Projet ALMF51

Algorithmes de graphes – Parcours, optimisation et applications

---

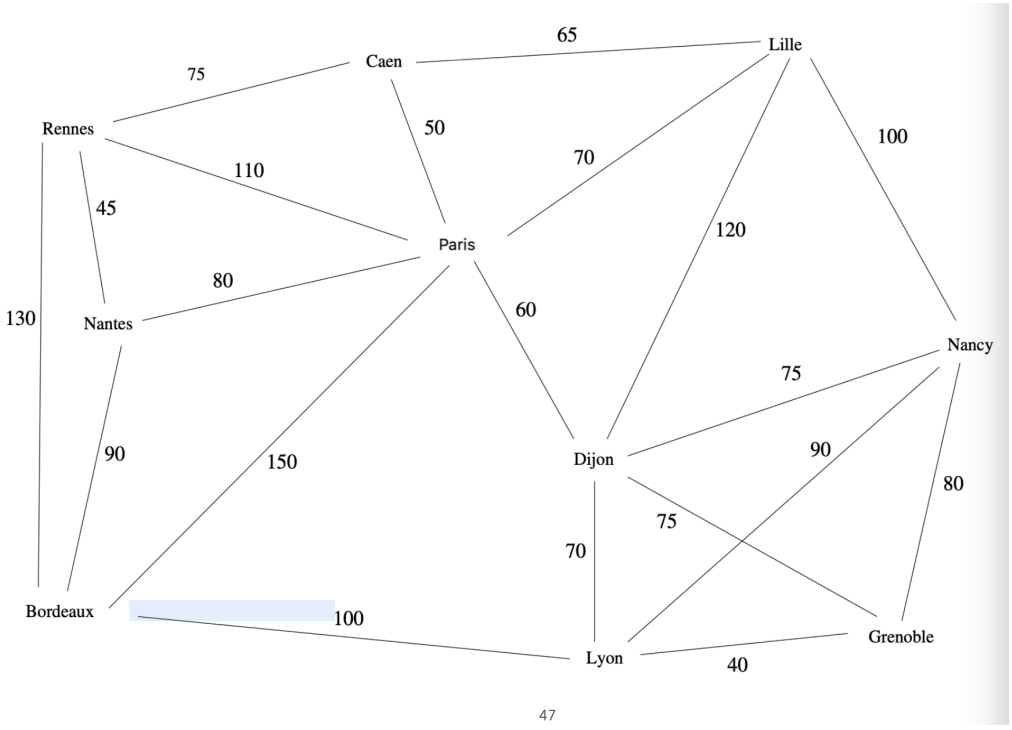
Deadline : 31-10-2025 à minuit

Depôt : via moodle

Contexte

On considère un graphe pondéré représentant un réseau routier entre plusieurs villes françaises. Les sommets représentent les villes et les arêtes les routes reliant ces villes. Le poids associé à chaque arête correspond à la distance (ou au coût) de la route.

La carte suivante sera utilisée comme base du projet :



Objectifs du projet

L'objectif est de développer une application permettant d'expérimenter différents algorithmes de graphes. Chaque partie correspond à un chapitre du cours et sera accompagnée d'une implémentation et d'une expérimentation sur le graphe fourni.

Travail demandé

Pour la partie implémentation, le choix du langage de programmation est libre. Les étudiants peuvent utiliser Python, Java, C++, ou tout autre langage dans lequel ils se sentent à l'aise. L'important est que le code soit clair, fonctionnel et correctement documenté.

I – Algorithmes de parcours

- Implémenter les algorithmes de parcours en largeur (BFS) et en profondeur (DFS).
- Exemple : donner l'ordre de visite des villes atteignables à partir de Rennes.

II – Arbre couvrant de poids minimum

- Implémenter les algorithmes de Kruskal et Prim.
- Construire le réseau routier minimal reliant toutes les villes.
- Afficher l'ensemble des routes choisies et le coût total.

III – Recherche du chemin optimal (Dijkstra)

- Implémenter l'algorithme de Dijkstra.
- Trouver le plus court chemin entre deux villes données par l'utilisateur (exemple : Bordeaux → Lille).

IV – Recherche du chemin optimal (Bellman-Ford, Floyd-Warshall)

- Implémenter Bellman-Ford et tester avec un graphe modifié incluant des poids négatifs.
- Implémenter Floyd-Warshall et afficher la matrice des plus courts chemins entre toutes les paires de villes.

Bonus – Interface graphique

En bonus, réaliser une interface permettant :

- L'affichage du graphe des villes (nœuds + routes pondérées).
- Le choix d'un algorithme (BFS, DFS, Dijkstra, Kruskal, Floyd-Warshall).
- L'affichage du résultat directement sur le graphe (chemin coloré, arbre couvrant,...).

Technologies possibles :

- **Desktop** : Python avec Tkinter ou PyQt5 (affichage avec NetworkX + Matplotlib).
- **Web** : Flask/Django (backend) et D3.js/Vis.js (visualisation des graphes).

Livrables

1. Le code source complet, bien documenté.
2. Un rapport présentant :
   — la théorie de chaque algorithme,
   — l'implémentation,
   — un exemple de résultat obtenu sur le graphe donné.
3. (Bonus) Une interface graphique permettant de visualiser et d'interagir avec les algorithmes.

Notation et évaluation

L'évaluation du projet se fera sur les critères suivants :

- **Qualité de l'implémentation** (40%) :
  — Fonctionnalité complète des algorithmes implémentés (BFS, DFS, Dijkstra, Kruskal, Floyd-Warshall).
  — Respect des bonnes pratiques de programmation et documentation du code.
  — Clarté et lisibilité du code.
- **Rapport écrit** (30%) :
  — Explication théorique des algorithmes.
  — Description de l'implémentation et choix techniques.
  — Résultats expérimentaux et interprétation.
- **Résultats et démonstrations** (20%) :
  — Justesse des résultats obtenus sur le graphe fourni.
  — Illustrations claires (graphiques, tableaux, captures d'écran).
- **Bonus – Interface graphique** (10%) :
  — Réalisation d'une interface interactive.
  — Qualité de l'affichage et ergonomie.
  — Possibilité d'expérimenter facilement avec différents algorithmes.

Remarque : la note finale sera calculée sur 20 points et pourra inclure des points bonus pour les fonctionnalités additionnelles ou la créativité dans la réalisation du projet.