UCLouvain - EPL

LINFO1114

Mathématiques discrètes

Rapport du projet

Professeur :
Saerens Marco

Assistants :
Courtain Sylvain
Leleux Pierre

Auteurs :
Ben Haddou Mehdi - 19912000
Hennebo Eliot - 43762000
Mottet Romain - 34391900

Table des matières

Introduction

Dans le cadre du projet du cours LINFO1114, Mathématiques Discrètes, il nous est demandé d'étudier des algorithmes de calcul de plus court chemin dans un graphe. Il nous a été fourni un graphe par groupe et il nous faut calculer les plus courts chemins entre toutes les paires de nœuds du graphe dans une matrice de distances avec les algorithmes de Dijkstra, Bellman Ford et Floyd Warshall.

Nous devons également étudier leurs spécificités et ainsi essayer de faire la comparaison entre les différents algorithmes en découvrant les avantages et inconvénients de chacun de ces algorithmes. Pour ce faire, nous allons commencer par décrire les algorithmes, les implémenter en Python et enfin calculer numériquement le plus court chemin entre 2 nœuds avec l'algorithme de Dijkstra.

1 Rappel théoriques sur les 3 algorithmes

1.1 Algorithme Dijkstra

L'algorithme de Dijkstra, de Edsger Dijkstra, est certainement le plus utilisé des trois algorithmes, il est utilisé pour calculer le plus court chemin entre un sommet source et tous les autres sommets du graphe. Il peut être utilisé pour les graphes dirigés ou non étant soit pondéré positivement soit non pondéré [1].

1.1.1 Intuition

L'intuition de l'algorithme est de construire au fur et à mesure un sous-graphe sur base d'une liste de priorité croissante des distances entre la source et les différents sommets. A chaque itération on va récupérer le sommet le plus proche et ajouter tous les nouveaux sommets atteignables depuis ce sommet. Ensuite, on répète le procédé jusqu'à ce qu'on ait parcouru tous les sommets tout en évitant de repasser deux fois par le même sommet.

En effet, étant donné que le graphe est pondéré positivement on a la garantie qu'à chaque étape, le chemin vers un sommet qui a déjà été accepté est le plus court chemin depuis la source.

Plus formellement, au moment de la i ème itération, l'algorithme a déjà identifié le chemin le plus court vers le i-1 ème sommet le plus proche. Le chemin emprunté peut-être écrit sous la forme d'un sous-arbre qui sera modifié à chaque itération. Les sommets adjacents du i-1 ème sommet sont envoyés au "fringe", ils sont les candidats pour être le i ème sommet le plus proche. Pour identifier ce dernier, l'algorithme calcule pour chaque sommet dans le "fringe" la somme de la distance au sommet de l'arbre le plus proche (donné par le poids de l'arc) et la longueur du chemin le plus court depuis l'origine. Ensuite il sélectionne le sommet avec la plus petite somme.

Enfin on retire le sommet du "fringe" et on ajoute ce sommet à l'arbre. Une fois que cela est fait, on actualise les coûts des sommets les plus proches et on relance l'algorithme à partir du point trouvé.

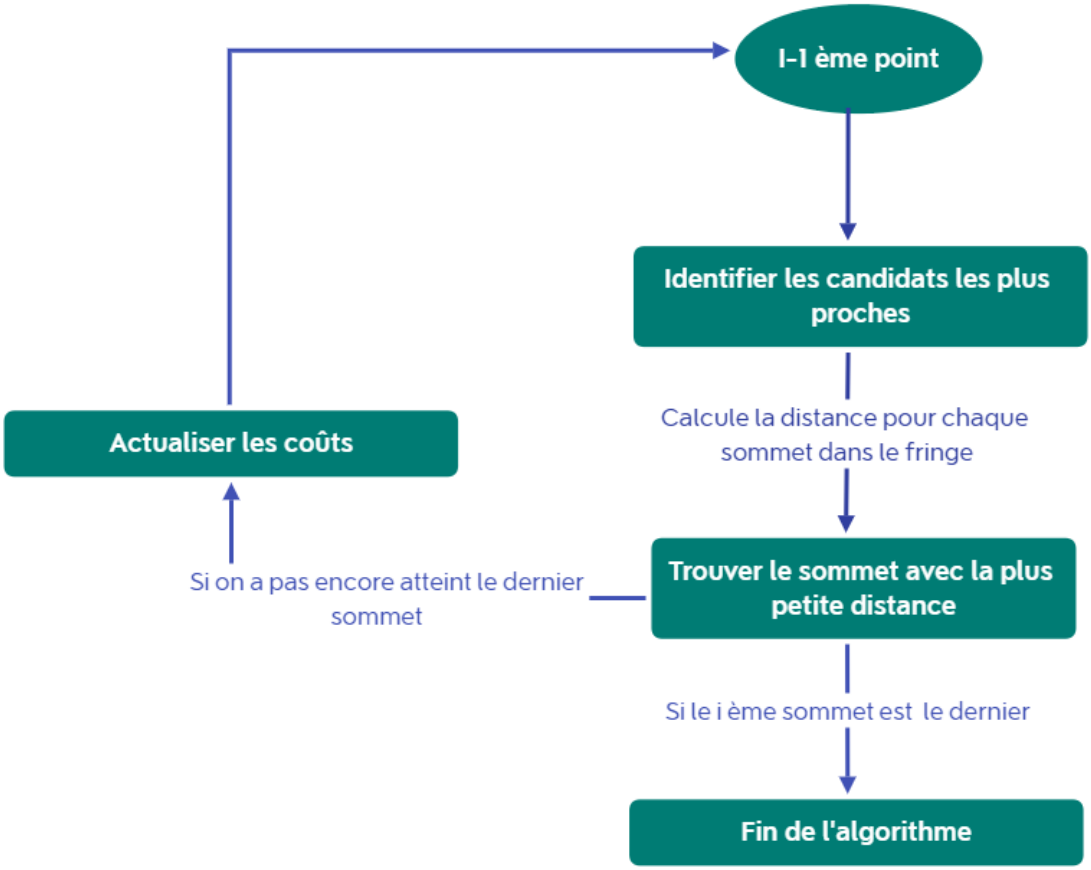
Pour obtenir les plus courts chemins pour chaque paire de sommets possibles, il suffit de lancer l'algorithme de Dijkstra S fois avec comme sommet source un nouveau sommet à chaque fois. On joint les S tableaux de plus courts chemins que nous renvoie chaque exécution de Dijkstra et nous avons une matrice des plus petites distances pour toutes les paires de sommets possibles.

1.1.2 Complexité

La complexité de cet algorithme peut varier en fonction de son implémentation. Il est possible d'implémenter l'algorithme de dijkstra avec une fil de priorités et un tas binaire. Si A représente le nombre d'arcs et S le nombre de sommets alors la complexité de cet algorithme est égale à $\mathcal{O}((A + S) * log(S))$.

1. Dans ce cas nous pondérons tous les arcs à 1.

Si maintenant on implémente un tas de Fibonacci [2], il est possible de réduire la complexité à $\mathcal{O}(A + S * log(S))$.



1.2 Algorithme Bellman-Ford

L'algorithme de Bellman-Ford, ou parfois appelé Bellman-Ford-Moore [3], permet de calculer les plus courts chemins depuis un sommet source vers les autres sommets dans un graphe orienté ou non et pondéré ou non (si le graphe n'est pas pondéré, on met un poids de 1 à chaque arc)

Contrairement à l'algorithme de Dijkstra, cet algorithme peut être utilisé pour des graphes contenant des arcs à poids négatifs. Cependant, le graphe ne peut pas contenir de cycle absorbant [4] sinon la notion de plus court chemin n'aurait plus de sens. En effet, s'il y a la possibilité de passer par un cycle absorbant dans le chemin d'un nœud A à un nœud B, il suffirait de boucler sur le cycle négatif pour atteindre une distance qui pourrait atteindre $-\infty$.

Heureusement, l'algorithme de Bellman-Ford peut être modifié simplement pour détecter les graphes possédant un cycle négatif. En partant du postulat que n'importe quel chemin entre deux sommets parcourt un maximum de $S - 1$ arcs (avec S équivalent au nombre de sommets), nous pouvons nous arrêter lors du calcul du plus court chemin entre deux sommets lorsqu'un chemin parcours S ou plus arcs. En effet, cela signifierait que l'algorithme passe plusieurs fois par le même cycle pour diminuer la distance totale.

2. Un tas de Fibonacci est une structure de données similaire au tas binomial, mais avec un meilleur temps d'exécution amorti. Les tas de Fibonacci ont été conçus par Michael L. Fredman et Robert E. Tarjan en 1984 et publiés pour la première fois dans un journal scientifique en 1987.
3. Edward F. Moore publia en 1959 une variation de l'algorithme ce qui fait que l'algorithme est parfois appelé Bellman-Ford-Moore
4. Un cycle absorbant est une suite d'arcs consécutifs qui se rejoignent dont la somme totale des poids est négative.

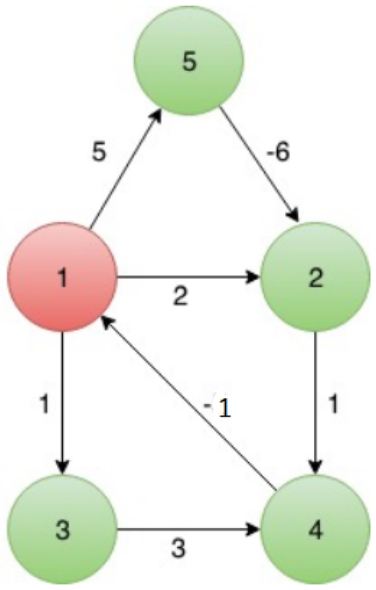
1.2.1 Intuition

L'algorithme en lui-même consiste en une optimisation de l'approche naïve qui parcourt tous les chemins possibles pour calculer les plus courts chemins. Le problème de cette approche est qu'on repasse de nombreuses fois sur des sommets et arcs qui ont déjà été explorés ce qui donne une complexité exponentielle. Pour éviter ces opérations inutiles, l'algorithme de Belman-Ford utilise la programmation dynamique et garde les informations en mémoire afin de les réutiliser intelligemment.

L'intuition de l'algorithme est qu'on parcourt K fois, avec K équivalent à S [5], tous les arcs du graphe en calculant les plus courts chemins entre tous les sommets et le sommet source tout en se basant sur les anciens chemins appris. Pour la première itération, il n'y a pas d'informations précédemment acquise, on aura donc juste un tableau avec les distances des voisins directs de la source. A partir de la deuxième itération nous pouvons utiliser les informations déjà apprises, pour tous les arcs (un arc est relié à deux sommets A_w et A_v) nous testons si la distance entre la source et A_v additionné au poids de l'arc en lui-même est inférieure à la distance entre la source et A_w.

Pour obtenir les plus courts chemins pour chaque paire de sommets possibles, il suffit de lancer l'algorithme Bellman-Ford S fois avec comme sommet source un nouveau sommet à chaque fois. On joint les S tableaux de plus courts chemins que nous renvoie chaque exécution de Bellman-Ford et nous avons une matrice des plus petites distances pour toutes les paires de sommets possibles.

1.2.2 Exemple



Dans cet exemple où la source est le sommet 1, nous aurons à la première itération les distances entre le sommet 1 et les sommets 5,3,2 qui seront équivalents aux poids des arcs qui les relient.

Lors de la deuxième itération, nous pouvons déjà tirer plusieurs nouvelles informations, nous savons déjà que la distance entre 1 et 2 est égale à 2 mais lors du parcours de tous les arcs nous testons si la distance de 1 à 5 additionnée à la distance de 5 à 2 est plus petite que l'ancienne distance entre 1 et 2 gardée en mémoire (lors de l'itération 1).

En l'occurence, la distance est ici plus courte (-1 contre 2) en passant par 5 pour aller à 2, on met donc à jour la plus courte distance vers 2 à $distTo[5] + weigth[5][2]$.

Lors de cette itération nous apprenons également que nous pouvons atteindre 4 en passant par le sommet 2 ou 3, en fonction de l'ordre dans lequel l'algorithme traverse les arcs il va soit mettre à jour la distance directement à 3 (s'il teste l'arc 2-4 en premier) ou d'abord la mettre à 4 (s'il teste l'arc 3-4 en premier) puis à 3 en testant l'arc 2-4 vu que la distance sera inférieure à celle précédemment enregistrée.

1.2.3 Complexité

La complexité de l'algorithme de Bellman-Ford est facile à évaluer. En effet, on effectue S passages parcourant chacun A arcs, on obtient donc une complexité temporelle $\mathcal{O}(S.A)$ où S est le nombre de sommets et A le nombre d'arcs. La complexité spatiale elle est de $\mathcal{O}(S)$ car on ne doit garder en mémoire qu'un tableau contenant les plus petites distances courantes d'un sommet vers tous les autres.

5. K est équivalent à S pour être sûr d'avoir exploré toutes les possibilités de chemin, un chemin ne peut pas utiliser plus de S-1 arcs.

1.3 Algorithme Floyd-Warshall

L'algorithme de Floyd-Warshall, ou parfois appelé algorithme de Roy-Floyd-Warshall [6], permet quant à lui de calculer le plus court chemin entre toutes les paires de sommets d'un graphe orienté ou non et pondéré ou non.

Tout comme l'algorithme de Bellman-Ford, l'algorithme de Floyd-Warshall peut être utilisé sur des graphes contenant des arcs à poids négatifs mais sans cycle absorbant pour les mêmes raisons. L'algorithme de Floyd-Warshall est également un exemple de programmation dynamique, il résout plusieurs sous-problèmes en gardant les informations utiles afin de résoudre un plus gros problème sans devoir faire de nombreuses opérations qui ont déjà été faites.

1.3.1 Intuition

L'intuition de l'algorithme est que pour une paire A-B, le plus court chemin est soit le plus court chemin A-B rencontré à présent ou soit une concaténation de deux plus court chemins A-C et C-B par exemple. L'algorithme va donc tester pour chaque paire de sommets possible si le passage par un des S sommets intermédiaires possibles réduit la distance entre la paire de sommets en question.

L'algorithme en lui-même est très simple et peut tenir en 5 lignes. Il faut tout d'abord initialiser la matrice des plus courts chemins qui sera équivalente à la matrice d'adjacence qui sera donnée en entrée (comme pour l'itération 0 de Bellman-Ford le résultat sera les poids des liens directs entre chaque sommet). Ensuite, on itère de 1 à k ou k représentera le sommet intermédiaire qui sera testé et pour chaque k on itère à travers tous les arcs. Enfin, il nous reste qu'à tester si le plus court chemin déjà connu est inférieur à la somme du plus court chemin de A_w [7] vers k additionné au plus court chemin de k vers A_v. On peut résumer le cœur de l'algorithme par cette simple seule ligne de code :

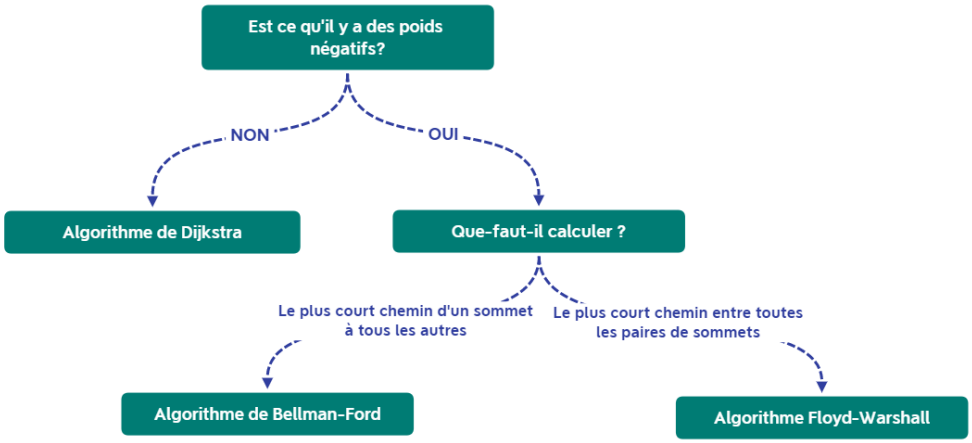
$$shortest\_dists[w][v] = min(shortest\_dists[w][v], shortest\_dists[w][k] + shortest\_dists[v][j])$$

A la fin des K itérations nous obtenons bien une matrice des plus petites distances pour chaque paire de sommets dans le graphe.

1.3.2 Complexité

La complexité de cet algorithme est assez intuitive à évaluer. En effet, pour chaque sommet intermédiaire de 1 à S on va boucler sur tous les arcs possibles donc S^2. On obtient donc $S * S^2$ qui donne une complexité de $\mathcal{O}(S^3)$.

1.4 Choix de l'algorithme



6. Bernard Roy a décrit en 1959 l'algorithme avant que S. Warshall et R. Floyd ne publient leur article en 1962
7. Un arc possède deux sommets $A_w et A_v$

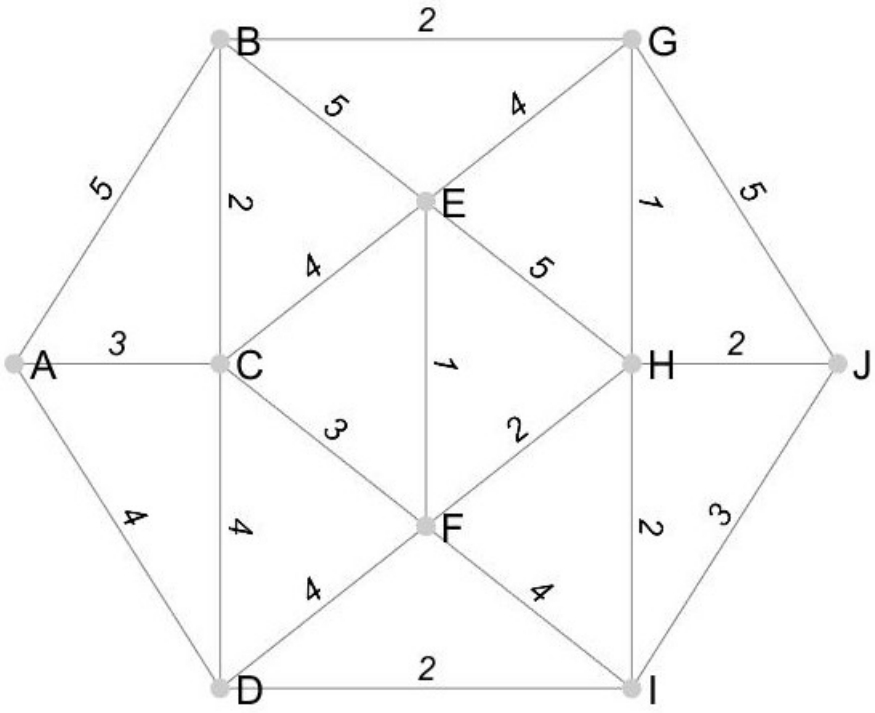
2 Graphe

2.1 Analyse générale

Un graphe différent a été assigné à chaque groupe inscrit au projet, celui-ci contient 10 nœuds représentés par des lettres allant de A à J et 21 arcs pondérés.

Le graphe est :

- ➔ **Non dirigé** : les arcs sont bidirectionnels, cela impliquera donc une matrice d'adjacence symétrique.
- ➔ **Connexe** : il existe un chemin entre toutes les paires de nœuds.
- ➔ **Non complet** : il n'existe pas d'arc entre toutes les paires de nœuds.



3 Calcul théorique et numérique par l'algorithme de Dijkstra

3.1 Analyse

Pour calculer la distance entre le noeud A et le noeud J, il faut exécuter l'algorithme de Dijkstra en prenant le noeud A comme noeud de départ.

On va donc effectuer ce calcul dans la matrice ci-dessous, à chaque itération de l'algorithme, un noeud est visité et mis en couleur, cela signifie qu'il ne sera plus visité par la suite et que la distance calculée pour l'atteindre est la plus courte depuis la source.

Pour obtenir le chemin et pas uniquement la distance entre les nœuds A et J, il nous suffit de noter au fur et à mesure des itérations la suite de nœuds que l'on visite. A savoir qu'à chaque itération, le chemin que l'on note est le plus court chemin entre la source et le noeud marqué durant l'itération. Cela signifie que nous pouvons nous arrêter dès que nous aurons marqué le noeud J, et que nous aurons également calculé la distance et le plus court chemin de A vers tous les nœuds marqués avant J.

3.2 Tableau d'itérations

	L0	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10
a	0	0	-	-	-	-	-	-	-	-	-
b	inf	5	5	5	5	-	-	-	-	-	-
c	inf	3	3	-	-	-	-	-	-	-	-
d	inf	4	4	4	-	-	-	-	-	-	-
e	inf	inf	7	7	7	7	7	7	-	-	-
f	inf	inf	6	6	6	6	-	-	-	-	-
g	inf	inf	inf	inf	7	7	7	7	7	-	-
h	inf	inf	inf	inf	inf	8	8	8	8	8	-
i	inf	inf	inf	6	6	6	6	-	-	-	-
j	inf	inf	inf	inf	inf	inf	9	9	9	9	9

3.3 Plus courts chemins

- L0 : /
- L1 : **A**
- L2 : **A** $\longrightarrow$ **C**
- L3 : **A** $\longrightarrow$ **D**
- L4 : **A** $\longrightarrow$ **B**
- L5 : **A** $\longrightarrow$ C $\longrightarrow$ **F**
- L6 : **A** $\longrightarrow$ D $\longrightarrow$ **I**
- L7 : **A** $\longrightarrow$ C $\longrightarrow$ **E**
- L8 : **A** $\longrightarrow$ B $\longrightarrow$ **G**
- L9 : **A** $\longrightarrow$ B $\longrightarrow$ G $\longrightarrow$ **H**
- L10 : **A** $\longrightarrow$ D $\longrightarrow$ I $\longrightarrow$ **J**

$\Longrightarrow$ Nous arrivons ainsi à la conclusion que le plus court chemin entre les nœuds A et J est obtenu lors de la 10 ème itération de l'algorithme de Dijkstra et que le noeud J est donc le dernier noeud visité.

$\Longrightarrow$ Le plus court chemin est donc **A** $\longrightarrow$ D $\longrightarrow$ I $\longrightarrow$ **J** et la distance totale de ce chemin est de **9**.

4 Résultats

Comme mentionné auparavant, le graph étant un graph non-dirigé, nous avons des matrices de coûts et distances symétriques.

4.1 Matrice de coûts

Pour obtenir cette matrice, il suffit simplement d'assigner une colonne et une ligne à chaque noeud et de récupérer pour toutes les paires de nœuds X Y, le poids P de l'arc qui lie les 2 nœuds et de le placer dans la matrice de coûts M tel que M[X][Y] = P et M[Y][X] = P. Le poids est égal à l'infini "inf" lorsqu'aucun lien n'existe entre 2 nœuds.

	A	B	C	D	E	F	G	H	I	J
A	0	5	3	4	inf	inf	inf	inf	inf	inf
B	5	0	2	inf	5	inf	2	inf	inf	inf
C	3	2	0	4	4	3	inf	inf	inf	inf
D	4	inf	4	0	inf	4	inf	inf	2	inf
E	inf	5	4	inf	0	1	4	5	inf	inf
F	inf	inf	3	4	1	0	inf	2	4	inf
G	inf	2	inf	inf	4	inf	0	1	inf	5
H	inf	inf	inf	inf	5	2	1	0	2	2
I	inf	inf	inf	2	inf	4	inf	2	0	3
J	inf	inf	inf	inf	inf	inf	5	2	3	0

4.2 Matrices de distances

Cette matrice est le résultat obtenu en appliquant les algorithmes de Dijkstra et Bellman Ford sur tous les nœuds, ou en appliquant l'algorithme de Floyd Warshall sur la matrice.

	A	B	C	D	E	F	G	H	I	J
A	0.	5.	3.	4.	7.	6.	7.	8.	6.	9.
B	5.	0.	2.	6.	5.	5.	2.	3.	5.	5.
C	3.	2.	0.	4.	4.	3.	4.	5.	6.	7.
D	4.	6.	4.	0.	5.	4.	5.	4.	2.	5.
E	7.	5.	4.	5.	0.	1.	4.	3.	5.	5.
F	6.	5.	3.	4.	1.	0.	3.	2.	4.	4.
G	7.	2.	4.	5.	4.	3.	0.	1.	3.	3.
H	8.	3.	5.	4.	3.	2.	1.	0.	2.	2.
I	6.	5.	6.	2.	5.	4.	3.	2.	0.	3.
J	9.	5.	7.	5.	5.	4.	3.	2.	3.	0.

5 Conclusion

Le calcul du plus court chemin est une pratique courante dans la vie de tous les jours, il est utilisé par exemples dans les GPS pour des raisons évidentes ou encore sur les réseaux sociaux pour mesurer les distances relatives entre 2 personnes reliées par des amis communs. L'utilité de ces algorithmes n'est donc évidemment plus à démontrer, mais il est enrichissant de s'intéresser à la variété qu'il existe parmi ceux-ci, lorsque l'on examine les 3 algorithmes étudiés dans ce projet, on constate qu'ils ont tous les 3 des avantages et des inconvénients et qu'ils sont pour cette raison tous indispensables.

Références

[1] Cormen, T. H. (2009). Introduction to Algorithms, 3rd Edition (The MIT Press) (3rd ed.). MIT Press.

[2] Lipschutz, S., Lipson, M. (2009). Schaum's Outline of Discrete Mathematics, Revised Third Edition (Schaum's Outlines) (3rd ed.). McGraw-Hill Education.

[3] Sedgewick, R., Wayne, K. (2011). Algorithms (4th Edition) (4th ed.). Addison-Wesley Professional.

[4] Rosen, K. (2018). Discrete Mathematics and Its Applications (Eighth Edition). Mc Graw Hill.

[5] Wikipedia contributors. (2021a, April 14). Graph (discrete mathematics). Wikipedia. https ://en.wikipedia.org/wiki/Graph_(discrete_mathematics)

[6] Wikipedia contributors. (2021, April 27). Floyd–Warshall algorithm. Wikipedia. https ://en.wikipedia.org/wiki/Floyd%E2%80%93Warshall_algorithm

[7] Wikipedia contributors. (2021, May 1). Bellman–Ford algorithm. Wikipedia. https ://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm

[8] Wikipedia contributors. (2021, May 6). Dijkstra's algorithm. Wikipedia. https ://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

[9] Wikipedia contributors. (2021, May 8). Shortest path problem. Wikipedia. https ://en.wikipedia.org/wiki/Shortest_path_problem

[10] Wikipedia contributors. (2021, May 9). Time complexity. Wikipedia. https ://en.wikipedia.org/wiki/Time_complexity

[11] hackerearth (2021, may 11). Shortest Path Algorithms. https ://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/

[12] Aunege. (2021, may 11). RESOLUTION DE PROBLEMES DE PLUS COURT CHEMIN. http ://ressources.aunege.fr/nuxeo/site/esupversions/2b1c56b6-109d-488a-94a3-3ea525f8beef/ModAidDec/cours/l4/l4.pdf