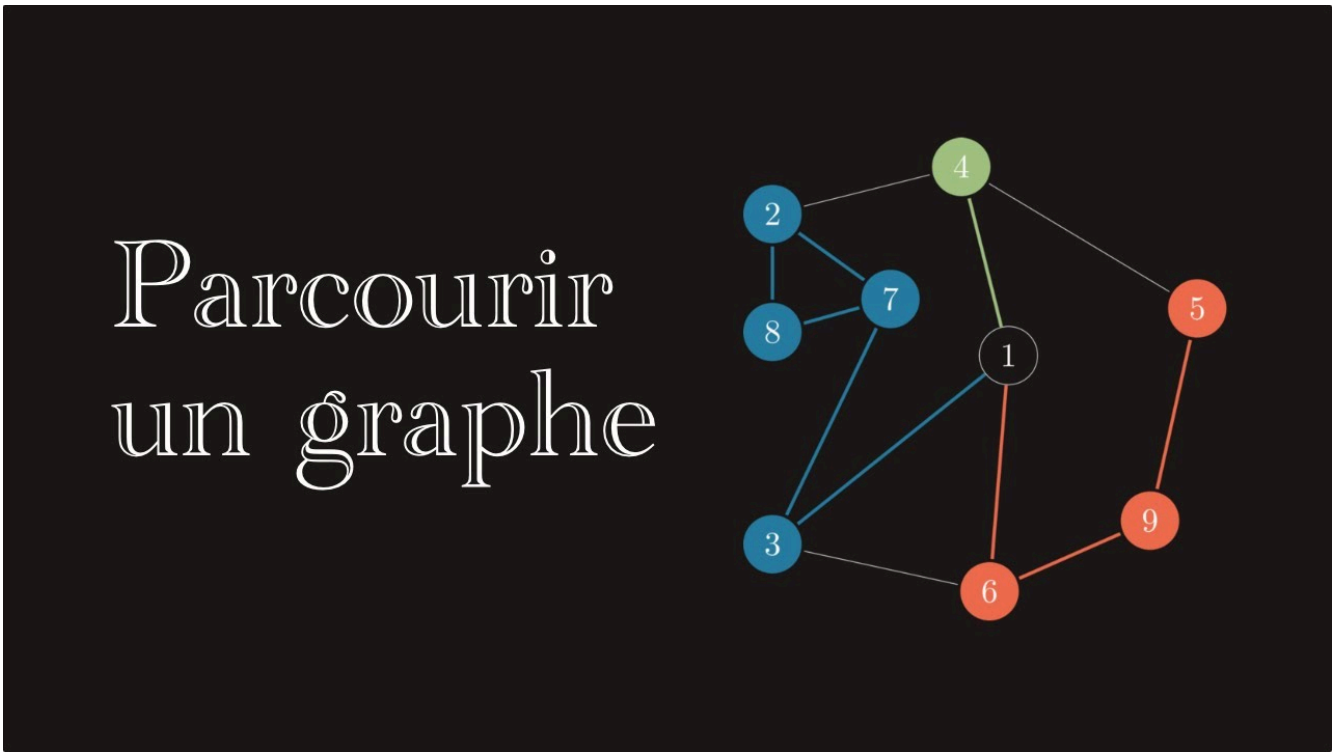
Analyse réflexive



A. Observation et Description

Objectif de la SAE

L'objectif principal de cette SAE est d'appliquer des algorithmes de graphes pour résoudre des problèmes concrets, tout en développant des compétences en programmation, en gestion de projet, et en analyse critique.

Contexte

Ce projet est réalisé dans le cadre du département Réseaux et Télécommunications de l'IUT de La Rochelle. Nous devons fournir plusieurs livrables : un rapport technique, une démonstration orale, ainsi que les fichiers sources. Le tout doit être rendu dans des délais stricts, ce qui exige une bonne organisation et une gestion rigoureuse du temps.

Présentation de la Trace et de la Compétence

Trace choisie : Algorithme utilisé pour coder une réponse à la problématique
Compétence ciblée : Analyser, modéliser, et développer une solution informatique à l'aide d'algorithmes de graphes.

Compétence du Référentiel Travaillée

La compétence concerne la capacité à utiliser des outils et des méthodes avancées en programmation pour modéliser des problèmes complexes, ainsi que la maîtrise de la gestion de projet.

Lien entre la Trace et la Compétence

L'implémentation d'un algorithme de graphe, par exemple pour optimiser des ressources ou minimiser des coûts, met en application directe les compétences de modélisation, de résolution algorithmique et de développement logiciel.

Trace

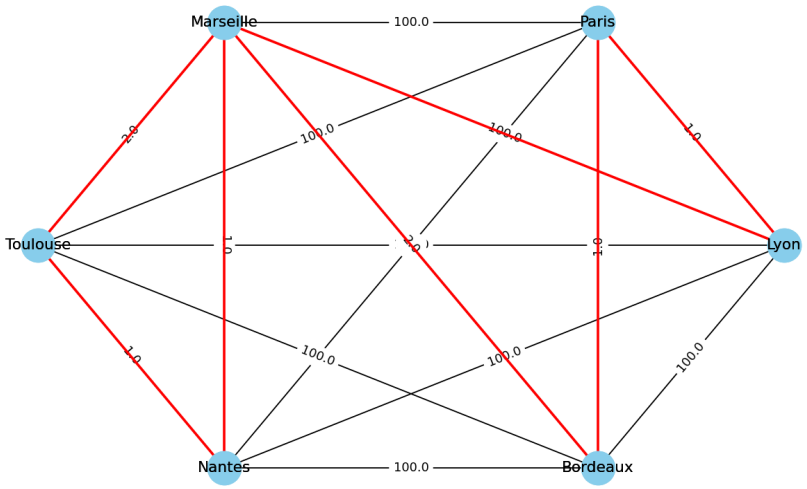
```python
import networkx as nx
import matplotlib.pyplot as plt

def affichage_simple(graph):
    pos = nx.circular_layout(graph)
    plt.figure(figsize=(10, 6))
    # Dessiner le graphe initial sans les noeuds (cercles)
    nx.draw(graph, pos, with_labels=True, node_color='blue', node_size=0, edge_color='black')
    # Personnaliser l'affichage des labels dans des rectangles rouges
    for node, (x, y) in pos.items():
        plt.text(x, y, node, fontsize=12, ha='center', va='center',
                 bbox=dict(facecolor='white', edgecolor='red', boxstyle="round,pad=0.3"))
    nx.draw_networkx_edge_labels(
        graph, pos, edge_labels=nx.get_edge_attributes(graph, 'weight')
    )
    plt.title("Graphe")
    plt.show()

def affichage_arbre(graph, arbre_couvrant):
    pos = nx.circular_layout(graph)
    pos2 = nx.circular_layout(arbre_couvrant)
    plt.figure(figsize=(10, 6))
    # Dessiner le graphe initial sans les noeuds (cercles)
    nx.draw(graph, pos, with_labels=True, node_color='blue', node_size=0, edge_color='black')
    # Personnaliser l'affichage des labels dans des rectangles rouges
    for node, (x, y) in pos.items():
        plt.text(x, y, node, fontsize=12, ha='center', va='center',
                 bbox=dict(facecolor='white', edgecolor='red', boxstyle="round,pad=0.3"))
    # Dessiner l'arbre couvrant (en rouge)
    nx.draw(
        arbre_couvrant, pos2, with_labels=True, node_color='blue', node_size=0,
        edge_color='red', width=2, label="Arbre couvrant"
    )
    # Personnaliser l'affichage des labels dans des rectangles rouges pour l'arbre couvrant
    for node, (x, y) in pos2.items():
        plt.text(x, y, node, fontsize=12, ha='center', va='center',
                 bbox=dict(facecolor='white', edgecolor='red', boxstyle="round,pad=0.3"))

    plt.title("Graphe avec arbre couvrant")
    plt.show()
```

La trace présentée ci-dessus représente le code d'une de nos fonctions qui nous permettent d'afficher le résultat final, comme sur la figure ci-dessous.



B. Analyse et Auto-évaluation

Connaissances et Enseignements Mobilisés

- **Connaissances techniques :** Utilisation de Python, bibliothèques comme *NetworkX* pour manipuler des graphes, et *Matplotlib* pour la visualisation.
- **Connaissances théoriques :** Algorithmes de graphes (Dijkstra, Kruskal, Prim), optimisation.
- **Enseignements pratiques :** Gestion du temps et des tâches en projet, diagramme de Gantt, communication lors des soutenances.

Qu'ai-je appris ?

- L'application des algorithmes de graphes à des scénarios réels.
- L'importance de planifier et de structurer un projet.

Compétences développées :

- Développement d'une méthodologie rigoureuse pour résoudre des problèmes complexes.
- Amélioration des compétences de programmation en Python.

Difficultés Rencontrées et Solutions

Difficultés :

- Compréhension initiale de certains algorithmes complexes.
- Gestion du temps face aux contraintes strictes de livraison.

Solutions :

- Étudier des exemples concrets d'applications des algorithmes.
- Répartir les tâches de manière claire au sein de l'équipe et utiliser des outils de gestion comme Trello.