COURS

Graphes — Structures et algorithmes de parcours

Tle

Programme de terminale générale, enseignement de spécialité — BO spécial n°8 du 25 juillet 2019

Deux parcours de graphe visitent exactement les mêmes sommets, et un seul donne au passage le plus court chemin. La différence tient à une structure : une file plutôt qu'une pile.

1 Représenter un graphe

DÉFINITION

Un **graphe** est un ensemble de **sommets** reliés par des **arêtes**. Il est **non orienté** si la relation est réciproque, **orienté** si elle ne l'est pas — on parle alors d'**arcs**. Un graphe est **pondéré** si ses arêtes portent un poids. Deux notions à connaître : la **connexité** — peut-on aller de tout sommet à tout autre ? — et le **cycle** — existe-t-il un chemin qui revient à son point de départ ?

PROPRIÉTÉ

Deux représentations coexistent, et le choix dépend de la **densité**. La **matrice d'adjacence** est un tableau à deux dimensions où la case (i, j) indique s'il existe une arête : le test d'adjacence est en O(1), mais l'espace occupé est en n² **même si le graphe a peu d'arêtes**. La **liste d'adjacence** associe à chaque sommet la liste de ses voisins : l'espace est proportionnel au nombre d'arêtes, et le test d'adjacence coûte davantage.

La règle pratique tient en une phrase : un graphe **dense** — beaucoup d'arêtes par rapport au nombre de sommets — se représente bien par une **matrice d'adjacence** ; un graphe **creux** par une liste d'adjacence. Un réseau social, où chacun connaît une poignée de personnes sur des millions, est extrêmement creux : une matrice y gaspillerait un espace considérable pour stocker presque uniquement des zéros.

2 Les deux parcours

RÈGLE

Les deux **parcours** visitent tous les sommets atteignables et ne diffèrent **que** par la structure qui mémorise les sommets à traiter. Le **BFS** — parcours en **largeur** — emploie une **file** : il explore les voisins immédiats, puis leurs voisins, couche par couche. Le **DFS** — parcours en **profondeur** — emploie une **pile**, ou la récursivité : il s'enfonce aussi loin que possible avant de revenir en arrière. Dans les deux cas, il faut **marquer** les sommets visités, faute de quoi un **cycle** provoque une boucle infinie.

$$BFS : file, couche par couche \cdot DFS : pile, en profondeur d'abord$$

LE PIÈGE QUI COÛTE LE PLUS DE POINTS

Utiliser un parcours en profondeur pour déterminer la plus courte distance entre deux sommets. — Le **DFS** trouve **un** chemin, sans aucune garantie sur sa longueur : il s'engage dans la première direction disponible et peut faire un immense détour avant d'atteindre un sommet voisin du départ. Seul le **BFS** explore **couche par couche**, si bien que le premier moment où il atteint un sommet correspond nécessairement au **plus petit nombre d'arêtes**. Le programme ne signale rien : il renvoie un chemin, simplement pas le plus court.

Le réflexe : Plus court chemin en nombre d'arêtes : **BFS**, donc une **file**.

Cette garantie du **BFS** vaut pour le nombre d'**arêtes**, c'est-à-dire pour un graphe **non pondéré**, ou dont toutes les arêtes coûtent la même chose. Dès que les arêtes portent des **poids** différents, le chemin le plus court en nombre d'arêtes n'est plus forcément le moins coûteux : il faut alors **Dijkstra**.

3 Dijkstra

PROPRIÉTÉ

L'algorithme de **Dijkstra** calcule le plus court chemin dans un graphe **pondéré** à poids **positifs**. Il maintient pour chaque sommet la meilleure distance connue, traite à chaque étape le sommet **non traité le plus proche**, et met à jour ses voisins. C'est une généralisation du **BFS** : si toutes les arêtes valent 1, les deux algorithmes traitent les sommets dans le même ordre.

La condition de **positivité** n'est pas une formalité. **Dijkstra fige** la distance d'un sommet dès qu'il le traite, en supposant qu'aucun chemin ultérieur ne fera mieux. Avec un poids **négatif**, un détour peut réduire la distance après coup : l'hypothèse tombe, et l'algorithme renvoie un résultat faux sans le signaler.

PROGRAMMER UN PARCOURS DE GRAPHE

1. Choisir la **représentation** selon la densité : **matrice d'adjacence** si dense, liste sinon.
2. Déterminer l'objectif : visiter tous les sommets, tester la **connexité**, ou trouver le plus court chemin ?
3. Choisir la structure en conséquence : **file** pour le **BFS**, **pile** ou récursion pour le **DFS**.
   Le plus court chemin en nombre d'arêtes impose le BFS.
4. Prévoir un ensemble de sommets **visités** et le consulter avant tout empilement — sinon un **cycle** boucle.
5. Pour reconstituer un chemin, mémoriser le **prédécesseur** de chaque sommet au moment où on l'atteint.

À VÉRIFIER

- Ai-je choisi la représentation selon la **densité** du graphe ?
- Ai-je **marqué** les sommets visités avant de les empiler ?
- Pour un **plus court chemin**, ai-je bien employé une **file** — donc le **BFS** ?
- Mon graphe est-il **pondéré** ? Alors **Dijkstra**, pas **BFS**.
- Mes poids sont-ils **positifs** avant d'appliquer **Dijkstra** ?
- Ai-je mémorisé les **prédécesseurs** si je dois reconstituer le chemin ?

À RETENIR

- **Graphe** : sommets et arêtes ; **orienté** ou non, **pondéré** ou non.
- **Connexité** : va-t-on de tout sommet à tout autre ? **Cycle** : retour au départ.
- **Matrice d'adjacence** : test O(1), espace n^2. Liste : espace proportionnel aux arêtes.
- Graphe **dense** → matrice ; graphe **creux** → liste d'adjacence.
- **BFS** : une **file**, exploration **couche par couche**.
- **DFS** : une **pile** ou la récursion, en profondeur d'abord.
- Seul le **BFS** donne le plus court chemin **en nombre d'arêtes**.
- Cette garantie ne vaut que sur un graphe **non pondéré**.
- Graphe **pondéré** à poids **positifs** : **Dijkstra**.
- Toujours **marquer** les sommets visités, sinon un cycle boucle.
- Mémoriser les **prédécesseurs** pour reconstituer un chemin.