("Gosier", "Abymes", 12),
    ("Gosier", "Pointe à pitre", 12),
    ("Pointe à pitre", "Gosier", 12),
    ("Lamentin", "Petit Bourg", 14),
    ("Petit Bourg", "Lamentin", 14),
    ("Deshaies", "Sainte Rose", 22),
    ("Sainte Rose", "Deshaies", 22),
```

```

    \ Deshaies , "Sainte Rose" , 22),
76    ("Sainte Rose", "Deshaises", 22),
77    ("Vieux Habitants", "Baillif", 13),
78    ("Baillif", "Vieux Habitants", 13),
79    ("Saint Claude", "Basse Terre", 6),
80    ("Basse Terre", "Saint Claude", 6),
81    ("Saint Claude", "Trois Rivi re", 6),
82    ("Trois Rivi re", "Saint Claude", 6),
83    ("Trois Rivi re", "Vieux Fort", 13),
84    ("Vieux Fort", "Trois Rivi re", 13),
85    ("Basse Terre", "Vieux Fort", 7),
86    ("Vieux Fort", "Basse Terre", 7),
87    ("Gourbeyre", "Vieux Fort", 10),
88    ("Vieux Fort", "Gourbeyre", 10),
89    ("Baillif", "Vieux Fort", 10),
90    ("Vieux Fort", "Baillif", 10)
91
92 ]
93
94
95 # Ajout des routes au graphe
96 G.add_weighted_edges_from(main_axes)
97
98 # Partie pour demander  l'utilisateur de saisir les informations
99 start_node = input("Entrez le sommet de d part : ")
100
101 destinations = []
102 for i in range(6): # Changez ici pour le nombre de sommets d'arriv e que vous souhaitez
103     destination = input(f"Entrez la destination {i+1} : ")
104     destinations.append(destination)
105
106 # V rification que les sommets saisis sont pr sents dans le graphe
107 if start_node not in G.nodes or any(dest not in G.nodes for dest in destinations):
108     print("Certains sommets saisis ne sont pas pr sents dans le graphe.")
109 else:
110     # Ajout du sommet de d part  la liste des destinations pour former un cycle
111     destinations.insert(0, start_node)
112
113     # Utiliser la fonction all_simple_paths pour trouver tous les chemins possibles
114     all_paths = list(nx.all_simple_paths(G, source=start_node, target=destinations[-1]))
115
116     # Utiliser la fonction all_simple_paths pour trouver tous les chemins possibles
117     all_paths = list(nx.all_simple_paths(G, source=start_node, target=destinations[-1]))
118
119     # Calculer le poids total de chaque chemin et trouver le chemin de poids minimum
120     min_weight = float('inf')
121     min_path = None
122
123     for path in all_paths:
124         weight = sum(G[path[i]][path[i + 1]]['weight'] for i in range(len(path) - 1))
125         if weight < min_weight:
126             min_weight = weight
127             min_path = path
128
129     # Dessiner le graphe avec le chemin de poids minimum en rouge
130     pos = nx.spring_layout(G)
131     nx.draw(G, pos, with_labels=True, node_size=700, node_color='skyblue', font_size=8, arrowsize=10, edge_color='gray', font_color='black', width=1)
132
133     edges = list(zip(min_path, min_path[1:]))
134     nx.draw_networkx_edges(G, pos, edgelist=edges, edge_color='red', width=2)
135
136     # Dessiner les sommets en rouge
137     nx.draw_networkx_nodes(G, pos, nodelist=min_path, node_color='red', node_size=700)
138
139     labels = nx.get_edge_attributes(G, 'weight')
140     nx.draw_networkx_edge_labels(G, pos, edge_labels=labels)
141
142     # Afficher le poids total du chemin
143     print(f"Poids total du chemin : {min_weight}")
144
145     plt.show()

```