Calcul d'un cycle Eulérien

29 octobre 2024

—

Martin Tom, Roy Tom

Université d'Orléans

Master 1 **Applications Réparties, Intelligence Artificielle et Sécurité (ARIAS)**

1. Présentation de l’algorithme

Cet algorithme implémente plusieurs fonctions pour analyser et traiter les graphes non orientés, plus précisément sur les graphes eulériens, c'est-à-dire ceux où un cycle eulérien peut être trouvé. Un graphe est eulérien s'il est connexe et si tous ses sommets ont un degré pair. La fonction est_connexe vérifie si un graphe est connexe en utilisant une exploration de profondeur. La fonction VerifGrapheEulerien s'assure que tous les sommets ont un degré pair et que le graphe est connexe, confirmant ainsi s'il est eulérien.

L'algorithme de Fleury trouve un cycle eulérien. Il parcourt le graphe en choisissant des arêtes à chaque étape. L'algorithme évite les isthmes, sauf en cas d'obligation, pour éviter de briser la connexité du graphe. La fonction EstIsthme vérifie si une arête est un isthme en testant si sa suppression déconnecte les sommets qu'elle relie.

Enfin, TestCycleEulerien vérifie si une séquence donnée d'arêtes constitue bien un cycle eulérien en s'assurant que toutes les arêtes sont parcourues une seule fois, et nombre_aretes calcule le nombre total d'arêtes dans le graphe.

2. Analyse de la complexité

L’algorithme que nous proposons contient plusieurs fonctions. Pour analyser la complexité de l’algorithme nous allons tout découper en plusieurs fonctions.

a) Analyse en temps

Fonction est_connexe : Cette fonction explore chaque sommet une seule fois, ainsi que tous ses voisins, en passant par toutes les arêtes pour visiter les voisins. Elle est basée sur un parcours en profondeur, et sa complexité est de l’ordre de O(m+n), où m est le nombre de sommets et n le nombre d’arêtes.

Fonction VerifGrapheEulerien : Cette fonction a également une complexité de O(m+n). Dans la boucle principale, elle parcourt chaque sommet une fois, soit un temps de O(n), mais elle utilise est_connexe, qui a une complexité en temps de O(m+n). Cette dernière fonction domine donc le temps d'exécution de VerifGrapheEulerien.

Fonction TestCycleEulerien : Cette fonction comporte une unique boucle principale qui parcourt la liste des sommets pour les comparer avec le graphe. Elle s'exécute donc en temps O(n).

Fonction EstIsthme : Cette fonction vérifie si une arête est un isthme en la supprimant du graphe et en vérifiant la connexité du graphe modifié. La complexité en temps est de O(m+n), en raison de la copie du graphe et de la vérification de connexité. Cette vérification est essentielle pour des algorithmes comme Fleury, qui nécessitent une exploration sans couper la connexité.

Fonction Fleury : Cette fonction construit un cycle eulérien en supprimant progressivement les arêtes tout en évitant les isthmes lorsque possible. À chaque étape, elle appelle EstIsthme, ce qui rend la complexité en temps de l'algorithme O(n×(m+n)). Fleury est donc coûteux en temps, mais permet de trouver un cycle eulérien sans altérer la structure du graphe.

b) Analyse de l'espace

Fonction est_connexe : La fonction est_connexe utilise une liste atteint qui peut contenir jusqu'à n sommets et repose sur une récursion dont la profondeur maximale est également n, ce qui donne une complexité spatiale totale de O(n).

Fonction VerifGrapheEulerien : La fonction VerifGrapheEulerien effectue une copie profonde du graphe GGG, nécessitant O(n+m) d'espace, et appelle est_connexe, ce qui maintient une complexité spatiale globale de O(n+m).

Fonction TestCycleEulerien : La fonction TestCycleEulerien utilise une liste est_passe pour enregistrer les arêtes parcourues, ce qui requiert O(n) d'espace, tandis que le reste de ses

calculs se fait avec un espace constant. La fonction nombre_aretes n'utilise que de l'espace constant O(1) pour compter les arêtes à partir des listes d'adjacence.

Fonction EstIsthme : La fonction EstIsthme nécessite une copie complète du graphe, occupant O(n+m), et appelle également est_connexe, ce qui conduit à une complexité spatiale totale de O(n+m).

Fonction Fleury : L'algorithme de Fleury utilise également O(n+m) d'espace pour copier le graphe et O(n) pour construire la liste cycle, ce qui maintient sa complexité spatiale globale à O(n+m). La fonction lire_graphe_json, utilisée pour charger un graphe à partir d'un fichier JSON, consomme une mémoire proportionnelle à la taille du graphe, soit O(n+m).

3. Exécution du programme

Pour exécuter le code, placez le fichier contenant le code dans un répertoire local et assurez-vous que le fichier JSON du graphe, nommé graphes.json, se trouve dans le même répertoire. Exécutez le script en tapant python Eulerien.py. Le programme lira le graphe depuis le fichier JSON et exécutera les fonctions définies dans main(), affichant les résultats directement dans la console. Vous pouvez personnaliser les paramètres, comme le graphe ou la liste du cycle, en modifiant le fichier JSON ou les variables dans le code.

4. Choix d'implémentation

Le graphe est représenté comme un dictionnaire d'adjacence pour accéder rapidement aux voisins. Les fonctions s'appuient sur des principes simples : la récursion pour vérifier la connexité (est_connexe) et des copies du graphe pour éviter les modifications accidentelles. L'algorithme de Fleury utilise la fonction EstIsthme pour éviter de supprimer des arêtes critiques, garantissant la construction correcte du cycle eulérien.

Les principales difficultés ont concerné les limites de la récursion, la gestion des modifications du graphe, et le coût élevé de la vérification des isthmes. Ces problèmes ont été résolus avec des copies et des structures comme est_passe pour suivre les arêtes.

Des optimisations pour éviter les copies inutiles et réduire les calculs répétitifs dans Fleury sont possibles. Utiliser des bibliothèques aurait pu simplifier aussi certaines tâches.

L’implémentation est fonctionnelle et permet de vérifier les propriétés eulériennes, d'identifier les isthmes et de construire des cycles eulériens. Elle reste toutefois améliorable pour des graphes plus complexes ou volumineux.