Leçon de CAPES

Résolution de problèmes à l'aide de graphes

Table des matières

1 Introduction **3**

2 Prérequis et objectifs **3**
- 2.1 Prérequis . . . 3
- 2.2 Objectifs de la leçon . . . 3
- 2.3 Compétences mathématiques et informatiques . . . 4

3 Modéliser un problème par un graphe **4**
- 3.1 Définitions fondamentales . . . 4
- 3.2 Exemple de modélisation . . . 4
- 3.3 Choix de la modélisation . . . 4

4 Parcours et connexité **5**
- 4.1 Chaînes, chemins et cycles . . . 5
- 4.2 Recherche d'un itinéraire . . . 5
- 4.3 Parcours en largeur . . . 5
- 4.4 Parcours en profondeur . . . 5
- 4.5 Application : existence d'un itinéraire . . . 6

5 Problèmes de plus court chemin **6**
- 5.1 Définition . . . 6
- 5.2 Algorithme de Dijkstra . . . 6
- 5.3 Exemple numérique . . . 6

6 Arbres et problèmes de connexion **7**
- 6.1 Définition d'un arbre . . . 7
- 6.2 Applications . . . 7
- 6.3 Arbre couvrant de poids minimal . . . 7

7 Coloration des graphes **7**
- 7.1 Définition . . . 7
- 7.2 Problème d'emploi du temps . . . 8
- 7.3 Difficulté algorithmique . . . 8

8 Graphes eulériens et hamiltoniens **8**
- 8.1 Parcours eulériens . . . 8
- 8.2 Parcours hamiltoniens . . . 8

9 Mise en oeuvre algorithmique **8**
9.1 Représentation d'un graphe 8
9.2 Exemple en Python : parcours en largeur 9
9.3 Complexité 9

10 Exploitation pédagogique **10**
10.1 Au collège 10
10.2 Au lycée 10
10.3 Exemple d'activité 10
10.4 Compétences mobilisées 10

11 Conclusion **11**

1 Introduction

De nombreux problèmes consistent à étudier des relations entre des objets : déplacements entre des villes, communications entre des ordinateurs, relations entre des personnes, organisation de tâches ou encore recherche d'un itinéraire optimal.

Dans ces situations, les objets sont représentés par des sommets et les relations entre eux par des arêtes ou des arcs. Cette modélisation conduit à la notion de graphe.

La théorie des graphes permet ainsi de transformer certains problèmes concrets en problèmes mathématiques, puis de les résoudre à l'aide d'outils combinatoires et algorithmiques.

L'objectif de cette leçon est de présenter plusieurs méthodes de résolution de problèmes utilisant les graphes, en mettant en évidence les notions mathématiques mobilisées et leurs applications.

Problématique : comment modéliser un problème à l'aide d'un graphe et choisir un algorithme adapté pour le résoudre ?

Nous étudierons successivement la modélisation par les graphes, les problèmes de parcours et de connexité, les problèmes de plus court chemin et quelques applications classiques.

2 Prérequis et objectifs

2.1 Prérequis

La résolution de problèmes à l'aide de graphes mobilise plusieurs connaissances mathématiques et informatiques préalables :

— la lecture et l'interprétation d'une représentation graphique ;

— le raisonnement logique et la résolution de problèmes ;

— les notions élémentaires de dénombrement ;

— la manipulation d'ensembles et de relations ;

— la lecture et l'écriture d'algorithmes simples ;

— selon les problèmes étudiés, les notions de distance, de coût et d'optimisation.

Ces prérequis peuvent être adaptés au niveau de classe considéré. La théorie des graphes permet également d'introduire certaines de ces notions à partir de situations concrètes, sans exiger au départ une formalisation complète.

2.2 Objectifs de la leçon

À l'issue de cette étude, on cherche à être capable de :

— modéliser une situation concrète par un graphe orienté ou non orienté, éventuellement pondéré ;

— identifier les sommets, les arêtes, les arcs, les degrés, les chaînes et les chemins ;

— déterminer si deux sommets sont reliés et rechercher un itinéraire dans un graphe ;

— utiliser un algorithme de parcours pour explorer un graphe ;

— résoudre des problèmes de plus court chemin et de connexion à l'aide d'algorithmes adaptés ;

— reconnaître les situations relevant de la coloration, des parcours eulériens ou des parcours hamiltoniens ;

— analyser la correction et la complexité d'un algorithme ;

— interpréter la solution obtenue dans le contexte du problème initial.

2.3 Compétences mathématiques et informatiques

Cette leçon fait intervenir plusieurs compétences :

— **Modéliser :** traduire une situation en un graphe adapté ;
— **Chercher :** explorer les solutions possibles et choisir une stratégie ;
— **Raisonner :** justifier les propriétés utilisées et la correction des méthodes ;
— **Calculer :** déterminer des distances, des degrés ou des coûts ;
— **Communiquer :** présenter une démarche et interpréter un résultat ;
— **Programmer :** implémenter ou utiliser un algorithme lorsque cela est pertinent.

3 Modéliser un problème par un graphe

3.1 Définitions fondamentales

Définition 1. *Un graphe non orienté est un couple $G = (V, E)$ où V est un ensemble de sommets et E un ensemble d'arêtes, chaque arête reliant deux sommets.*

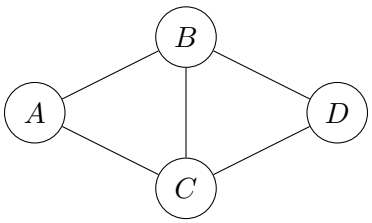
Définition 2. *Un graphe orienté est un couple $G = (V, A)$ où V est un ensemble de sommets et A un ensemble d'arcs. Un arc est un couple ordonné (u, v) : il représente une relation de u vers v.*

Définition 3. *Deux sommets sont adjacents s'ils sont reliés par une arête. Le degré d'un sommet est le nombre d'arêtes qui lui sont incidentes. Dans un graphe orienté, on distingue le degré entrant et le degré sortant.*

Définition 4. *Un graphe est pondéré lorsque chaque arête ou chaque arc est associé à un poids, par exemple une distance, une durée ou un coût.*

3.2 Exemple de modélisation

Considérons quatre villes A, B, C et D, reliées par des routes. On associe un sommet à chaque ville et une arête à chaque route.



Le graphe permet de répondre à plusieurs questions :

— Existe-t-il un itinéraire entre deux villes ?
— Quel est le nombre minimal de routes à emprunter ?
— Quelle est la distance minimale entre deux villes, si les routes sont pondérées ?

3.3 Choix de la modélisation

Le choix du graphe dépend du problème :

— un graphe non orienté modélise une relation réciproque ;
— un graphe orienté modélise une relation à sens unique ;
— un graphe pondéré représente des distances, des durées ou des coûts ;
— un graphe simple ne comporte ni boucle ni arêtes multiples.

Une modélisation pertinente doit conserver les informations nécessaires à la résolution du problème, sans introduire de complexité inutile.

4 Parcours et connexité

4.1 Chaînes, chemins et cycles

Définition 5. *Dans un graphe non orienté, une chaîne est une succession de sommets tels que deux sommets consécutifs soient reliés par une arête. Sa longueur est le nombre d'arêtes parcourues.*

Dans un graphe orienté, on parle généralement de chemin lorsque chaque arc est parcouru dans son sens.

Un cycle est une chaîne fermée dont les sommets, à l'exception du premier et du dernier, sont distincts. Des conventions légèrement différentes existent selon les ouvrages.

Définition 6. *Un graphe non orienté est connexe si toute paire de sommets peut être reliée par une chaîne.*

4.2 Recherche d'un itinéraire

Pour savoir s'il existe un itinéraire entre deux sommets, on peut explorer le graphe en partant du sommet initial et en visitant tous les sommets accessibles.

Deux algorithmes classiques sont :

— le parcours en largeur, qui explore les sommets par niveaux ;

— le parcours en profondeur, qui explore une branche aussi loin que possible avant de revenir en arrière.

4.3 Parcours en largeur

Le parcours en largeur, ou BFS (*Breadth-First Search*), utilise une file d'attente. Il permet notamment de déterminer les distances minimales en nombre d'arêtes dans un graphe non pondéré.

1. Placer le sommet de départ dans une file.
2. Retirer le premier sommet de la file.
3. Ajouter à la file ses voisins qui n'ont pas encore été découverts.
4. Répéter jusqu'à ce que la file soit vide ou que la destination soit atteinte.

Propriété 1. *Dans un graphe non pondéré, le parcours en largeur permet de calculer la distance minimale en nombre d'arêtes entre le sommet de départ et chacun des sommets accessibles.*

Remarque 1. *Pour un graphe représenté par des listes d'adjacence, le parcours en largeur a une complexité en temps de l'ordre de* $O(|V| + |E|)$.

4.4 Parcours en profondeur

Le parcours en profondeur, ou DFS (*Depth-First Search*), explore un sommet puis un de ses voisins non encore visité, et poursuit l'exploration récursivement ou à l'aide d'une pile.

Il permet notamment :

— de déterminer les sommets accessibles ;

— de tester la connexité d'un graphe ;

— de détecter certains cycles ;

— de construire des composantes connexes.

4.5 Application : existence d'un itinéraire

Dans le graphe des quatre villes présenté précédemment, il existe un itinéraire entre A et D, par exemple $A - B - D$ ou $A - C - D$.

Le parcours en largeur donne une distance minimale de deux arêtes entre A et D.

Cet exemple illustre la distinction entre la simple existence d'un itinéraire et la recherche d'un itinéraire optimal.

5 Problèmes de plus court chemin

5.1 Définition

Dans un graphe pondéré, le poids d'un chemin est la somme des poids des arêtes ou des arcs qui le composent.

Le problème du plus court chemin consiste à trouver, entre deux sommets donnés, un chemin dont le poids total est minimal.

5.2 Algorithme de Dijkstra

L'algorithme de Dijkstra résout le problème du plus court chemin depuis une source dans un graphe dont tous les poids sont positifs ou nuls.

Principe :

1. Attribuer la distance 0 au sommet de départ et la distance $+\infty$ aux autres sommets.
2. Choisir, parmi les sommets non définitivement traités, celui dont la distance provisoire est minimale.
3. Mettre à jour les distances de ses voisins : pour un arc (u, v) de poids $w(u, v)$, remplacer la distance de v par
$$d(v) \leftarrow \min\big(d(v), d(u) + w(u, v)\big).$$
4. Répéter jusqu'à ce que tous les sommets accessibles aient été traités ou que la destination soit définitivement traitée.

Théorème 1 (Algorithme de Dijkstra)**.** *Si tous les poids des arêtes sont positifs ou nuls, l'algorithme de Dijkstra calcule les distances minimales depuis le sommet de départ.*

Remarque 2. *L'algorithme de Dijkstra ne garantit pas la correction du résultat en présence de poids négatifs. Pour certains graphes comportant des arcs de poids négatifs, on peut utiliser l'algorithme de Bellman-Ford, qui permet également de détecter les cycles de poids négatif accessibles depuis la source.*

5.3 Exemple numérique

Considérons un réseau routier comportant les arêtes pondérées suivantes :

$$AB = 4, \quad AC = 2, \quad BC = 1, \quad BD = 5, \quad CD = 8.$$

On cherche un plus court chemin de A à D.

Le chemin $A - B - D$ a pour longueur

$$4 + 5 = 9.$$

Le chemin $A - C - B - D$ a pour longueur

$$2 + 1 + 5 = 8.$$

Le chemin $A - C - D$ a pour longueur

$$2 + 8 = 10.$$

Le plus court chemin est donc $A - C - B - D$, de longueur 8.

Cet exemple montre qu'un chemin comportant davantage d'arêtes peut être plus court qu'un chemin plus direct.

6 Arbres et problèmes de connexion

6.1 Définition d'un arbre

Définition 7. *Un arbre est un graphe non orienté, connexe et sans cycle.*

Théorème 2. *Pour un graphe non orienté fini comportant n sommets, les propriétés suivantes sont équivalentes :*

1. *le graphe est un arbre ;*
2. *le graphe est connexe et possède $n - 1$ arêtes ;*
3. *le graphe est sans cycle et possède $n - 1$ arêtes ;*
4. *deux sommets quelconques sont reliés par une unique chaîne simple.*

6.2 Applications

Les arbres interviennent dans de nombreux problèmes :

— connecter plusieurs sites avec une structure sans cycle ;

— représenter une hiérarchie ;

— construire des structures de données ;

— organiser des réseaux de communication.

6.3 Arbre couvrant de poids minimal

Dans un graphe connexe pondéré, un arbre couvrant est un sous-graphe qui contient tous les sommets et qui est un arbre.

Un arbre couvrant de poids minimal est un arbre couvrant dont la somme des poids des arêtes est minimale.

Deux algorithmes classiques sont :

— l'algorithme de Kruskal, qui examine les arêtes par poids croissant et ajoute celles qui ne créent pas de cycle ;

— l'algorithme de Prim, qui construit progressivement un arbre en ajoutant une arête de poids minimal reliant l'arbre à un sommet extérieur.

Ces algorithmes permettent, par exemple, de relier plusieurs bâtiments en minimisant la longueur totale de câble nécessaire.

7 Coloration des graphes

7.1 Définition

Définition 8. *Une coloration des sommets d'un graphe est une attribution d'une couleur à chaque sommet, de sorte que deux sommets adjacents reçoivent des couleurs différentes.*

Le nombre chromatique d'un graphe G, noté $\chi(G)$, est le plus petit nombre de couleurs nécessaires à une telle coloration.

7.2 Problème d'emploi du temps

Supposons que plusieurs examens doivent être organisés. Deux examens ne peuvent pas avoir lieu simultanément si au moins un étudiant doit passer les deux.

On construit un graphe :

— chaque sommet représente un examen ;

— une arête relie deux examens incompatibles ;

— chaque couleur représente un créneau horaire.

La coloration du graphe fournit alors une organisation sans conflit. Le nombre minimal de créneaux correspond au nombre chromatique du graphe.

7.3 Difficulté algorithmique

Déterminer le nombre chromatique exact d'un graphe quelconque est un problème difficile en général. Une méthode gloutonne consiste à colorer successivement les sommets avec la première couleur disponible, mais elle ne garantit pas toujours une coloration optimale.

Cette application permet de montrer comment un problème d'organisation concret devient un problème combinatoire.

8 Graphes eulériens et hamiltoniens

8.1 Parcours eulériens

Définition 9. *Une chaîne eulérienne est une chaîne qui parcourt chaque arête exactement une fois. Un cycle eulérien est une chaîne eulérienne fermée.*

Théorème 3. *Un graphe non orienté connexe possédant au moins une arête admet un cycle eulérien si et seulement si tous ses sommets sont de degré pair.*

Un graphe connexe possède une chaîne eulérienne ouverte si et seulement si exactement deux sommets sont de degré impair.

Application : déterminer s'il est possible d'effectuer une tournée qui emprunte chaque rue d'un réseau exactement une fois, en autorisant éventuellement un départ et une arrivée différents.

8.2 Parcours hamiltoniens

Définition 10. *Un chemin hamiltonien passe une fois par chaque sommet du graphe. Un cycle hamiltonien passe une fois par chaque sommet puis revient au sommet initial.*

Le problème du voyageur de commerce consiste à rechercher une tournée de coût minimal qui visite chaque ville exactement une fois avant de revenir au point de départ.

Il s'agit d'un problème classique d'optimisation combinatoire. Contrairement au problème eulérien, qui porte sur les arêtes, le problème hamiltonien porte sur les sommets.

La recherche d'une solution optimale est difficile en général, notamment lorsque le nombre de villes devient important.

9 Mise en oeuvre algorithmique

9.1 Représentation d'un graphe

Un graphe peut être représenté par :

— une liste d'adjacence ;

— une matrice d'adjacence ;

— une liste d'arêtes ou d'arcs, éventuellement pondérés.

Pour un graphe comportant n sommets, la matrice d'adjacence occupe $O(n^2)$ emplacements. Une liste d'adjacence occupe un espace de l'ordre de $O(n+m)$, où m désigne le nombre d'arêtes, dans le cas usuel d'un graphe simple non orienté.

Le choix de la représentation dépend notamment de la densité du graphe et des opérations à effectuer.

9.2 Exemple en Python : parcours en largeur

Le code suivant utilise un dictionnaire de listes d'adjacence.

```
from collections import deque

def parcours_largeur(graphe, depart):
    distances = {depart: 0}
    file = deque([depart])

    while file:
        sommet = file.popleft()

        for voisin in graphe[sommet]:
            if voisin not in distances:
                distances[voisin] = distances[sommet] + 1
                file.append(voisin)

    return distances

graphe = {
    "A": ["B", "C"],
    "B": ["A", "C", "D"],
    "C": ["A", "B", "D"],
    "D": ["B", "C"]
}

print(parcours_largeur(graphe, "A"))
```

Le dictionnaire retourné associe à chaque sommet accessible sa distance minimale depuis le sommet de départ, mesurée en nombre d'arêtes.

9.3 Complexité

La complexité algorithmique permet d'évaluer les ressources nécessaires à la résolution d'un problème en fonction de la taille de ses données.

Pour un graphe représenté par des listes d'adjacence :

— le parcours en largeur est en $O(n + m)$;

— le parcours en profondeur est en $O(n + m)$;

— Dijkstra avec une file de priorité adaptée est en $O((n+m)\log n)$ dans une implémentation usuelle.

Ces ordres de grandeur permettent de comparer les méthodes et de comprendre pourquoi certains problèmes deviennent difficiles lorsque le nombre de sommets augmente.

10 Exploitation pédagogique

Les graphes permettent de travailler simultanément la modélisation, le raisonnement, l'algorithmique et la résolution de problèmes.

10.1 Au collège

Les élèves peuvent découvrir les graphes à travers :

— la recherche d'itinéraires dans un réseau ;

— les problèmes de déplacements ;

— les jeux de parcours et les labyrinthes ;

— la représentation de relations entre objets.

L'objectif est d'identifier les sommets et les arêtes, de traduire un énoncé en graphe et de justifier une solution.

10.2 Au lycée

Selon les programmes et les spécialités concernés, on peut étudier :

— les graphes pondérés et les matrices d'adjacence ;

— les algorithmes de parcours ;

— la recherche de plus courts chemins ;

— la programmation et l'analyse de la complexité ;

— les problèmes d'optimisation et leurs limites.

10.3 Exemple d'activité

Une commune souhaite relier plusieurs bâtiments par des câbles. Les longueurs possibles entre bâtiments sont connues.

On demande aux élèves :

1. de construire le graphe associé à la situation ;
2. de distinguer les arêtes possibles des liaisons retenues ;
3. de rechercher un réseau reliant tous les bâtiments ;
4. de comparer plusieurs solutions en calculant leur coût total ;
5. de justifier pourquoi la solution retenue est optimale.

Cette activité conduit naturellement à la notion d'arbre couvrant de poids minimal.

10.4 Compétences mobilisées

La résolution de problèmes à l'aide de graphes mobilise :

— la compétence *chercher*, pour explorer des solutions ;

— la compétence *modéliser*, pour construire le graphe ;

— la compétence *raisonner*, pour justifier les résultats ;

— la compétence *calculer*, pour déterminer des distances ou des coûts ;

— la compétence *communiquer*, pour expliquer la démarche ;

— la compétence *programmer*, lorsque la solution est algorithmique.

Le niveau de formalisation doit être adapté aux connaissances des élèves et aux objectifs de la séance.

11 Conclusion

La théorie des graphes fournit un cadre de modélisation puissant pour résoudre de nombreux problèmes concrets.

Le choix de l'outil dépend de la question posée :

— le parcours en largeur et le parcours en profondeur permettent d'explorer un graphe ;

— Dijkstra permet de rechercher un plus court chemin lorsque les poids sont positifs ou nuls ;

— les arbres couvrants de poids minimal permettent de construire un réseau de coût minimal ;

— la coloration traite des problèmes d'affectation sous contraintes ;

— les propriétés eulériennes et hamiltoniennes éclairent différents problèmes de tournée.

La démarche générale consiste à traduire une situation en graphe, à choisir un algorithme ou un résultat théorique adapté, puis à interpréter la solution obtenue dans le contexte initial.

L'étude des graphes est ainsi un point de rencontre entre mathématiques discrètes, algorithmique et applications concrètes.

Ouverture

On peut prolonger cette étude par les problèmes de flot maximal, d'appariement dans les graphes bipartis ou de recherche opérationnelle. Ces thèmes illustrent la place des graphes dans l'optimisation et dans la conception de réseaux.

Références

[1] R. Diestel, *Graph Theory*, 5e édition, Springer, 2017.

[2] M. Gondran et M. Minoux, *Graphes et algorithmes*, Éditions Eyrolles, Paris, 1995.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest et C. Stein, *Introduction to Algorithms*, 4e édition, MIT Press, 2022.

[4] J. A. Bondy et U. S. R. Murty, *Graph Theory*, Springer, 2008.

[5] K. H. Rosen, *Discrete Mathematics and Its Applications*, 8e édition, McGraw-Hill Education, 2019.

[6] Python Software Foundation, *Documentation officielle de Python*, `https://docs.python.org/3/`, consultée le 20 octobre 2026.

[7] Ministère de l'Éducation nationale, *Éduscol : ressources d'accompagnement des programmes de mathématiques*, `https://eduscol.education.fr/`, à consulter pour les programmes et ressources pédagogiques en vigueur.

[8] Ministère de l'Éducation nationale, *Programmes de mathématiques du lycée général et technologique*, Bulletin officiel de l'Éducation nationale, `https://www.education.gouv.fr/`, à consulter pour les textes réglementaires en vigueur.