***UNIVERSITE SIDI MOHAMED BEN ABDELLAH***

***جامعة سيدي محمد بن عبد الله***

**كلية العلوم ظهر المهراز**

**- فاس-**

***Faculté des Sciences Dhar El Mahraz – Fès***

***Année universitaire 2023-2024***

**Master Big Data Analytics & Smart System**

**Module: Théorie des graphes**

Une image contenant habits, chaussures, dessin humoristique, art

Description générée automatiquement

Réaliser Par :

**ELMAHDAOUI Abdelghani**

Réalisation d'une application utilisant l'algorithme A\* pour déterminer le chemin le plus court à Fès.

Remerciement

On remercie dieu le tout puissant de nous avoir donné la santé et la volonté d’entamer et de terminer ce projet.

C’est un plaisir et un moment très agréable de rendre hommage et de formuler des remerciements aux personnes qui, d’une manière ou d’une autre, ont apporté leur soutien et contribué à finaliser ce travail.

Tout d’abord, ce travail ne serait pas aussi riche et n’aurait pas pu avoir le jour sans l’aide et l’encadrement de **Mr. Bennani Mohamed Taj**, on le remercie pour la qualité de son encadrement exceptionnel, pour sa patience, sa rigueur et sa disponibilité durant notre préparation de ce mémoire.

Notre remerciement s’adresse également à tous nos professeurs pour leurs générosités et la grande patience dont ils ont su faire preuve malgré leurs charges académiques et professionnelles. Nos profonds remerciements vont également à toutes les personnes qui nous ont aidés et soutenue de près ou de loin.

Résumé

Le travail présenté dans le cadre d'un mini projet, est d’implémenter l’algorithme de **A\*** en python pour trouver le plus court chemin entre un état de départ et un état but, cet algorithme prend une matrice adjacente de la ville FES.

Table des matières

[Introduction Générale : 6](#_Toc158226852)

[Chapitre I . Généralités 8](#_Toc158226853)

[1. Historique de A\* : 8](#_Toc158226854)

[2. Définition et de Description de l’algorithme de A\* : 8](#_Toc158226855)

[a. Définition de l’algorithme A\* : 8](#_Toc158226856)

[b. Description de l’algorithme A\* : 9](#_Toc158226857)

[3. Avantage et inconvénient de l’algorithme de A\* : 10](#_Toc158226858)

[a. Les Avantages : 10](#_Toc158226859)

[b. Les inconvénients : 12](#_Toc158226860)

[4. Analyse de la complexité : 13](#_Toc158226861)

[Chapitre II. Présentation de l’algorithme 14](#_Toc158226862)

[1. Introduction : 14](#_Toc158226863)

[2. Exemple d’application : 14](#_Toc158226864)

[3. Propriétés de l'algorithme A\* : 15](#_Toc158226865)

[a. Heuristique Admissible : 15](#_Toc158226866)

[b. Heuristique Consistante : 15](#_Toc158226867)

[Chapitre III. Implémentation de l’algorithme 16](#_Toc158226868)

[1. Application de l’algorithme A\* sur la carte de FES : 16](#_Toc158226869)

[a. Définition de graphe : 16](#_Toc158226870)

[b. Fonctions utilisées dans A\* : 16](#_Toc158226871)

[2. Implémentation de l’algorithme : 17](#_Toc158226872)

[a. Les bibliothèques : 17](#_Toc158226873)

[b. La matrice adjacente : 18](#_Toc158226874)

[c. Le Dictionnaire : 19](#_Toc158226875)

[d. Recherche de plus cours chemin avec A\* : 20](#_Toc158226876)

[3. Exemple d’exécution : 22](#_Toc158226877)

[Conclusion Générale 24](#_Toc158226878)

# Introduction Générale :

Au cours du module de théorie des graphes, notre exploration approfondie des graphes en tant que structures de données fondamentales nous a permis de comprendre leur rôle crucial dans la représentation des relations entre des objets. De la recherche en largeur et en profondeur à des algorithmes plus sophistiqués comme l'algorithme de Dijkstra et l'algorithme A\*, nous avons étudié un large éventail de techniques qui constituent le cœur de la théorie des graphes. Ces connaissances ne se limitent pas à un domaine spécifique, mais trouvent des applications essentielles dans de nombreux secteurs, de la logistique à l'informatique en passant par la gestion de réseaux.

Fort de cette compréhension théorique acquise au fil de notre exploration des intrications des graphes, nous avons ressenti le besoin impérieux de traduire ces connaissances en solutions tangibles pour des problèmes concrets de notre quotidien. Ainsi est né notre projet : concevoir une application innovante, une passerelle entre la théorie des graphes et les défis de la vie réelle. Notre ambition était de créer un outil fonctionnel capable de mettre en œuvre l'un des algorithmes de recherche sur les graphes étudiés et de l'appliquer de manière pertinente à un problème de navigation dans un environnement urbain. Pour ce faire, nous avons choisi la carte de la ville de Fès comme terrain d'application, avec pour objectif de résoudre un problème concret : trouver le chemin le plus court entre deux localisations.

Ce rapport sera divisé en quatre parties principales pour fournir une vue d'ensemble complète de notre projet :

* ***Généralités :*** Présentation du contexte général du projet, de ses objectifs et de son importance.
* ***Présentation de l'algorithme :*** Explication détaillée de l'algorithme de recherche sur les graphes choisi, en l'occurrence l'algorithme de A\*.
* ***Implémentation de l'algorithme :*** Description du processus de développement de l'application, de la création du graphe représentant la carte de Fès à l'application de l'algorithme pour trouver le chemin le plus court.
* ***Conclusion :*** Récapitulation des principaux points abordés dans le rapport, évaluation des résultats obtenus et réflexion sur les implications de notre travail.

# Chapitre I . Généralités

## Historique de A\* :

L'algorithme A\* a été développé par Peter Hart, Nils Nilsson et Bertram Raphael en 1968. Il a été conçu comme une amélioration de l'algorithme Dijkstra, visant à trouver le chemin le plus court dans un graphe pondéré tout en utilisant une heuristique pour guider la recherche de manière plus efficace.

L'algorithme A\* est devenu un élément fondamental dans le domaine de l'intelligence artificielle et de la résolution de problèmes de chemin optimal. Il est largement utilisé dans divers domaines, y compris la robotique, les jeux vidéo et la planification de trajectoires.

L'algorithme A\* continue d'être un sujet de recherche actif. Les chercheurs travaillent à l'amélioration de l'efficacité de l'algorithme, à son extension à des graphes plus complexes et à son application à de nouvelles domaines.

## Définition et de Description de l’algorithme de A\* :

### Définition de l’algorithme A\* :

L'algorithme A\* est un algorithme de recherche de chemin utilisé dans les domaines de l'intelligence artificielle et de théorie des graphes. Il est conçu pour trouver le chemin le plus court entre deux points dans un graphe pondéré, tout en utilisant une heuristique pour guider la recherche de manière efficace.

L'algorithme A\* fonctionne en conservant une liste de nœuds ouverts, qui sont les nœuds qui n'ont pas encore été explorés mais qui pourraient être sur le chemin le plus court vers le nœud final. Pour chaque nœud ouvert, l'algorithme calcule une estimation du coût du chemin le plus court du nœud initial à ce nœud. Le nœud avec l'estimation du coût la plus faible est ensuite ajouté à une liste de nœuds fermés, qui sont les nœuds qui ont déjà été explorés.

L'algorithme A\* continue d'explorer le graphe jusqu'à ce qu'il trouve le nœud final. Si le nœud final est trouvé, l'algorithme retourne le chemin le plus court du nœud initial à ce nœud.

Voici une brève explication des composants clés de l'algorithme A\* :

1. Coût cumulatif (g) : Il représente le coût total pour atteindre un nœud donné depuis le nœud de départ en suivant le chemin actuel.

2. Estimation Heuristique (h) : C'est une estimation du coût restant pour atteindre le nœud cible depuis le nœud actuel. Cette heuristique doit être admissible, ce qui signifie qu'elle ne doit jamais surestimer le coût réel.

3. Coût Total (f) : Il est calculé en additionnant le coût cumulatif (g) et l'estimation heuristique (h). L'algorithme choisit le nœud avec le coût total le plus bas à chaque étape.

L'utilisation d'une heuristique permet à A\* de diriger la recherche vers les zones du graphe susceptibles de contenir la solution, ce qui améliore considérablement son efficacité par rapport à des méthodes de recherche non informées comme l'algorithme de Dijkstra

### Description de l’algorithme A\* :

Voici une description plus détaillée de l'algorithme A\* :

**Étape 1 : Initialisation**

L'algorithme A\* commence par ajouter le nœud initial à la liste de nœuds ouverts.

**Étape 2 : Exploration**

Tant que la liste de nœuds ouverts n'est pas vide :

Retirez le nœud avec l'estimation du coût la plus faible de la liste de nœuds ouverts et ajoutez-le à la liste de nœuds fermés.

Pour chaque successeur du nœud actuel :

Calculez l'estimation du coût du chemin le plus court du nœud initial au successeur.

Si l'estimation du coût du successeur est inférieure à l'estimation du coût du successeur dans la liste de nœuds ouverts, mettez à jour l'estimation du coût du successeur dans la liste de nœuds ouverts.

**Étape 3 : Fin**

Si le nœud final est trouvé dans la liste de nœuds fermés, l'algorithme retourne le chemin le plus court du nœud initial à ce nœud. Sinon, l'algorithme échoue.

* **Estimation du coût**

L'estimation du coût du chemin le plus court du nœud initial à un nœud donné est calculée à l'aide d'une fonction d'estimation. La fonction d'estimation doit être non négative et croissante. Une fonction d'estimation courante est la fonction d'estimation euclidienne , qui est définie comme suit : ***f(n) = d(n, n\_f)***

où :

n est le nœud dont l'estimation du coût doit être calculée

n\_f est le nœud final

d(n, n\_f) est la distance euclidienne entre n et n\_f

## Avantage et inconvénient de l’algorithme de A\* :

### Les Avantages :

L'algorithme A\* présente plusieurs avantages qui en font un choix populaire dans de nombreuses applications, notamment dans la recherche de chemins dans les graphes pondérés. Voici quelques-uns de ses principaux avantages :

* ***Efficacité :*** L'algorithme A\* est généralement plus efficace que des approches non informées, comme l'algorithme de Dijkstra, car il utilise une heuristique pour guider la recherche vers les zones les plus prometteuses du graphe.
* ***Optimalité :*** Sous certaines conditions, l'algorithme A\* garantit de trouver la solution optimale, c'est-à-dire le chemin le plus court, en explorant sélectivement les nœuds du graphe.
* ***Adaptabilité :*** L'algorithme A\* peut être appliqué à une grande variété de problèmes de recherche de chemin, y compris des problèmes avec des obstacles en ajustant la fonction heuristique. Cela le rend polyvalent et applicable à une variété de domaines, tels que la planification de trajectoires, la résolution de jeux, la navigation robotique, etc.
* ***Complexité maîtrisée :*** La complexité temporelle de l'algorithme A\* est généralement raisonnable, surtout avec l'utilisation de structures de données appropriées, ce qui le rend applicable à des graphes de taille modérée.
* ***Ajustement du coût :*** L'algorithme A\* permet de prendre en compte des coûts différents pour différentes actions ou arêtes du graphe, ce qui le rend adaptable à des situations où certains chemins peuvent être plus coûteux que d'autres.

**Exemple des avantages de A\* :**

* Dans la navigation robotique, l'algorithme A\* peut être utilisé pour trouver le chemin le plus court pour un robot pour se déplacer d'un point à un autre dans un environnement avec des obstacles.
* Dans la planification de routes, l'algorithme A\* peut être utilisé pour trouver le meilleur itinéraire pour un véhicule pour se déplacer d'un point à un autre dans un réseau routier.
* Dans le jeu vidéo, l'algorithme A\* peut être utilisé pour contrôler les mouvements des agents intelligents, tels que des ennemis ou des personnages non-joueurs.

### Les inconvénients :

Bien que l'algorithme A\* présente de nombreux avantages, il comporte également certains inconvénients. Voici quelques-uns des points à prendre en compte :

* ***Complexité :*** L'algorithme A\* a une complexité en temps de O(b^d), où b est la largeur du graphe et d est la profondeur du chemin le plus court. Cela signifie que l'algorithme peut être lent pour les graphes larges ou profonds.
* ***Utilisation de la mémoire :*** L'algorithme A\* nécessite de stocker une liste de tous les nœuds ouverts. Cela peut nécessiter beaucoup de mémoire pour les graphes larges.
* ***Heuristique admissible nécessaire :*** Pour garantir l'optimalité de l'algorithme, une heuristique admissible est nécessaire. Si l'heuristique surestime le coût restant, cela peut conduire à des solutions non optimales.
* ***Dépendance du choix de l'heuristique :*** La qualité de la solution trouvée dépend en grande partie de la qualité de l'heuristique choisie. Trouver une heuristique appropriée peut être un défi, et une mauvaise heuristique peut conduire à des résultats moins optimaux.
* ***Non adapté aux graphes non connexes :*** L'algorithme A\* est conçu pour des graphes connexes. S'il y a des composantes non connexes dans le graphe, des ajustements sont nécessaires pour traiter ces situations.
* ***Sensibilité aux obstacles :*** Bien que l'A\* puisse gérer des obstacles, il peut ne pas être aussi efficace dans des environnements dynamiques où les obstacles changent fréquemment.

**Exemple des inconvénients de A\* :**

* Dans la navigation robotique, l'algorithme A\* peut être lent pour trouver le chemin le plus court dans un environnement complexe avec de nombreux obstacles.
* Dans la planification de routes, l'algorithme A\* peut nécessiter beaucoup de mémoire pour stocker la liste de tous les itinéraires possibles.
* Dans le jeu vidéo, l'algorithme A\* peut ne pas trouver le chemin le plus court si la fonction d'estimation est mal choisie.

Malgré ces inconvénients, A\* reste largement utilisé et ses performances peuvent souvent être améliorées ou adaptées en fonction des spécificités du problème à résoudre.

## Analyse de la complexité :

La complexité de l'algorithme A\* dépend de plusieurs facteurs, notamment la taille du graphe, la méthode de représentation du graphe (matrice d'adjacence, liste d'adjacence, etc.), et la manière dont les coûts de déplacement entre les nœuds sont calculés.

En général, la complexité de l'algorithme A\* peut être exprimée comme suit:

- Dans le pire des cas, la complexité temporelle de l'algorithme A\* est exponentielle en fonction de la longueur du chemin optimal.

- Cependant, dans des conditions normales et avec une heuristique bien choisie, la complexité temporelle de l'algorithme A\* est souvent beaucoup plus proche de la complexité linéaire, ce qui le rend très efficace dans de nombreuses situations.

En termes de complexité spatiale, l'algorithme A\* stocke généralement les informations relatives aux nœuds visités et à la frontière de recherche. La complexité spatiale dépend donc également de la taille du graphe et du nombre de nœuds visités.

En résumé, bien que la complexité de l'algorithme A\* puisse varier en fonction des facteurs mentionnés ci-dessus, il est généralement considéré comme un algorithme efficace pour la recherche de chemin dans de nombreux problèmes pratiques, en particulier lorsque des heuristiques appropriées sont utilisées pour guider la recherche.

# Chapitre II. Présentation de l’algorithme

## Introduction :

Cette partie présente seulement un exemple d’application de l’algorithme de A\*.

## Exemple d’application :

**Routes entre les villes :**  
V0 : Ville de départ

V6: Destination

h : Distance à vol d’oiseaux

C : Distance réel entre deux villes

h(V0)

**9**

C(V1,V2)

1

1

7

**2**

4

4

**0**

2

**3**

3

**2**

4

**2**

**5**

2

**Contenu de Open à chaque itération ( état , f , parent ) :**

1. (V0 , 9 , - )
2. (V1 ,5 , V0) , (V2 , 6 , V0) , ( V3 , 7 , V0 )
3. (V2 , 6 , V0) , ( V3 , 7 , V0 ) , (V5 , 12 , V1)
4. ( V3 , 7 , V0 ) , ( V4 , 9 , V2 ) , (V5 , 12 , V1)
5. (V2 , 5 , V3) , (V4 , 6 , V3) , (V5 , 12 , V1)
6. (V4 , 6 , V3) , (V5 , 12 , V1)
7. (V6 , 7 , V4) , (V5 , 12 , V1)
8. Solution : V0 - V3 - V4 - V6

**Contenu de Closed à la sortie (nœud , f):**

(V4 , 6) , ( V3 , 7 ) , (V2 , 5 ) , (V1 ,5 ) , (V0 , 9 )

## Propriétés de l'algorithme A\* :

### Heuristique Admissible :

h ne surestime jamais le cout réel : h(n) C(n,nf)

Dans l’Example en haut nous avons la relation d’admissibilité est vérifié

n { V0 , V1 , V2 , V3 , V4 , V5 , V6 }

### Heuristique Consistante :

un neud n1 et n2 son successeur h(n1) C(n1,n2) + h(n2)

Prendrons l’Example de noud V0 et V1 ,

nous avons h(V0) = 9 C(V0 , V1 ) + h(V1) = 3 + 2 = 5

Alors l’heuristique n’est pas Consistante

# Chapitre III. Implémentation de l’algorithme

## Application de l’algorithme A\* sur la carte de FES :

### Définition de graphe :

Le réseau routier de la ville de Fès est représenté sous forme de graphe, où chaque nœud correspond à un emplacement géographique défini par ses coordonnées latitude et longitude. Les arêtes du graphe représentent les connexions routières entre ces emplacements, modélisant ainsi les itinéraires possibles à travers la ville. Cette représentation graphique est générée à l'aide de la bibliothèque OSMnx, qui extrait les données du réseau routier d'OpenStreetMap et les traduit en une structure de graphe utilisable dans le contexte de l'algorithme A\*.

### Fonctions utilisées dans A\* :

g(n) pourrait être la longueur réelle de la route parcourue depuis le point de départ jusqu'au nœud n.

h(n) pourrait être calculé en utilisant la fonction **ox.distance.euclidean\_dist\_vec** pour calculer la distance euclidienne entre les coordonnées géographiques de deux nœuds. Cette distance est utilisée comme estimation du coût restant pour atteindre le point d'arrivée

f(n) = g(n) + h(n) représente le coût total prévu pour parcourir le chemin du point de départ jusqu'au point d'arrivée en passant par le nœud n.

## Implémentation de l’algorithme :

### Les bibliothèques :

Une image contenant texte, capture d’écran, Police, conception

Description générée automatiquement

Cette section du code initialise un ensemble crucial de bibliothèques et de modules pour développer une application visant à trouver le plus court chemin dans le réseau routier de Fès. En résumé :

Le code commence par importer les bibliothèques **OSMnx**, **NetworkX, Geopy, et Folium,** indispensables pour extraire et analyser les données du réseau routier, géocoder les coordonnées, et générer des cartes interactives.

Il met en place l'interface graphique à l'aide des bibliothèques **Matplotlib** et **Tkinter**. Ces composants facilitent la création d'une fenêtre avec des libellés, des boutons, et des menus déroulants pour simplifier l'interaction avec l'utilisateur.

Les modules **os** et **webbrowser** sont inclus, suggérant probablement l'intention d'utiliser des fonctionnalités liées au système d'exploitation et à la navigation web, bien que leur utilisation précise reste à clarifier sans le reste du code.

### La matrice adjacente :

Une image contenant texte, capture d’écran, Système d’exploitation, logiciel

Description générée automatiquement

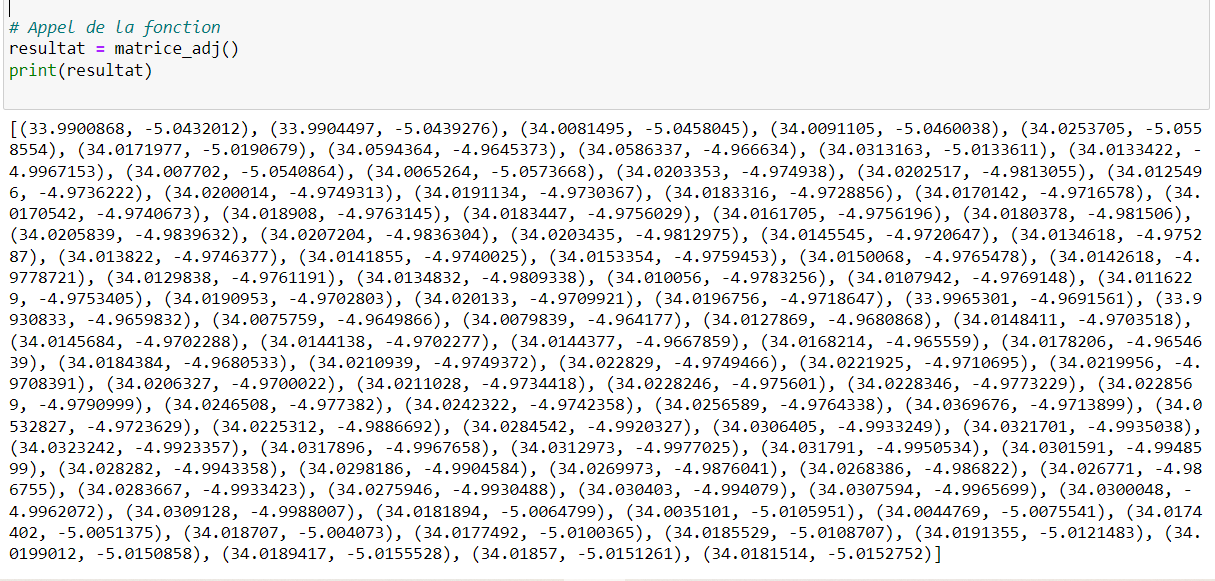
Cette section de code débute par définir le nom de la ville (place\_name) comme **"Fes, Morocco"** et utilise la bibliothèque OSMnx pour extraire le graphe du réseau routier de cette localité grâce à la fonction **ox.graph\_from\_place**. Les données du réseau routier sont ensuite stockées dans la variable graph1.

En suivant, une fonction nommée **matrice\_adj** est introduite. Cette fonction parcourt les nœuds du graphe, qui représentent des emplacements géographiques, à l'aide d'une boucle for. Pour chaque nœud, les coordonnées géographiques **(latitude et longitude)** sont extraites du dictionnaire data associer à chaque nœud, et ces coordonnées sont ensuite ajoutées à la liste M sous la forme d'un tuple **(latitude, longitude).**

La fonction possède également des paramètres optionnels tels que **max\_points\_to\_print, points\_printed**, et graph.

Le paramètre **max\_points\_to\_print** limite le nombre maximal de points à imprimer. La boucle s'interrompt une fois que ce nombre maximal de points a été atteint.

**Exécution de la fonction :**



### Le Dictionnaire :

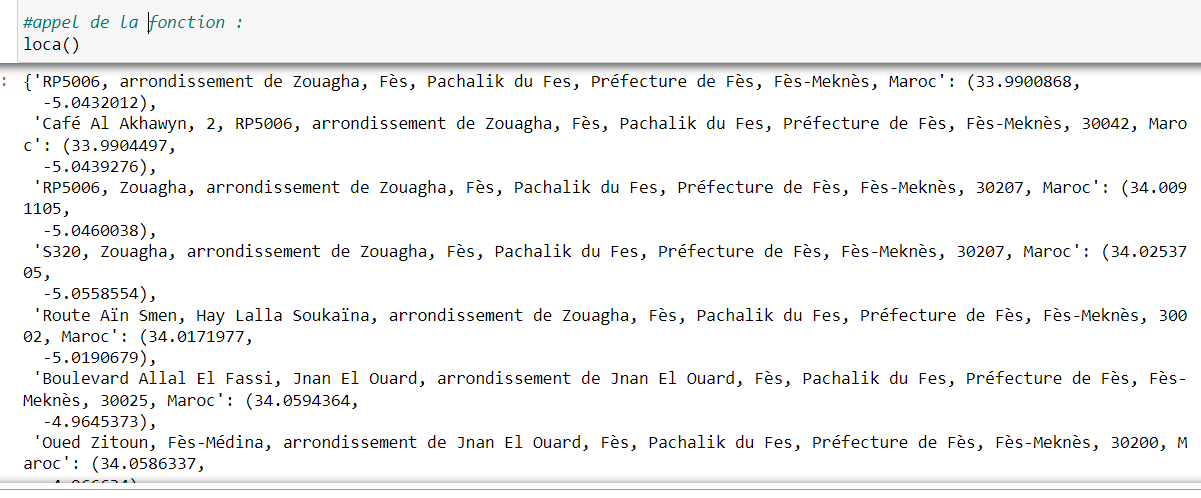
Une image contenant texte, capture d’écran, Police, Système d’exploitation

Description générée automatiquement

Dans cette section du code, le processus débute en initialisant un **géolocaliseur (geolocator)** à l'aide de la bibliothèque Geopy. La classe **Nominatim** est utilisée pour cette instance, avec une spécification de l'agent utilisateur en tant que "**my\_geocoder**". Ensuite, la fonction matrice\_adj est invoquée pour récupérer une liste de coordonnées géographiques (coordinates).

La fonction **loca** est alors définie, prenant ces coordonnées en tant qu'argument (coordinate) et instanciant un dictionnaire vide (**locations\_dict**) par défaut. Elle procède ensuite à une itération sur chaque paire de coordonnées dans la liste. Pour chaque paire, le géolocaliseur est utilisé pour obtenir l'adresse correspondante dans la langue spécifiée ("**fr**" pour le français) à l'aide de la méthode reverse. Cette adresse est par la suite ajoutée au dictionnaire **locations\_dict** avec les coordonnées comme valeur associée.

**Exécution de la fonction :**



### Recherche de plus cours chemin avec A\* :

Une image contenant texte, capture d’écran, nombre, document

Description générée automatiquement

Dans cette section du code, nous présentons une application graphique permettant de déterminer le chemin le plus court dans le réseau routier de la ville de Fès. Voici un résumé simplifié :

La classe **ShortestPathApp** est créée pour encapsuler l'application, avec son constructeur (**init**) appelé lors de l'instanciation. Cette méthode prend généralement comme argument la fenêtre principale de l'application.

La méthode **init** initialise l'interface utilisateur en intégrant des éléments tels que des libellés, des menus déroulants, et un bouton, facilitant ainsi la spécification du point de départ et d'arrivée pour la recherche du chemin le plus court.

La variable **locations\_dict** est préalablement définie via la fonction **loca()** avant d'être utilisée dans **init**. Cette fonction **géocode** les coordonnées géographiques des nœuds du graphe en adresses, créant un dictionnaire associant chaque adresse à ses coordonnées.

Lorsque l'utilisateur clique sur le bouton "Trouver le chemin le plus court", la méthode **find\_shortest\_path** est déclenchée. Elle récupère les points de départ et d'arrivée choisis par l'utilisateur, identifie les nœuds correspondants dans le graphe, puis applique l'algorithme A\* pour calculer le chemin optimal.

Le résultat est affiché graphiquement sur une carte générée avec les bibliothèques **OSMnx** et **Matplotlib**.

La méthode **get\_predefined\_places** retourne une liste de lieux prédéfinis extraits du dictionnaire **locations\_dict**, utilisée pour peupler les menus déroulants du point de départ et d'arrivée.

La section **if name == "main":** vérifie si le script est exécuté en tant que programme principal. Dans ce cas, une instance de la classe **ShortestPathApp** est créée, et la boucle principale (**root.mainloop()**) est activée pour lancer l'interface utilisateur.

## Exemple d’exécution :

Choisissons par exemple Café Al Akhawayn comme point de départ , et Zouhour 1 comme point d’arrivée

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

On obtient le résultat suivant :

Une image contenant texte, carte, atlas

Description générée automatiquement

# Conclusion Générale

Dans ce projet, nous avons exploré en profondeur l'algorithme A\* et son application sur la résolution du problème du chemin le plus court dans un graphe pondéré. Nous avons commencé par présenter l'historique de l'algorithme A\* et sa définition, ainsi que ses avantages et inconvénients. Ensuite, nous avons analysé la complexité de cet algorithme pour mieux comprendre ses performances.

Dans le cadre de ce projet, nous avons implémenté l'algorithme A\* pour trouver le chemin le plus court sur la carte de la ville de Fès. Nous avons défini le graphe à partir des données de la carte, puis nous avons utilisé différentes fonctions pour appliquer l'algorithme A\* et trouver le chemin optimal entre deux points. Nous avons également discuté des propriétés importantes de l'algorithme A\*, telles que l'heuristique admissible et l'heuristique consistante.

L'implémentation de l'algorithme A\* sur la carte de Fès a été couronnée de succès, et nous avons pu observer des résultats satisfaisants avec des temps d'exécution raisonnables. Cela démontre l'efficacité de l'algorithme A\* dans la résolution de problèmes de cheminement dans des environnements réels.

En conclusion, ce projet nous a permis de mieux comprendre les principes et les applications de l'algorithme A\* dans la résolution de problèmes de cheminement. Il nous a également donné l'occasion d'appliquer nos connaissances théoriques à un problème pratique et de tirer des conclusions significatives sur les performances de cet algorithme dans un contexte réel.