DIJKSTRA ALGORITHM

Encadré par : Pr YASSINE SADQI

Réalisé par : HIND HAKKAOUI

FATIMA ZAHRA HAOUATE

LES CHAPITRES

- INTRODUCTION
- FONDEMENTS THEORIQUES
- STRUCTURE DE DONNEES
- CONCLUSION
- BIBLIOGRAPHIE

L'algorithme de **Dijkstra** est un algorithme de recherche de plus court chemin dans un graphe pondéré.

Il a été proposé par l'informaticien néerlandais Edsger W.Dijkstra en 1956.

L'algorithme est utilisé dans de nombreux domaines, tels que la planification de trajets routiers, la conception de réseaux de télécommunications et la résolution de problèmes d'optimisation.

L'efficacité de l'algorithme dépend fortement de la structure de données utilisée pour stocker les informations nécessaires à son exécution.

En général, l'algorithme de Dijkstra peut être implémenté en utilisant plusieurs types de structures de données,

chacune ayant ses avantages et ses inconvénients en termes de complexité et de performance.

INTRODUCTION

Dans ce projet, nous allons étudier et implémenter l'algorithme de Dijkstra en utilisant plusieurs types de structures de données, notamment la file de priorité basée sur un tas binaire, la liste chaînée et potentiellement d'autres structures de données.
Nous allons comparer les performances de chaque implémentation en termes de temps d'exécution et d'utilisation de la mémoire pour différents types de graphes.
Cette étude nous permettra de mieux comprendre les avantages et les inconvénients de chaque structure de données et de proposer des perspectives d'amélioration pour l'algorithme de Dijkstra.
En somme, ce projet vise à fournir une étude détaillée et une implémentation pratique de l'algorithme de Dijkstra en utilisant différentes structures de données, et à évaluer leurs performances pour résoudre des problèmes de plus court chemin dans un graphe pondéré.

Shortest Path Problems:
Dijkstra's Algorithm

FONDEMENTS THEORIQUES

1. Présentation des concepts de graphe, de poids d'arête et de plus court chemin

Les graphes sont des structures de données utilisées pour représenter des relations entre différents objets.
Un graphe est composé de nœuds (ou sommets) et d'arêtes (ou liens) qui relient ces nœuds. Les arêtes peuvent être pondérées, c'est-à-dire qu'elles peuvent avoir un poids qui représente une certaine mesure, comme la distance, le temps, ou le coût, associé à la relation entre les nœuds.
Le poids d'une arête est une valeur numérique qui représente le coût ou la distance nécessaire pour passer de l'un des nœuds à un autre.
Le plus court chemin est le chemin le plus rapide, le plus court, ou le moins coûteux pour passer d'un nœud à un autre dans un graphe pondéré.

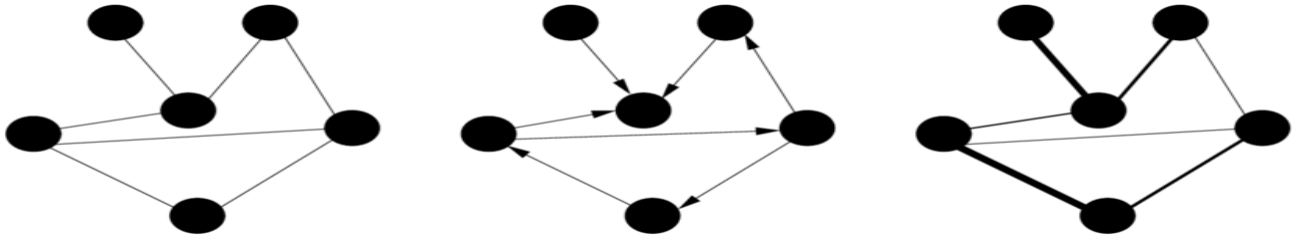
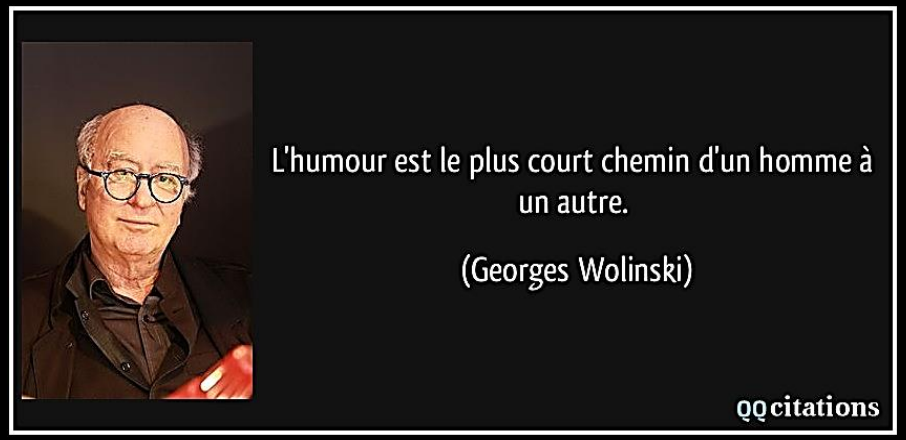


Fig. A.1 -Différents types de graphes : graphe non orienté (` a gauche), graphe orienté (au centre) et graphe pondéré (` a droite)

La théorie des graphes est une branche des mathématiques et de l'informatique qui consiste à modéliser différents problèmes de la vie réelle sous forme de graphes.
L'une des utilisations les plus classiques est la modélisation d'un réseau routier entre différentes villes. L'une des problématiques principales étant l'optimisation des distances entre deux points. Pour trouver le plus court chemin , on utilise souvent l'algorithme de Dijkstra**.**

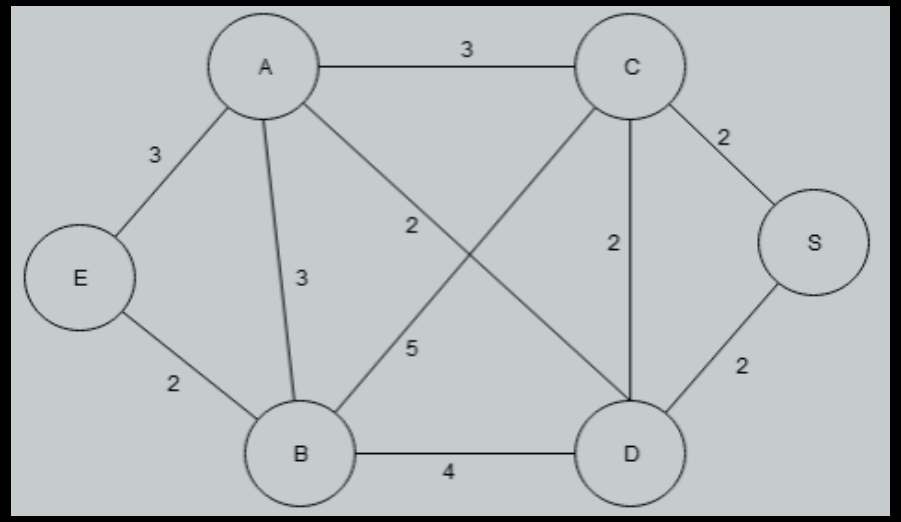
2. Explication de l'algorithme de Dijkstra

Le problème du plus court chemin ou problème de cheminement, a pour objectif de trouver le chemin le plus court entre deux points. Cela peut s'agir de la distance entre les points d'origine et de destination ou bien le temps écoulé d'un point à l'autre.
Le problème du plus court chemin possède de nombreuses applications: Problèmes d'optimisation de réseaux (routiers, télécommunications) certains problèmes d'investissements et de gestion de stocks certains problèmes en intelligence artificielle.

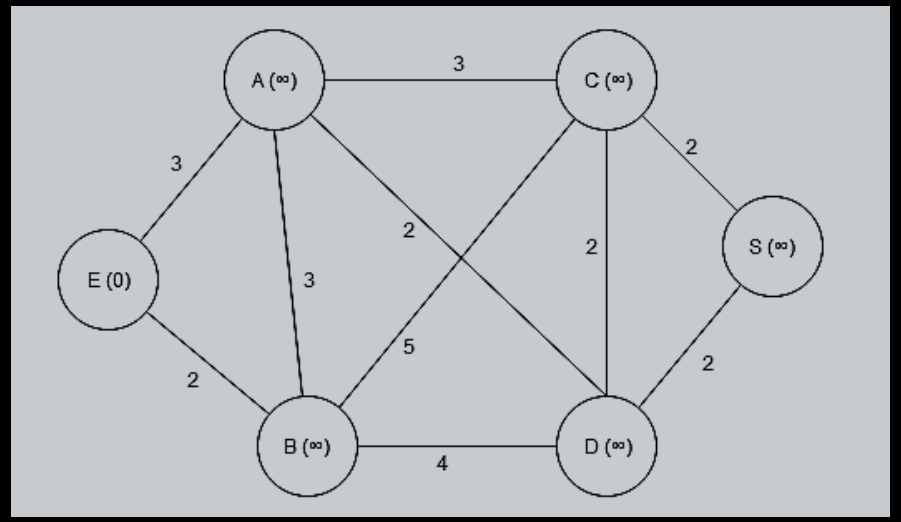


3. exemple explicatif

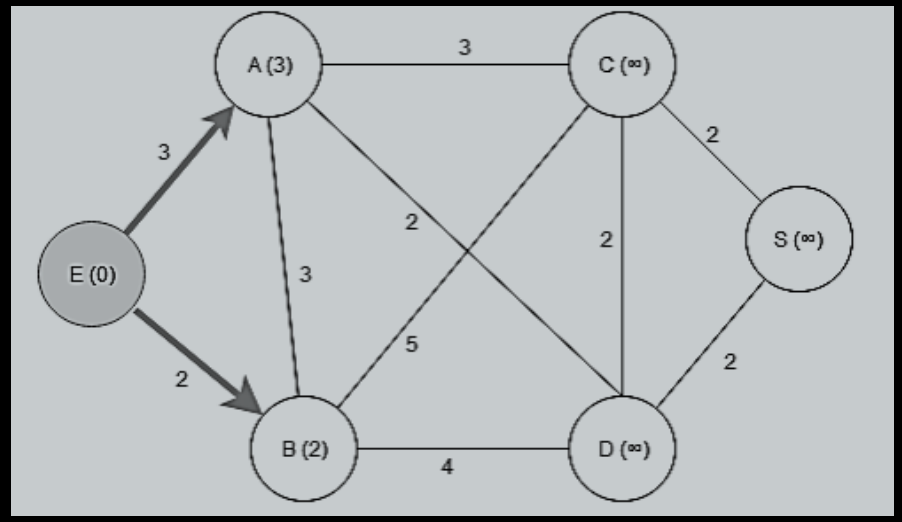
Voici un graphe pondéré connexe. On cherche le plus court chemin emmenant du sommet E au sommet S.



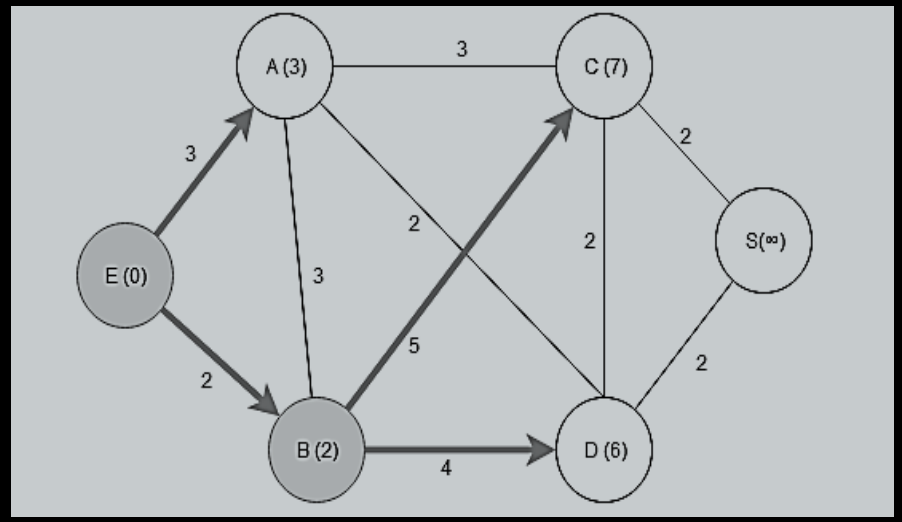
Initialisation:



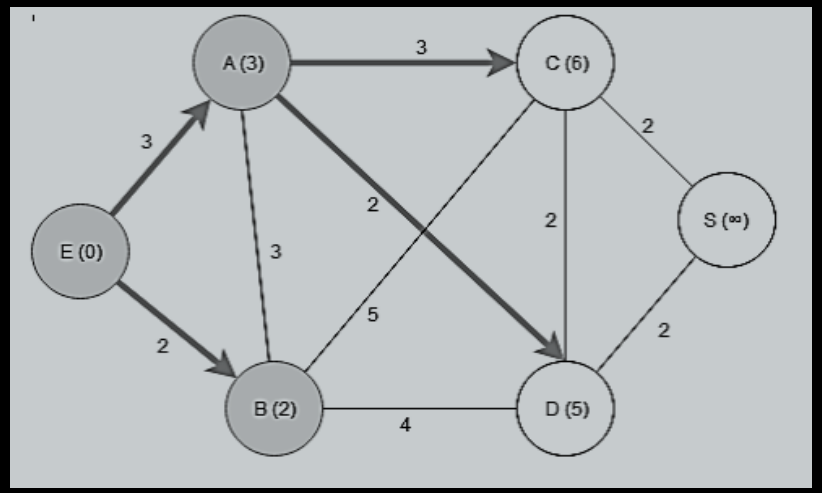
On sélectionne E et on actualise la marque de ses voisins:



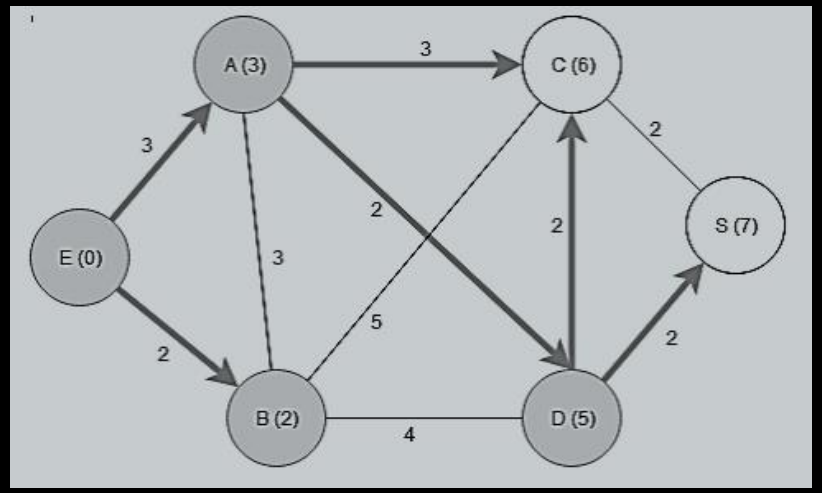
Parmi les non sélectionnés, on sélectionne la marque la plus petite et on traite ses voisins non sélectionnés.
La marque du sommet A est inchangée puisque marque(B)+poids(B-A) > marque(A)



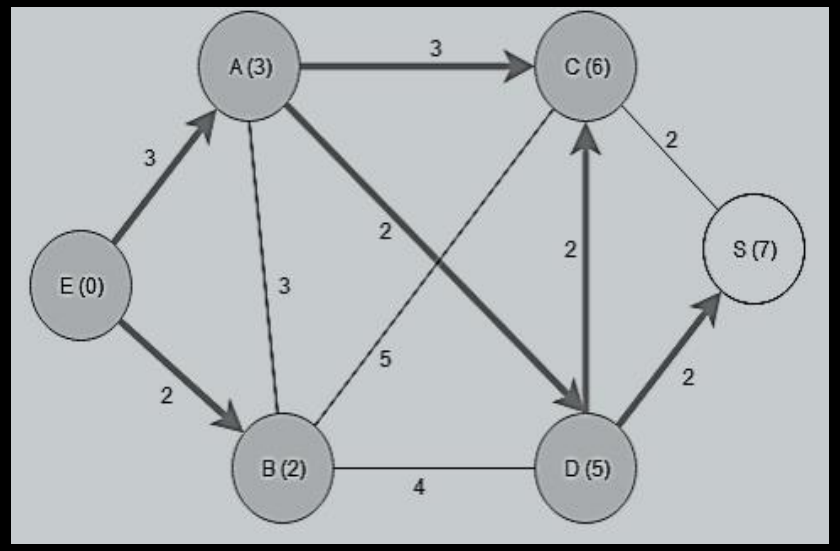
Parmi les non sélectionnés, on sélectionne la marque la plus petite et on traite ses voisins non sélectionnés. La marque de C est modifiée puisque marque(A)+poids(A-C) < marque(C). Idem pour la marque de D. On modifie donc la couleur des arêtes.



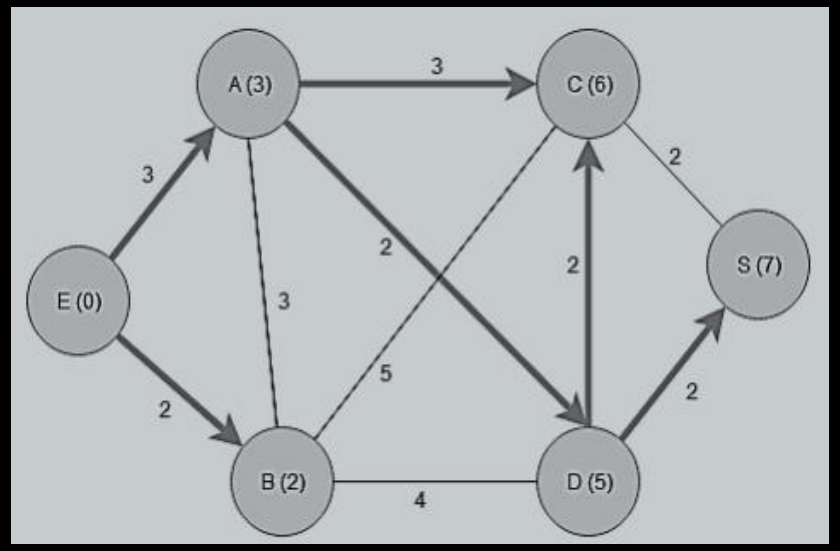
Etape suivante:



Etape suivante:



Dernière étape:



Interprétation: la longueur minimale de E à S est 7 et en remontant les flèches violettes, on sait que le chemin correspondant est E-A-D-S.

On peut présenter la mise en œuvre de l'algorithme de Moore-Dijkstra de la manière suivante (chaque ligne correspond à une étape de l'algorithme):

E	A	B	C	D	S	Sommet sélectionné
0	∞	∞	∞	∞	∞	E
	0 + 3 = 3(E)	0 + 2 = 2(E)	∞	∞	∞	B
	~~2 + 3 = 5~~ 3(E)		2 + 5 = 7(B)	2 + 4 = 6(B)	∞	A
			3 + 3 = 6(A)	3 + 2 = 5(A)	∞	D
			~~5 + 2 = 7~~ 6(A)		5 + 2 = 7(D)	C
					~~6 + 2 = 8~~ 7(D)	S

Pour déterminer le résultat, on lit la chaîne à l'envers (en partant du sommet S) en se rendant dans la colonne du sommet de provenance.

Ainsi, la chaîne à l'envers est :
S (poids total = 7 en venant de D)
D (poids = 5 en venant de A)
A (poids = 3 en venant de E)
E (sommet d'entrée)
La chaîne de poids minimal est donc E-A-D-S

4. Analyse de la complexité de l'algorithme

L'algorithme de Dijkstra garantit que la distance enregistrée pour chaque nœud est la plus courte distance connue à ce nœud à partir du nœud de départ.
La complexité temporelle de l'algorithme est O(E log V), où E est le nombre d'arêtes et V est le nombre de nœuds dans le graphe.
Cela en fait un algorithme efficace pour les graphes de taille modérée, mais sa complexité peut devenir prohibitivement élevée pour les graphes très grands.

4. quelques perspectives d'amélioration pour l'algorithme de Dijkstra :

- Utilisation de structures de données optimisées : L'algorithme de Dijkstra peut être amélioré en utilisant des structures de données optimisées pour le stockage des données. Par exemple, l'utilisation d'un tas de Fibonacci peut améliorer la complexité temporelle de l'algorithme.
- Optimisation de la recherche de nœuds : L'algorithme de Dijkstra peut être optimisé en réduisant le temps nécessaire pour la recherche de nœuds. Par exemple, l'utilisation de tables de hachage peut accélérer la recherche de nœuds dans le graphe.
- Utilisation de techniques de prétraitement : Il est possible d'optimiser l'algorithme de Dijkstra en utilisant des techniques de prétraitement pour réduire le temps nécessaire pendant l'exécution de l'algorithme. Par exemple, en utilisant une approche de division et conquête pour précalculer les chemins les plus courts entre certains nœuds.
- Utilisation d'heuristiques : L'algorithme de Dijkstra peut être amélioré en utilisant des heuristiques pour guider la recherche des nœuds les plus proches du nœud de départ. Par exemple, l'algorithme A* utilise une heuristique pour guider la recherche des nœuds les plus proches du nœud de départ, ce qui peut réduire le temps nécessaire pour trouver le chemin le plus court.
- Parallélisation de l'algorithme : L'algorithme de Dijkstra peut être parallélisé pour accélérer le temps nécessaire pour trouver le chemin le plus court. Par exemple, en utilisant des techniques de calcul parallèle pour exécuter l'algorithme sur plusieurs cœurs ou processeurs.

STRUCTURES DE DONNÉES

Les structures de données sont des moyens de stocker et d'organiser des données dans un système informatique. Elles sont essentielles pour le traitement efficace des données dans de nombreux domaines, tels que la programmation, la gestion de bases de données, l'analyse de données, l'intelligence artificielle, etc.

Le fonctionnement des structures de données dépend du type de structure utilisé. Il existe plusieurs types de structures de données, tels que les tableaux, les listes, les piles, les files d'attente, les arbres, les graphes, etc. Chaque type de structure a ses propres règles et opérations pour stocker et accéder aux données.

Dans cette partie, nous allons traiter le fonctionnement, les avantages, et les inconvénients de trois structures de données que nous allons les utiliser dans l'implémentation de l'algorithme de **Dijkstra** : les listes chainées, les tables de hachage, et les files de priorité basées sur un tas binaire .

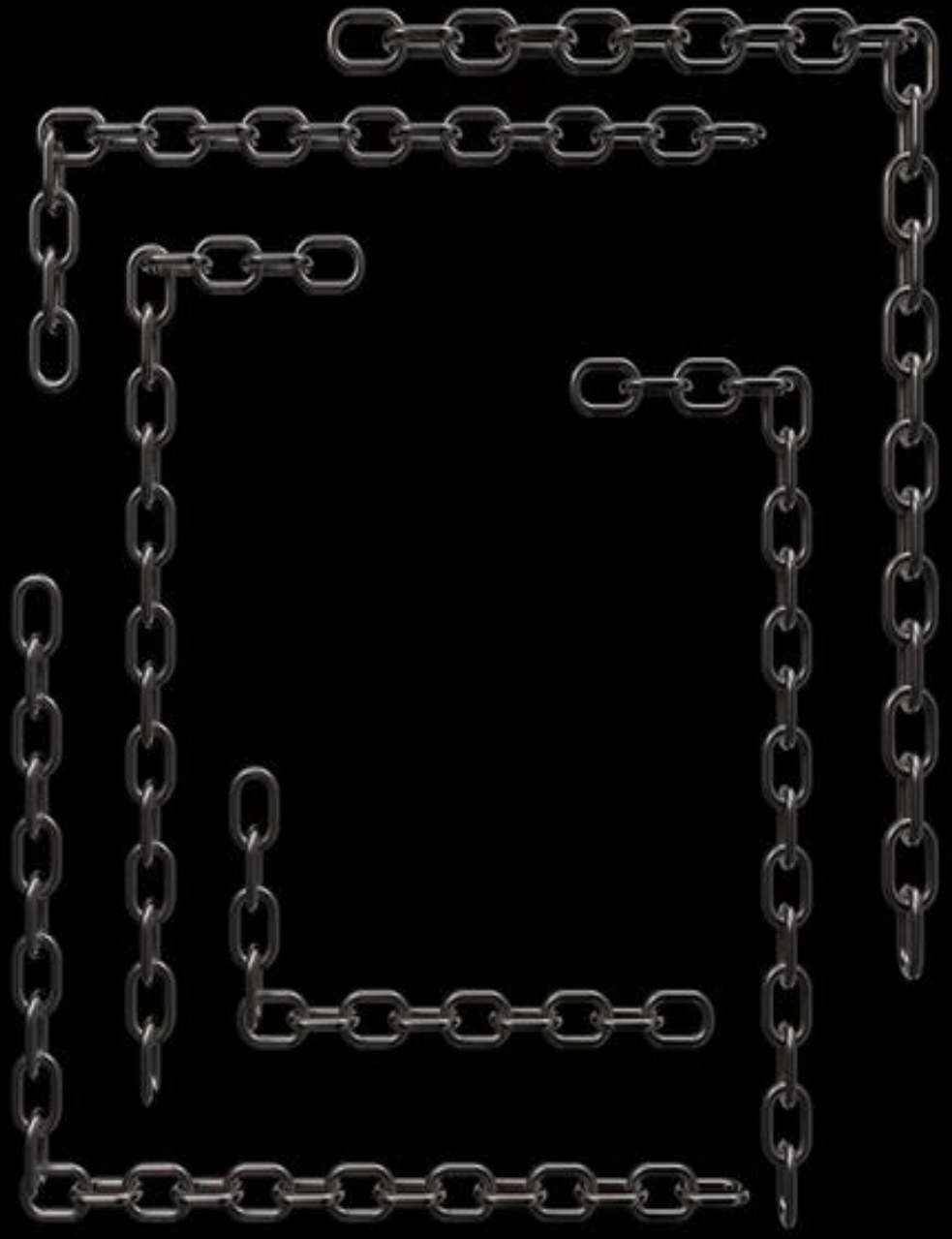
1. Listes chaînées :

-Une liste chaînée est une structure de données linéaire dans laquelle les éléments sont stockés dans des nœuds liés les uns aux autres par des pointeurs.

- Chaque nœud contient une donnée et un pointeur vers le nœud suivant dans la liste. Le premier nœud de la liste est appelé "tête" et le dernier est appelé "queue".

- Les opérations d'insertion et de suppression dans une liste chaînée sont rapides, car elles ne nécessitent aucun déplacement d'éléments et peuvent être effectuées en temps constant.

- Cependant, les opérations de recherche dans une liste chaînée peuvent être plus lentes que dans une table de hachage, car il faut parcourir la liste à partir de son début pour atteindre un nœud donné.



2. Tables de hachage :

- Une table de hachage est une structure de données qui permet de stocker et de rechercher efficacement des éléments en utilisant une fonction de hachage pour calculer l'indice de stockage

- Les éléments sont stockés dans un tableau et chaque élément est associé à une clé unique qui est utilisée pour calculer l'indice de stockage à l'aide de la fonction de hachage.

- En général, la fonction de hachage est conçue de manière à minimiser les collisions, qui se produisent lorsque deux éléments différents sont associés au même indice de stockage.

- Les opérations de recherche dans une table de hachage sont très rapides, car elles peuvent être effectuées en temps constant, mais les collisions peuvent affecter les performances et la consommation de mémoire.

3. Files de priorité basées sur un tas binaire :

- Une file de priorité est une structure de données qui permet de stocker et de récupérer des éléments en fonction de leur priorité.

- Une file de priorité basée sur un tas binaire est une implémentation spécifique de la file de priorité qui utilise une structure de tas binaire pour stocker les éléments.

- Dans un tas binaire, chaque nœud a une valeur qui est supérieure ou égale à la valeur de ses enfants. Les éléments sont stockés dans le tas binaire en fonction de leur priorité, de sorte que l'élément le plus prioritaire est toujours à la racine du tas.

- Les opérations d'insertion et de recherche dans une file de priorité basée sur un tas binaire sont très rapides en temps logarithmique, mais les opérations de suppression peuvent être plus lentes que dans une liste chaînée, car elles nécessitent de réorganiser la structure du tas binaire.

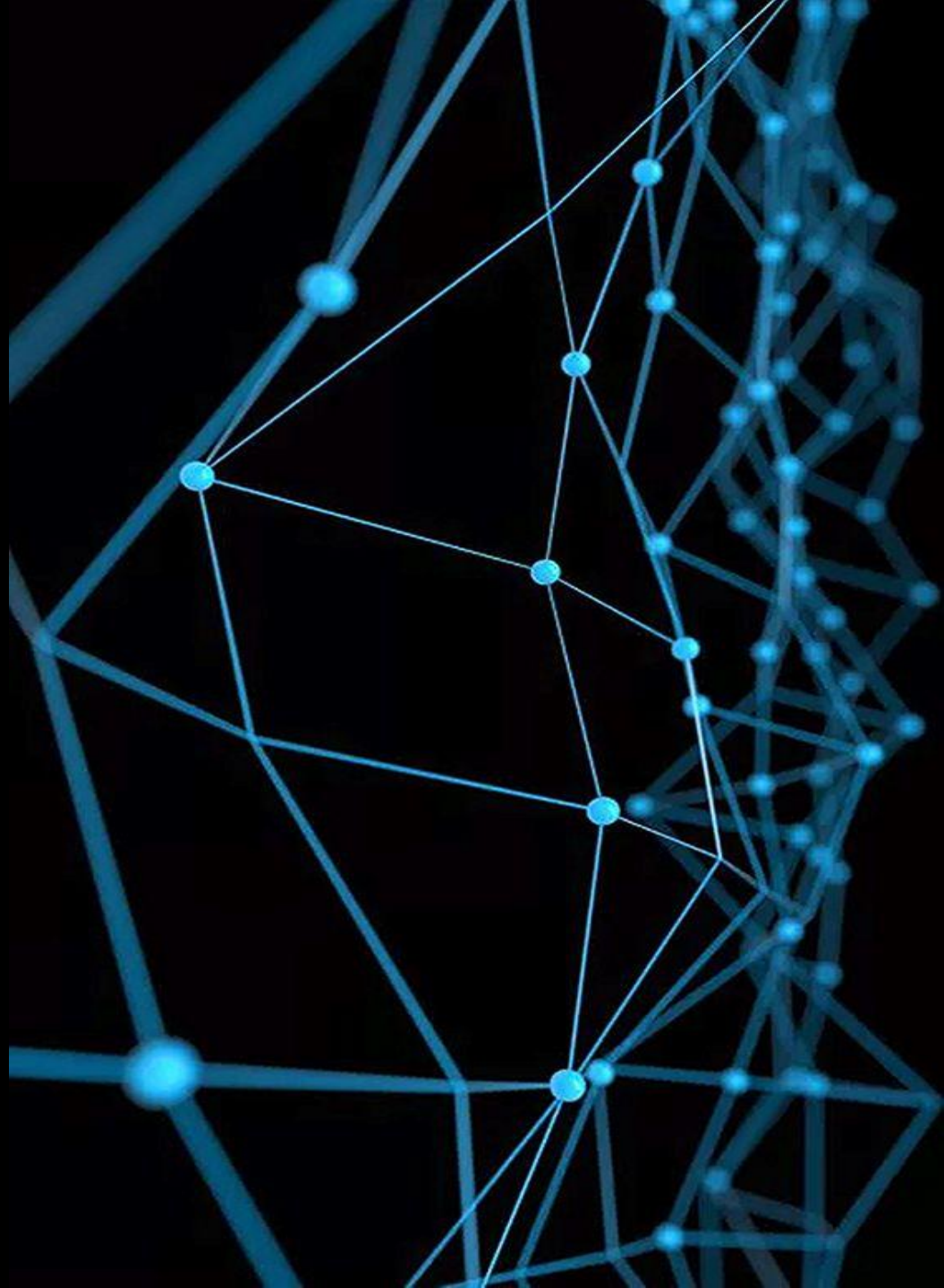
CONCLUSION

Enfin de compte, l'étude comparative de différentes structures de données pour l'implémentation de l'algorithme de Dijkstra montre que chaque structure de données a ses avantages et ses inconvénients en termes de temps d'exécution, d'utilisation de la mémoire et de complexité.

Le choix de la structure de données dépend des besoins spécifiques du programme et des compromis entre performances et complexité.

La file de priorité basée sur un tas binaire est généralement la structure de données la plus efficace pour l'implémentation de l'algorithme de Dijkstra, mais la liste chaînée et la table de hachage peuvent également être utilisées dans des contextes spécifiques.

Y a-t-il des questions concernant notre sujet ?



BIBLIOGRAPHIE

- https://perso.liris.cnrs.fr
- https://zestedesavoir.com/tutoriels/681/a-la-decouverte-des-algorithmes-de-graphe/727_bases-de-la-theorie-des-graphes/3352_graphes-et-representation-de-graphe/#1-10500_dessine-moi-un-graphe
- http://yallouz.arie.free.fr/terminale_cours/graphes/graphes.php?page=g3
- https://www.pinterest.fr
- https://math.univ-lyon1.fr/irem/Formation_ISN/formation_parcours_graphes/dijkstra/1_algorithme.html#
- https://www.techno-science.net/definition/6470.html
- https://datascientest.com/algorithme-de-dijkstra
- https://www.pinterest.fr
- https://www.programiz.com/dsa/hash-table

La fin de notre projet

MERCI

Nous tenons à exprimer notre profonde gratitude envers notre professeur Yassine Sadqi et nos collègues (SMI S4) pour leur soutien et leur contribution à notre parcours d'apprentissage. Merci de nous avoir inspirées, encouragées et aidées à atteindre nos objectifs académiques et personnels. Nous sommes fiers de faire partie de cette communauté d'apprentissage et nous sommes reconnaissantes pour les liens que nous avons créés ensemble.