Travaux Pratiques

Exploration de Graphes

Informatique MP2I

Année académique 2024-2025, Clément Rouvroy

Semestre 2

5 et 12 mai 2025

Table des matières

1 Introduction et objectifs

Les graphes sont des structures fondamentales en informatique, permettant de modéliser des relations entre objets. On les retrouve dans de nombreux domaines : réseaux sociaux, réseaux routiers, planification de tâches, bioinformatique, etc. Ce TP explore les méthodes de parcours de graphes et la recherche de plus courts chemins, des algorithmes essentiels pour résoudre de nombreux problèmes pratiques.

Nous utiliserons OCaml pour implémenter les structures de données et les algorithmes. Une bonne maîtrise des structures inductives (listes, arbres) et de la récursivité est utile. L'objectif est de comprendre et d'implémenter les algorithmes de parcours en largeur (BFS), parcours en profondeur (DFS) et l'algorithme de Dijkstra pour les plus courts chemins pondérés.

Consignes :

— Lisez attentivement les questions.

— Soignez la clarté et la justification de vos réponses et de votre code.

— Commentez votre code OCaml pour expliquer les parties importantes.

— Testez vos fonctions avec des exemples pertinents.

2 Préliminaires OCaml

2.1 Listes et Files

Les parcours de graphes utilisent souvent des structures auxiliaires comme les piles (implicitement via la récursion pour DFS) ou les files (explicitement pour BFS).

Question 1. Écrire une fonction OCaml `appartient: 'a -> 'a list -> bool` qui vérifie si un élément appartient à une liste.

Question 2. Rappeler le principe de fonctionnement d'une file (Queue) et les opérations de base (ajouter, retirer, est_vide). Comment peut-on implémenter une file simple en OCaml en utilisant deux listes ? (Indication : une liste pour l'entrée, une pour la sortie).

3 Représentation des Graphes

Un graphe $G = (S, A)$ est défini par un ensemble de sommets S et un ensemble d'arcs (ou arêtes) A. Les sommets sont souvent représentés par des entiers. Un graphe peut être orienté ou non orienté, pondéré ou non pondéré.

Ici, nous considérerons principalement des graphes orientés et potentiellement pondérés. Les sommets seront numérotés de 0 à $n-1$, où n est le nombre total de sommets.

3.1 Matrice d'Adjacence

Une représentation possible est la matrice d'adjacence : une matrice M de taille $n \times n$ où $M[i][j]$ contient une information sur l'arc (i, j).

— Pour un graphe non pondéré, $M[i][j] = 1$ s'il existe un arc de i vers j, et 0 sinon.

— Pour un graphe pondéré, $M[i][j] = w$ si l'arc (i, j) existe et a pour poids w. On utilise souvent une valeur spéciale (ex : ∞ ou -1) si l'arc n'existe pas.

Question 3. Quel est l'avantage et l'inconvénient principal de la matrice d'adjacence en termes de complexité mémoire et de complexité pour vérifier l'existence d'un arc ?

3.2 Listes d'Adjacence

Une autre représentation courante est la liste d'adjacence : un tableau (ou une liste) où l'indice i contient la liste des voisins du sommet i.

— Pour un graphe non pondéré, l'entrée i contient la liste des sommets j tels que (i, j) est un arc.

— Pour un graphe pondéré, l'entrée i contient une liste de paires (j, w) où j est un voisin et w le poids de l'arc (i, j).

Question 4. Définir un type OCaml `graph_list` pour représenter un graphe orienté pondéré (poids de type `float`) en utilisant des listes d'adjacence (un tableau de listes). Écrire une fonction `voisins: graph_list -> int -> (int * float) list` qui renvoie la liste des voisins (avec poids) d'un sommet donné.

Question 5. Quel est l'avantage et l'inconvénient principal des listes d'adjacence par rapport à la matrice d'adjacence, notamment pour les graphes peu denses (avec peu d'arcs) ?

4 Parcours de Graphes

Le parcours de graphe consiste à visiter tous les sommets accessibles depuis un sommet de départ.

4.1 Parcours en Largeur (BFS - Breadth-First Search)

Le BFS explore le graphe niveau par niveau à partir d'un sommet source. Il utilise une file pour mémoriser les sommets à visiter.

Pour utiliser le module `Queue` en OCaml, vous pouvez utiliser :

— `open Queue` au début de votre fichier pour y avoir accès.

— `let file = Queue.create () in` pour créer une file.

— `Queue.add: 'a -> 'a Queue -> unit` pour ajouter un élément à une file.

— `Queue.is_empty : 'a Queue -> bool` pour vérfier si une file est vide ou non.

— `Queue.take : 'a Queue -> 'a` pour défiler.

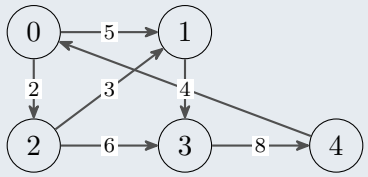
Question 6. Écrire une fonction OCaml `bfs: graph_list -> int -> unit` qui effectue un parcours BFS à partir d'un sommet `depart` et affiche l'ordre de visite des sommets. Vous utiliserez une file (par exemple, le module `Queue` de la bibliothèque standard OCaml ou l'implémentation par deux listes) et un tableau de booléens pour marquer les sommets visités.

4.2 Parcours en Profondeur (DFS - Depth-First Search)

Le DFS explore une branche du graphe aussi loin que possible avant de revenir sur ses pas (backtracking). Il peut être implémenté récursivement (utilisant implicitement la pile d'appels) ou itérativement avec une pile explicite.

Question 7. Écrire une fonction OCaml récursive `dfs_rec: graph_list -> int -> unit` qui effectue un parcours DFS à partir d'un sommet `depart` et affiche l'ordre de visite. Utilisez un tableau de booléens pour marquer les sommets visités afin d'éviter les cycles infinis.

Question 8. Appliquer BFS et DFS (en partant du sommet 0) sur le graphe suivant et donner l'ordre de visite des sommets pour chaque parcours.



5 Plus Courts Chemins

Un problème classique est de trouver le chemin le plus court entre deux sommets. La notion de "court" dépend si le graphe est pondéré ou non.

5.1 Cas non pondéré

Dans un graphe non pondéré, le plus court chemin est celui qui a le moins d'arcs.

Question 9. Expliquer pourquoi le parcours BFS trouve naturellement le plus court chemin en termes de nombre d'arcs depuis un sommet source vers tous les autres sommets accessibles. Modifier la fonction `bfs` pour qu'elle renvoie un tableau des distances (nombre d'arcs) depuis le sommet de départ. Initialiser les distances à ∞ (ou une grande valeur / -1) et la distance du départ à 0.

5.2 Cas pondéré : Algorithme de Dijkstra

Pour trouver le plus court chemin dans un graphe pondéré **avec des poids positifs ou nuls**, l'algorithme de Dijkstra est une référence. Il maintient une estimation de la distance la plus courte depuis la source vers chaque sommet et améliore ces estimations itérativement. Il utilise une file de priorité pour choisir le prochain sommet à traiter (celui avec la distance estimée la plus faible parmi les non-finalisés).

Principe simplifié de Dijkstra :

1. Initialiser les distances : $dist(source) = 0$, $dist(s) = \infty$ pour les autres sommets s.
2. Maintenir un ensemble V de sommets dont la distance minimale est finalisée (initialement vide).

3. Tant qu'il reste des sommets non finalisés accessibles :
   (a) Choisir le sommet u non finalisé avec la plus petite distance estimée $dist(u)$.
   (b) Ajouter u à V (sa distance est maintenant finale).
   (c) Pour chaque voisin v de u : Si $dist(u) + poids(u, v) < dist(v)$, mettre à jour $dist(v) = dist(u) + poids(u, v)$. (Relaxation de l'arc (u, v)).

Pour une implémentation simple sans file de priorité optimisée, on peut rechercher le minimum à chaque étape dans un tableau de distances, ce qui est moins efficace ($O(n^2)$ avec matrice, $O(n^2)$ ou $O(n \log n + m)$ avec listes d'adjacence selon l'implémentation de la recherche du min, vs $O(m + n \log n)$ avec une bonne file de priorité).

Question 10. Implémenter l'algorithme de Dijkstra en OCaml. La fonction `dijkstra: graph_list -> int -> float array` prendra le graphe et le sommet de départ, et renverra un tableau des distances les plus courtes depuis le départ. Utiliser `Float.infinity` pour l'infini et un tableau de booléens pour marquer les sommets finalisés. Pour sélectionner le sommet de plus petite distance non finalisé, vous pouvez parcourir le tableau des distances.

Question 11. Appliquer l'algorithme de Dijkstra (en partant du sommet 0) sur le graphe de la question 9. Donner le tableau final des distances.

Question 12. Pourquoi l'algorithme de Dijkstra ne fonctionne-t-il pas correctement si le graphe contient des arcs de poids négatif ? Donner un exemple simple de graphe où Dijkstra échoue.

6 Pour aller plus loin (Optionnel)

Question 13. Comment modifier l'algorithme de Dijkstra pour reconstruire non seulement la distance mais aussi le chemin le plus court lui-même ?

Question 14. Citer un algorithme permettant de calculer les plus courts chemins entre toutes les paires de sommets. Quelle est sa complexité typique ?