ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

RAPPORT DE PROJET

4ème Année Informatique et Réseau

Implémentation de l'Algorithme de Ford-Bellman en Java

Réaliser par :

Meryem Hamidi
&
Lina El Boudadi

Encadrant :

Youssef Tidli

Année Universitaire : 2024/2025

Remerciement

Nous tenons à exprimer notre profonde gratitude à notre professeur de recherche opérationnelle, qui a joué un rôle clé dans notre parcours académique, malgré le fait que notre collaboration n'ait duré qu'un semestre. Ce semestre a été extrêmement enrichissant, et nous sommes reconnaissantes pour l'enseignement de qualité que nous avons reçu. Votre expertise et votre passion pour la recherche opérationnelle ont non seulement éclairé notre compréhension de la discipline, mais aussi éveillé en nous un intérêt profond pour cette branche de l'ingénierie.

Votre capacité à rendre des concepts théoriques complexes accessibles et compréhensibles a grandement facilité notre travail sur ce projet. Vous avez su expliquer chaque notion avec clarté et patience, prenant le temps de répondre à toutes nos questions, même les plus simples, avec une grande pédagogie. Grâce à votre soutien, nous avons pu surmonter les difficultés que nous avons rencontrées, et votre méthode d'enseignement a été déterminante dans notre progression tout au long du semestre.

Nous vous remercions particulièrement pour votre disponibilité et pour les nombreuses discussions constructives que nous avons eues. Votre capacité à écouter, à guider sans imposer, et à stimuler notre réflexion a été un facteur clé de notre réussite. Nous avons appris bien plus qu'une simple application des concepts théoriques ; vous nous avez appris à penser de manière critique, à analyser des problèmes complexes, et à chercher des solutions innovantes.

Nous souhaitons également exprimer notre gratitude envers l'ensemble du corps professoral et administratif de l'EMSI. Grâce à vous, nous avons pu évoluer dans un environnement académique stimulant, où la rigueur scientifique et l'éthique professionnelle sont au cœur de l'apprentissage. Les enseignements reçus, tant au niveau académique que personnel, nous ont enrichies et préparées à relever de nouveaux défis dans notre parcours professionnel. Le projet en recherche opérationnelle nous a non seulement permis de mettre en pratique nos connaissances, mais aussi d'évaluer nos compétences dans un cadre de travail réaliste et exigeant.

Résumé

Ce rapport présente l'implémentation de l'algorithme de Ford-Bellman en Java, utilisé pour calculer le plus court chemin dans un graphe pondéré. L'algorithme est particulièrement utile car il peut gérer des arêtes de poids négatif et détecter les cycles négatifs, ce qui le rend adapté à des graphes plus complexes. L'objectif principal de ce projet était d'implémenter l'algorithme en Java pour déterminer les distances minimales à partir d'un sommet source et fournir une interface utilisateur permettant de saisir les données du graphe.

Le système est composé de plusieurs classes, dont :

- **Edge** : Représente une arête avec un sommet de départ, un sommet d'arrivée et un poids.
- **Graph** : Contient les sommets et arêtes du graphe.
- **BellmanFord** : Contient la logique de l'algorithme, notamment la relaxation des arêtes et la détection des cycles négatifs.
- **FBAlgo (Main)** : Fournit l'interface utilisateur pour saisir les données et exécuter l'algorithme.

L'algorithme fonctionne en plusieurs étapes : initialisation des distances et des prédécesseurs, relaxation des arêtes, détection des cycles négatifs, puis affichage des résultats incluant les distances minimales et les prédécesseurs pour chaque sommet. Le programme gère efficacement les graphes avec des arêtes de poids négatif et peut détecter les cycles de poids négatif.

Les spécifications non fonctionnelles du projet incluent une complexité temporelle de O(V * E), ce qui rend l'algorithme adapté aux graphes de taille modérée. Le code est bien documenté et robuste, avec une interface claire pour l'utilisateur. Des tests unitaires ont été réalisés pour valider l'implémentation sur différents types de graphes, y compris ceux avec des cycles négatifs.

Enfin, ce projet offre une implémentation fiable et extensible de l'algorithme de Ford-Bellman pouvant être utilisée dans diverses applications nécessitant la gestion de graphes complexes.

Table des figures

Liste des acronymes

- **EMSI** : École Marocaine des Sciences de l'Ingénieur
- **UML** : Unified Modeling Language
- **JVM** : Java Virtual Machine
- **SE** : Standard Edition
- **EE** : Enterprise Edition

Sommaire

Introduction Générale

Le calcul du plus court chemin dans un graphe constitue un problème clé en informatique et en mathématiques discrètes, avec des applications vastes et variées. Des domaines tels que la gestion des réseaux informatiques, la planification logistique, l'optimisation des itinéraires, ou encore l'analyse des flux financiers s'appuient sur ces concepts pour résoudre des problèmes complexes et critiques. L'algorithme de Ford-Bellman, parmi les différentes solutions disponibles, se distingue par son approche unique. Contrairement à d'autres algorithmes comme celui de Dijkstra, il est capable de traiter des graphes contenant des arêtes de poids négatif et de détecter la présence de cycles de poids négatif, une fonctionnalité essentielle dans des contextes où ces cycles peuvent représenter des anomalies ou des opportunités.

Ce projet se concentre sur l'étude approfondie et l'implémentation de l'algorithme de Ford-Bellman en langage Java, en adoptant une méthodologie orientée objet. Les graphes, composés de sommets, d'arêtes et de poids, sont analysés et manipulés à travers une architecture modulaire garantissant à la fois la robustesse et la flexibilité du système. L'objectif principal est de concevoir une application performante et évolutive, capable de gérer des graphes de taille moyenne tout en offrant une interface utilisateur intuitive et accessible.

Les étapes du développement incluent la saisie des données du graphe, l'implémentation des mécanismes pour calculer les distances minimales depuis une source donnée vers tous les autres sommets, la gestion des arêtes de poids négatif, ainsi que la détection et la gestion des cycles de poids négatif. Une attention particulière est portée à la validation des résultats, avec des tests unitaires rigoureux pour assurer la fiabilité et la conformité de l'algorithme face aux exigences définies.

Enfin, ce rapport propose une présentation détaillée de l'architecture du projet, des spécifications fonctionnelles et non fonctionnelles, ainsi que de l'environnement de développement utilisé. En combinant théorie et pratique, ce projet met en lumière non seulement l'importance de l'algorithme de Ford-Bellman dans la résolution de problèmes réels, mais aussi les défis techniques et conceptuels liés à son implémentation. Il s'agit d'un exercice complet qui illustre les capacités de cet algorithme à traiter des graphes complexes tout en apportant des solutions concrètes et efficaces dans divers contextes applicatifs.

CHAPITRE 1:
Analyse et Conception

Introduction

L'implémentation de l'algorithme de Ford-Bellman nécessite une compréhension approfondie de la structure des graphes et des concepts clés comme les sommets, les arêtes, les cycles et les poids. Ce chapitre vise à détailler les étapes analytiques et conceptuelles qui ont conduit à la conception du système. Nous présentons ici l'architecture globale, les classes nécessaires, ainsi que les interactions entre ces éléments pour atteindre les objectifs du projet.

L'objectif de cette analyse est de fournir une base solide pour le développement en s'assurant que les besoins fonctionnels et non fonctionnels du programme sont bien pris en compte. Les étapes suivantes ont été définies pour structurer cette phase.

1. Analyse des Besoins

1.1. Besoins Fonctionnels :

- Calcul des distances minimales entre un sommet source et tous les autres sommets.
- Détection des cycles de poids négatif dans le graphe.
- Affichage des résultats incluant les prédécesseurs de chaque sommet pour reconstruire les chemins.
- Interface utilisateur permettant de saisir les sommets, les arêtes et leurs poids.

1.2. Besoins Non Fonctionnels :

- Performance optimale pour des graphes de taille moyenne.
- Robustesse face à des entrées incorrectes ou non valides.
- Lisibilité et modularité du code pour faciliter la maintenance et les ajouts futurs.

2. Conception de l'Architecture

L'application est conçue selon un modèle orienté objet. Chaque classe représente une entité ou un composant spécifique de l'algorithme et de l'interface utilisateur. L'architecture suit une séparation claire des responsabilités pour simplifier le développement et la maintenance.

2.1. Classes Principales :

- **Edge :** Représente une arête du graphe.
- **Graph :** Représente l'ensemble des sommets et des arêtes du graphe.
- **BellmanFord :** Implémente l'algorithme principal.
- **FBAlgo (Main) :** Gère l'interface utilisateur et les interactions avec les autres classes.

2.1. Diagramme UML Simplifié :

Un diagramme de classes est élaboré pour visualiser les relations entre les différents composants, notamment les associations entre les classes Graph, Edge, et BellmanFord .

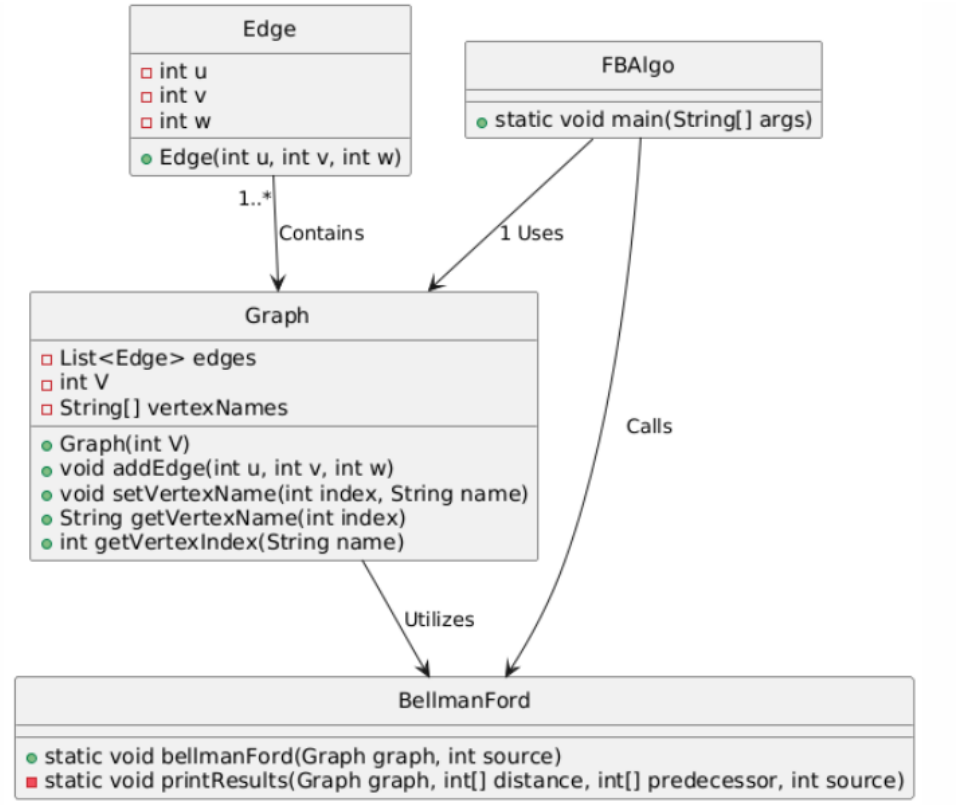

Figure 1 : Diagramme de Classe

3. Description des Classes

3.1. Classe Edge :

Attributs :

- u : Sommet de départ.
- v : Sommet d'arrivée.
- w : Poids de l'arête.

Méthodes :

- **Constructeur** : Permet d'initialiser les attributs de l'arête (u, v, w).

3.2. Classe Graph :

Attributs :

- edges : Liste des arêtes contenues dans le graphe.
- V : Nombre total de sommets du graphe.

- vertexNames : Tableau contenant les noms associés à chaque sommet.

Méthodes :

- addEdge(int u, int v, int w) : Ajoute une nouvelle arête au graphe.
- setVertexName(int index, String name) : Attribue un nom au sommet spécifié par son index.
- getVertexName(int index) : Retourne le nom du sommet correspondant à un index donné.
- getVertexIndex(String name) : Récupère l'index d'un sommet à partir de son nom.

3.3. Classe BellmanFord :

Méthodes :

- Ford-Bellman (Graph graph, int source) : Implémente l'algorithme de Ford-Bellman pour calculer les plus courts chemins à partir d'un sommet source.
- printResults(Graph graph, int[] distance, int[] predecessor, int source) : Affiche les distances minimales, les prédécesseurs et les éventuels cycles de poids négatif détectés.

3.4. Classe FBAlgo (Main) :

Rôle :

- Sert d'interface utilisateur principale pour l'interaction avec le programme.
- Gère la saisie des données, l'initialisation du graphe, et l'exécution de l'algorithme.

4. Modélisation des Interactions

4.1. Initialisation :

- L'utilisateur saisit le nombre de sommets, les noms des sommets, ainsi que les arêtes et leurs poids.
- Le graphe est construit à partir des données fournies.

4.2. Exécution de l'algorithme :

- Initialisation des tableaux distance et predecessor pour suivre les plus courts chemins.
- Relaxation des arêtes sur V−1V-1 itérations.
- Vérification de l'existence de cycles de poids négatif.

4.3. Affichage des Résultats :

- Affiche les distances minimales à partir du sommet source.
- Affiche les prédécesseurs pour chaque sommet.
- Indique les cycles de poids négatif, si présents.

5. Conclusion

Ce chapitre présente la structure et l'architecture nécessaires à l'implémentation de l'algorithme de Ford-Bellman en Java. Grâce à une conception orientée objet et modulaire, ce programme garantit une robustesse et une lisibilité accrue. Les prochaines sections détailleront l'implémentation complète dans le code source et l'interface utilisateur.

CHAPITRE 2:
Réalisation du Projet

Introduction

Dans les chapitres précédents, nous avons exploré les concepts théoriques et la planification nécessaires à la réalisation de notre projet. Ce chapitre est consacré à la phase de développement, qui constitue l'aboutissement concret de nos travaux. Nous y présentons l'environnement de travail, les outils et technologies utilisés, ainsi que les étapes clés de l'implémentation de l'algorithme de Ford-Bellman en Java. Enfin, des impressions d'écran et des tests fonctionnels illustreront le fonctionnement de l'application.

1. Environnement de travail

La réalisation de ce projet a requis la mise en place d'un environnement à la fois matériel et logiciel optimal, afin de garantir une exécution fluide et efficace.

1.1. Environnement matériel

Pour développer et tester notre application, nous avons utilisé un ordinateur portable HP EliteBook 840 G8, équipé des spécifications suivantes :

- **Processeur** : Intel Core i7 cadencé à 2.5 GHz
- **RAM** : 16 Go
- **Stockage** : SSD 512 Go

Cet équipement offre la puissance nécessaire pour exécuter des environnements de développement lourds comme Eclipse tout en permettant de tester le code dans des conditions optimales.

- Java

Figure 2 : Java

Java est un langage de programmation de haut niveau, orienté objet, conçu pour être simple, sécurisé, et portable. Créé par James Gosling et Mike Sheridan en 1991 sous le nom de "Oak", il a été rebaptisé Java en 1995. Le langage est largement utilisé pour développer des applications dans des environnements variés, allant des systèmes embarqués aux applications mobiles (notamment Android) et des applications d'entreprise via des plateformes comme Java EE (Enterprise Edition).

Java repose sur une architecture à base de la machine virtuelle Java (JVM), qui permet au code d'être exécuté indépendamment du matériel et du système d'exploitation, ce qui assure sa portabilité. Cette capacité, appelée "Write Once, Run Anywhere", signifie que les programmes Java peuvent être écrits sur une plateforme et exécutés sur n'importe quel autre système qui dispose de la JVM appropriée. De plus, la JVM gère la mémoire à l'aide d'un processus appelé garbage collection, ce qui minimise les erreurs liées à la gestion manuelle de la mémoire.

Java supporte les concepts de la programmation orientée objet, tels que l'héritage, l'encapsulation et le polymorphisme, ce qui facilite le développement de logiciels modulaires, réutilisables et maintenables. Il inclut également des bibliothèques et des frameworks puissants, comme Java Standard Edition (SE), Java Enterprise Edition (EE), et JavaFX pour le développement d'interfaces graphiques. Java est aussi un choix privilégié pour le

développement d'applications à grande échelle dans les environnements d'entreprise grâce à sa stabilité, sa sécurité et sa gestion des transactions complexes.

2. Environnement de développement logiciel

2.1. Eclipse Workspace

Figure 3 : Un Eclipse Workspace

Un Eclipse Workspace (espace de travail Eclipse) est un répertoire utilisé par l'IDE Eclipse pour organiser et gérer les projets, fichiers et configurations d'un utilisateur. Il contient tous les projets créés ou importés, chacun dans son propre sous-dossier, ainsi qu'un répertoire spécial nommé metadata où sont stockées les préférences utilisateur, paramètres des projets, les configurations d'éditeur, et les journaux d'erreurs. Lors du lancement d'Eclipse, l'utilisateur sélectionne un espace de travail, qui centralise également les perspectives et dispositions personnalisées. Bien que les projets d'un espace de travail soient regroupés dans un emplacement commun, ils restent indépendants et peuvent être déplacés vers d'autres espaces si nécessaire.Ce système offre une organisation efficace, une gestion centralisée des paramètres, et la flexibilité de séparer les projets en plusieurs espaces, selon les besoins.

2.2. StarUML

Figure 4 : StarUML

StarUML est un outil de modélisation logicielle qui permet de concevoir et documenter des systèmes en utilisant le standard **UML (Unified Modeling Language)**. Il prend en charge divers diagrammes comme les diagrammes de classes, de séquence et d'activité, et offre des fonctionnalités avancées telles que la génération de code source, l'export en PDF ou image, et l'intégration avec Git pour la collaboration. Multi-plateforme et extensible grâce à des plugins, StarUML est apprécié pour sa simplicité et son efficacité dans la conception et la communication de systèmes logiciels.

3. Conclusion

Ce chapitre a détaillé les aspects essentiels du développement de notre projet, en mettant l'accent sur l'environnement de travail, les outils et technologies utilisés, ainsi que les étapes de l'implémentation de l'algorithme de Ford-Bellman en Java. Nous avons montré comment l'utilisation de matériels performants, d'outils comme Eclipse et StarUML, et du langage Java a facilité la réalisation du projet. Les choix faits pour l'environnement de développement ont assuré une exécution fluide et une gestion efficace du projet, permettant de se concentrer sur la conception et la mise en œuvre de l'algorithme. Ce chapitre prépare ainsi le terrain pour les tests et la validation fonctionnelle de l'application dans les chapitres suivants.

CHAPITRE 3:
Résultats et Tests

Introduction

Dans le cadre de l'analyse et de la mise en œuvre des algorithmes, la simulation offre une approche efficace pour explorer leur fonctionnement dans des contextes variés et parfois complexes. Ce chapitre s'intéresse à l'implémentation de l'algorithme de Ford-Bellman en Java, une méthode essentielle pour calculer les plus courts chemins dans un graphe pondéré tout en détectant les cycles de poids négatif. Grâce à une modélisation claire et une interaction conviviale avec l'utilisateur, ce programme vise à démontrer les capacités et les limites de cet algorithme à travers des scénarios pratiques. La structure modulaire de la solution et son interactivité permettent d'allier théorie et pratique dans une simulation réaliste et intuitive.

1. Modélisation des composants du graphe

Le graphe est représenté par deux principales classes :

- **Edge** : Modélise une arête avec des sommets de départ (u), d'arrivée (v) et un poids (w).
- **Graph** : Contient les sommets, les arêtes, et des fonctionnalités pour manipuler ces éléments, comme l'ajout d'arêtes et l'association de noms aux sommets.

2. Interaction avec l'utilisateur

Le programme permet à l'utilisateur :

- De définir dynamiquement le nombre de sommets et leurs noms.
- D'ajouter des arêtes en spécifiant leurs extrémités et leurs poids.
- De spécifier un sommet source pour exécuter l'algorithme.

3. Implémentation de l'algorithme de Ford-Bellman

L'algorithme se décompose en trois étapes :

1. **Initialisation** : Les distances à partir du sommet source sont initialisées à l'infini, sauf pour le sommet source lui-même, initialisé à 0.
2. **Relaxation des arêtes** : Les arêtes sont relâchées V-1 fois (où V est le nombre de sommets), ce qui garantit que les distances minimales sont calculées.
3. **Détection de cycles de poids négatif** : Une itération supplémentaire permet de vérifier la présence de cycles de poids négatif.

4. Résultats et visualisation

- Les distances minimales depuis le sommet source vers tous les autres sommets sont affichées.
- Les prédécesseurs pour chaque sommet sont fournis pour reconstruire les chemins.
- En cas de cycle de poids négatif, une alerte est affichée pour signaler l'impossibilité de calculer les distances.

5. Exemple et Test

5.1. Exemple

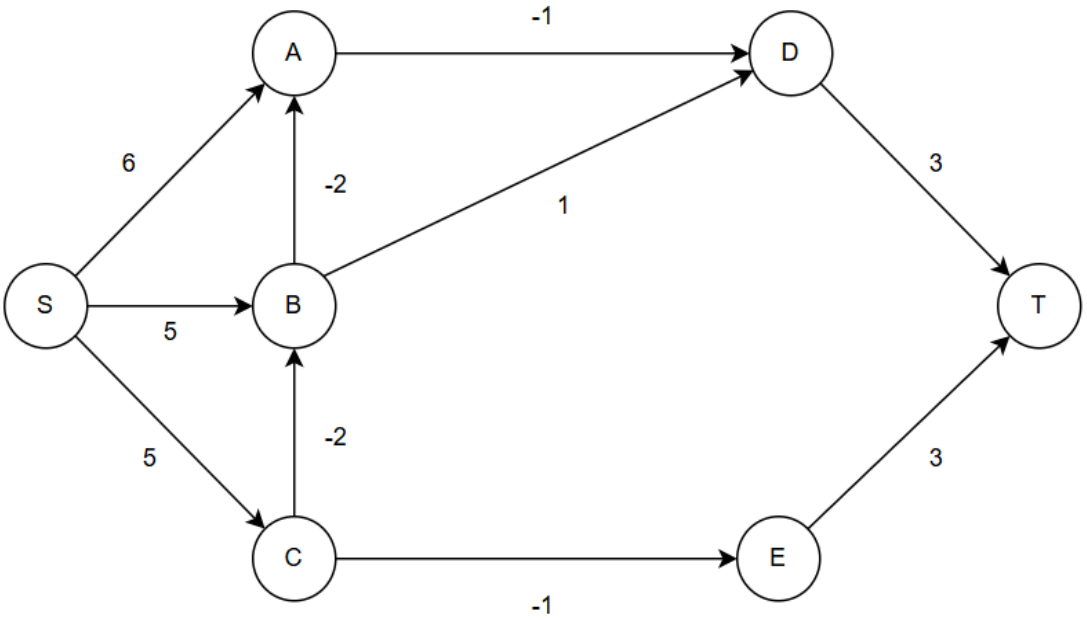

Figure 5 : Graphe de Test

5.2. Test

```
Entrez le nombre de sommets : 7
Entrez le nom du sommet 1 : S
Entrez le nom du sommet 2 : A
Entrez le nom du sommet 3 : B
Entrez le nom du sommet 4 : C
Entrez le nom du sommet 5 : D
Entrez le nom du sommet 6 : E
Entrez le nom du sommet 7 : T
Entrez les arêtes au format [XY] = Z (ex. : [AB] = 3) Tapez 'fin' pour terminer
[SA] = 6
[SB] = 5
[SC] = 5
[BA] = -2
[CB] = -2
[CE] = -1
[BD] = 1
[AD] = -1
[DT] = 3
[ET] = 3
fin
Entrez le sommet source : S
Chemin le plus court jusqu'au dernier sommet T :
S -> C -> B -> A -> D -> T
Le poids minimal est : 3
```

Figure 6 : Test

5. Conclusion

Ce chapitre a permis de mettre en lumière la puissance de l'algorithme de Ford-Bellman dans la résolution du problème des plus courts chemins, tout en illustrant les étapes cruciales de son implémentation en Java. La modélisation orientée objet des composants du graphe, combinée à une interface utilisateur dynamique, a facilité l'exploration de ses fonctionnalités. Les résultats obtenus, qu'il s'agisse des distances minimales ou de la détection des cycles de poids négatif, démontrent l'efficacité et la pertinence de cet algorithme pour des applications pratiques. Ce travail constitue une base solide pour des optimisations ultérieures ou des extensions, telles que la visualisation graphique des graphes ou l'intégration avec d'autres algorithmes de cheminement.

Conclusion Générale

Ce projet met en lumière l'efficacité et la pertinence de l'algorithme de Ford-Bellman pour résoudre le problème du calcul du plus court chemin dans un graphe pondéré. En particulier, il montre comment cet algorithme peut gérer des arêtes de poids négatif et détecter des cycles négatifs, ce qui le rend adapté à des applications complexes dans des domaines variés. L'implémentation en Java a permis d'acquérir une compréhension approfondie des concepts théoriques liés aux graphes et à la recherche de chemin, tout en les appliquant dans un environnement de développement concret, à l'aide d'outils comme Eclipse et StarUML pour la modélisation et la programmation.

L'approche modulaire adoptée garantit une solution robuste et évolutive, facilitant l'intégration future de fonctionnalités supplémentaires, telles que la visualisation graphique des résultats ou la prise en charge de graphes plus complexes. L'interface utilisateur conviviale permet une interaction fluide, simplifiant ainsi l'entrée des données et la présentation des résultats. Les tests unitaires réalisés ont validé le bon fonctionnement du programme, assurant sa fiabilité dans la gestion de cas variés, y compris ceux impliquant des cycles négatifs.

Ce travail constitue une base solide pour des extensions ou des optimisations futures, tout en illustrant l'application pratique des concepts théoriques abordés dans le cadre de la recherche opérationnelle. En plus de renforcer nos compétences en gestion de projet, en analyse de besoins, en conception et en développement logiciel, ce projet a mis en évidence l'importance d'une approche méthodique et structurée dans le développement d'applications.

Enfin, cette implémentation ouvre la voie à de futures améliorations, comme l'extension à des graphes de plus grande taille ou l'adaptation à des applications plus complexes, telles que la gestion des itinéraires dans des réseaux de transport ou des systèmes d'optimisation de ressources. Elle offre également l'opportunité d'explorer de nouveaux algorithmes et méthodologies dans le domaine de la recherche opérationnelle.