Chapitre 3

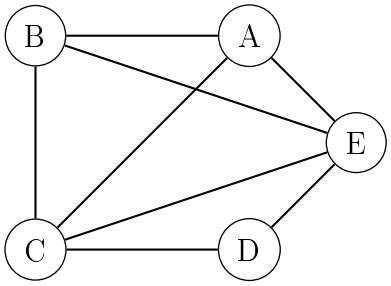
Graphes

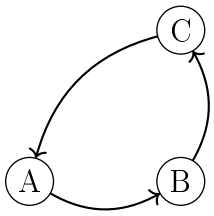
3.1 Graphe

3.1.1 Généralités

Un **graphe** est une structure composée d'**objets** dans laquelle certaines paires d'objets sont en **relation**. Les objets sont appelés sommets (ou nœuds, *vertex / vertices* en anglais), et les relations entre sommets sont appelés des arêtes (ou liens, *edge* en anglais). Un graphe se représente par un diagramme comme dans l'exemple ci-dessous.

Exemples : Le graphe ci-dessous à gauche comporte cinq sommets : A, B, C, D, E. On peut constater que tous les sommets ne sont pas forcément reliés entre eux et ses relations sont symétriques. Le graphe ci-dessous à droite comporte trois sommets tous reliés entre eux ; cependant, les relations ne sont pas symétriques.





Rigoureusement, un graphe G se définit comme un couple $G = (S\,;A)$ où :

— S est l'ensemble des sommets du graphe.

— A est l'ensemble des arêtes – ou relations – entre les différents sommets. Une relation peut s'écrire sous la forme d'un couple de sommets : $\{x\,;y\}$ où x, $y \in S$. Ainsi,

$$A = \{\{x\,;y\} \mid (x\,;y) \in A\}.$$

Définition 3.1.

- *Un graphe est* ***non orienté*** *si et seulement si les relations entre ses sommets sont symétriques : pour tout couple* $(S_1\,;S_2)$ *de sommets, si* S_1 *est en relation avec* S_2*, alors* S_2 *est en relation avec* S_1*.*
- *Un graphe est* ***orienté*** *si et seulement si les relations entre ses sommets ne sont pas nécessairement symétriques : il existe un couple* $(S_1\,;S_2)$ *de sommets tel que* S_1 *est en relation avec* S_2 *mais* S_2 *n'est pas en relation avec* S_1*.*

Remarque : Dans un graphe orienté, les relations sont représentées par des flèches afin de montrer dans quel sens est la relation. On ne parle plus d'arêtes mais d'**arcs**.

Vocabulaire :

- Deux sommets en relations sont dits **voisins** ou **adjacents**.
- Un sommet en relation avec aucun autre est dit **isolé**.
- Si aucun sommet n'est isolé, on dit que le graphe est **connexe**.
- Un sommet peut être en relation avec lui-même, dans ce cas l'arête qui le relie à lui-même constitue une **boucle**.
- Deux arêtes sont dites **parallèles** si et seulement si elles relient deux mêmes sommets.
- Un graphe n'ayant ni boucles ni arêtes parallèles est dit **simple**.
- Un graphe simple est dit **complet** si tous ses sommets sont adjacents deux à deux.

Exemples : le graphe de gauche du premier exemple ci-dessus est simple, il ne comporte ni boucles ni arêtes parallèles. Les deux ci-dessous ne le sont pas ; celui de gauche possède des boucles et celui de droite des arêtes parallèles.

Exemples : le premier graphe donné en exemple n'est évidemment pas complet, B et D ne sont pas en relation par exemple.

Exemples : Les graphes non orientés permettent de modéliser des situations où les relations entre les objets étudiés sont symétriques, par exemples :

- des relations d'amitié ;
- un réseau ferroviaire ;
- un réseau électrique.

Les graphes orientés permettent eux de représenter des situations où ces relations ne sont pas symétriques, par exemples :

- des abonnements sur des réseaux sociaux ;
- un réseau routier (car il existe des rues à sens unique).

Remarques : les arbres sont des graphes particuliers.

3.1.2 Propriétés

Les caractéristiques suivantes sont valables pour tout type de graphe.

Ordre d'un graphe : nombre de sommets d'un graphe.

Degré d'un sommet : nombre d'arêtes auxquelles il est relié.

Distance entre deux sommets : nombre d'arêtes minimal nécessaire pour aller d'un sommet à un autre.

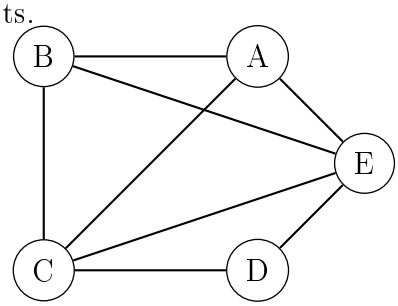
Écartement d'un sommet : distance maximale entre ledit sommet et les autres sommets du graphe.

Centre d'un graphe : sommet d'écartement minimal (pas forcément unique).

Rayon d'un graphe : écartement d'un des centres du graphe.

Diamètre d'un graphe : distance maximale entre deux sommets d'un graphe.

Exemple : Reprenons le graphe ci-dessous. Il est clairement d'ordre 5 puisqu'il possède cinq sommets.



Sommet	Degré
A	3
B	3
C	4
D	2
E	4

Distance de • à •	A	B	C	D	E	Écartement
A	0	1	1	2	1	2
B	1	0	1	2	1	2
C	1	1	0	1	1	1
D	2	2	1	0	1	2
E	1	1	1	1	0	1

Le tableau ci-dessus nous permet d'établir que le graphe admet deux centres : C et E, que son rayon vaut 1 et son diamètre 2.

Remarques :

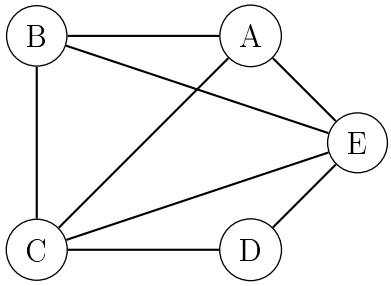
— Le tableau (ou matrice) des distances possède une symétrie axiale par rapport à sa diagonale gauche droite. On parle de matrice symétrique.

— Attention , le diamètre n'est pas forcément égal au double du rayon.

3.1.3 Matrice d'adjacence

Il est possible de représenter un graphe à l'aide d'une **matrice d'adjacence**. Il s'agit d'un tableau recensant le nombre de relations entre deux sommets.

Exemple : Reprenons notre graphe exemple et recensons le notre de relations entre chaque sommet. On obtient la matrice d'adjacence ci-dessous.

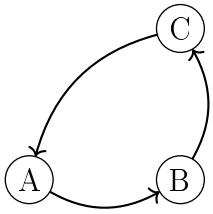


	A	B	C	D	E
A	0	1	1	0	1
B	1	0	1	0	1
C	1	1	0	1	1
D	0	0	1	0	1
E	1	1	1	1	0

La matrice d'adjacence est généralement représentée par une matrice au sens mathématique du terme. Avec l'exemple ci-dessus, on obtient

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Exemples : Reprenons notre exemple de graphe non orienté. Sa matrice d'adjacence, comme on peut le constater, n'est pas symétrique.



$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Propriété 3.1. *Un graphe est non orienté si et seulement si sa matrice d'adjacence est symétrique.*

Propriété 3.2. *Un graphe d'ordre n est simple si et seulement si sa matrice d'adjacence M vérifie les deux conditions suivantes :*

- *tous les éléments de M ont valeurs 0 ou 1 :* $\forall\,(i\,;j) \in [\![1\,;n]\!]^2\,,\ M_{i,j} \in \{0\,;1\}$;
- *les éléments diagonaux sont tous nuls :* $\forall i \in [\![1\,;n]\!]\,,\ M_{i,i} = 0$.

En effet,

- « tous les éléments de la matrice d'adjacence ont valeurs 0 ou 1 » signifie que le graphe ne contient pas d'arêtes parallèles ;
- « les éléments diagonaux sont tous nuls » signifie que le graphe ne contient pas de boucles.

3.1.4 Graphe pondéré

Un **graphe pondéré** est un graphe où chaque arête possède un poids. Ce poids peut représenter :

— un coût, comme dans le protocole de routage OSPF ;

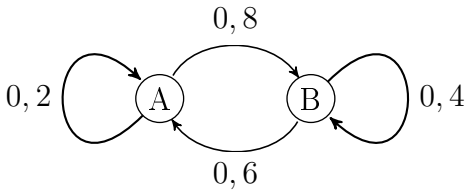
— un temps, comme dans le trajet construit par un GPS ;

— une distance ;

— une probabilité (on parle de graphe probabiliste) ;

— etc.

Exemple : ci-dessous, un graphe orienté pondéré. On constatera par ailleurs qu'il n'est pas simple. Il s'agit même pour être plus précis d'un graphe probabiliste : la somme des poids des arêtes émanant de chaque nœud vaut 1.



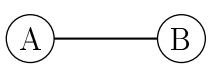
3.2 Parcours de graphes

3.2.1 Parcours en largeur

Le parcours en largeur d'un graphe est très similaire à celui d'un arbre. La principale différence réside dans le fait qu'il n'y a pas de racine par laquelle commencer ; il faut donc choisir un nœud de départ.

Une fois ce nœud de départ choisi, on place dans une file tous les nœuds vers lesquels pointe notre nœud (et qui sont donc à une distance 1 de lui). On fait ensuite défiler la file en réitérant le processus jusqu'à ce que la file soit vide.

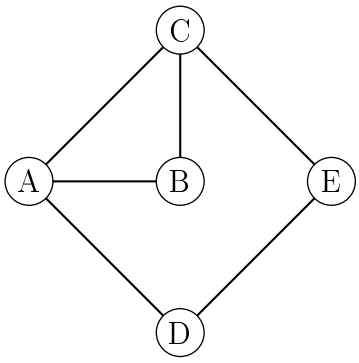
Remarque : avant d'ajouter un nœud à la file, il est essentiel de vérifier que celui-ci n'a pas déjà été parcouru. En effet, on se retrouverait instantanément avec une file se remplissant à l'infini sinon, et ce même avec les graphes les plus simples comme on peut le voir dans l'exemple ci-dessous.



En commençant à partir de A, on doit ajouter B à notre file. Au moment de considérer B, si A n'a pas été noté comme parcouru, on l'ajoute à nouveau à la file. On rentre alors dans une boucle infinie où A et B sont tour à tour ajoutés puis supprimés.

C'est l'autre grande différence avec les arbres dans lesquels on ne peut pas revenir en arrière. Pour éviter cet écueil, il faut marquer les nœuds parcourus à l'aide d'un booléen (on parle aussi de marquage par une couleur).

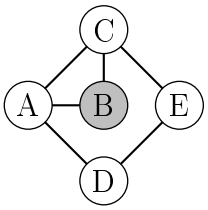
Exemple : Considérons le graphe ci-dessous.



On commence par noter la liste des nœuds couplés à un booléen symbolisant leur parcours ou non :

(A ; 0), (B ; 0), (C ; 0), (D ; 0), (E ; 0).

Il faut à présent choisir un nœud de départ. Commençons par B, son booléen vaut 0, on peut donc l'ajouter à la file et mettre alors son booléen à 1.

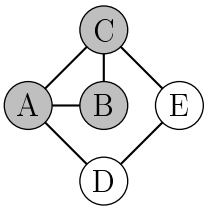


On place B dans la file en mettant son booléen à 1.

Nœuds : (A ; 0), (B ; 1), (C ; 0), (D ; 0), (E ; 0).

File : B

Affichés :

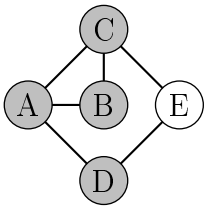


On ajoute A et C qui ont des booléens valant 0 et on les marque alors en les passant à 1.

Nœuds : (A ; 1), (B ; 1), (C ; 1), (D ; 0), (E ; 0).

File : A, C

Affichés : B

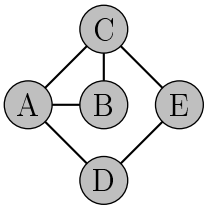


C est relié à A mais son booléen vaut déjà 1, on ne le remet pas dans la file. On peut cependant ajouter D dont le booléen vaut 0 puis changer ce dernier en 1.

Nœuds : (A ; 1), (B ; 1), (C ; 1), (D ; 1), (E ; 0).

File : C, D

Affichés : B, A

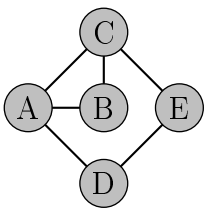


A est relié à C mais son booléen vaut déjà 1, on ne le remet pas dans la file. On peut cependant ajouter E dont le booléen vaut 0 puis changer ce dernier en 1.

Nœuds : (A ; 1), (B ; 1), (C ; 1), (D ; 1), (E ; 1).

File : D, E

Affichés : B, A, C

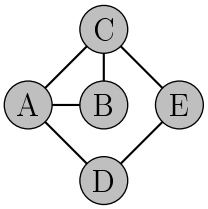


A et E sont reliés à D mais leurs booléens valent déjà 1, on ne les remet pas dans la file.

Nœuds : (A ; 1), (B ; 1), (C ; 1), (D ; 1), (E ; 1).

File : E

Affichés : B, A, C, D



C et D sont reliés à E mais leurs booléens valent déjà 1, on ne les remet pas dans la file. La file est maintenant vide et le parcours prend fin.

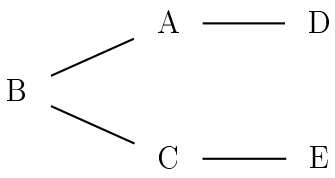
Nœuds : (A ; 1), (B ; 1), (C ; 1), (D ; 1), (E ; 1).

File :

Affichés : B, A, C, D, E

Ce processus revient d'une certaine façon à voir le graphe sous la forme d'un arbre dont la racine serait le premier nœud considéré. Chaque niveau de profondeur de l'arbre correspond à une distance d'éloignement du nœud initial.

Exemple : l'arbre correspondant au parcours en largeur du graphe précédent est



3.2.2 Parcours en profondeur

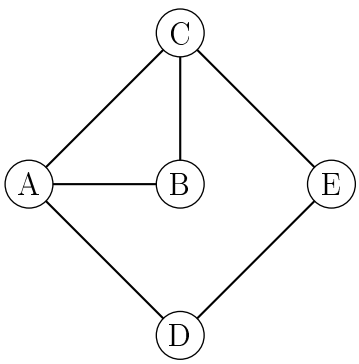
La parcours en profondeur d'un graphe est lui aussi similaire à celui de l'arbre. Toutefois, en plus de la différence du choix du premier nœud, on a aussi le fait qu'il n'existe pas plusieurs types de parcours en profondeur contrairement aux arbres (préfixe, infixe, suffixe). En effet, n'ayant ici ni racine ni enfants à proprement parler, il n'y a pas de choix à faire sur l'ordre de parcours de ceux-ci.

Une fois ce nœud de départ choisi, on le place dans une pile. On a alors deux options.

1. Si le sommet de la pile pointe vers des nœuds qui ne sont pas / plus dans la pile (on retrouve à nouveau la nécessité de marquer les nœuds parcourus), on ajoute un de ces sommets dans la pile en le marquant au préalable.
2. Sinon, on dépile et on affiche.

On réitère ce processus jusqu'à temps que la pile soit vide.

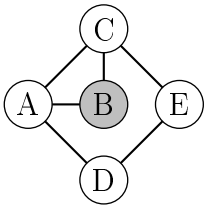
Exemple : Reprenons le graphe ci-dessous.



On commence par noter la liste des nœuds couplés à un booléen symbolisant leur parcours ou non :

(A ; 0), (B ; 0), (C ; 0), (D ; 0), (E ; 0).

Il faut à présent choisir un nœud de départ. Commençons à nouveau par B, son booléen vaut 0, on peut donc l'ajouter à la file et mettre alors son booléen à 1.

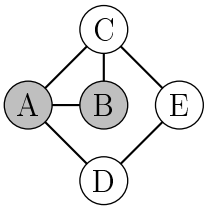


On place B dans la file en mettant son booléen à 1.

Nœuds : (A ; 0), (B ; 1), (C ; 0), (D ; 0), (E ; 0).

Pile : B

Affichés :

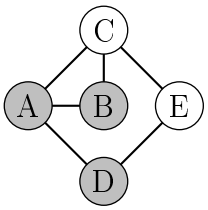


B pointe vers A et C qui ont des booléens valant 0, on choisit A et on le marque alors en le passant à 1.

Nœuds : (A ; 1), (B ; 1), (C ; 0), (D ; 0), (E ; 0).

Pile : B, A

Affichés :

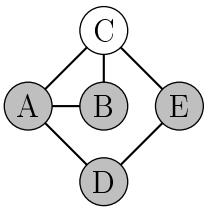


A pointe vers B – dont le booléen vaut 1 et qui ne peut donc être considéré –, C et D qui ont tous les deux des booléens valant 0. On peut ajouter D puis changer son booléen.

Nœuds : (A ; 1), (B ; 1), (C ; 0), (D ; 1), (E ; 0).

Pile : B, A, D

Affichés :

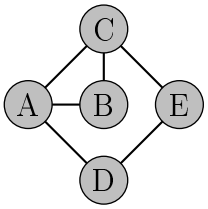


D pointe vers A – dont le booléen vaut 1 et qui ne peut donc être considéré – et E. On ajoute E et change son booléen.

Nœuds : (A ; 1), (B ; 1), (C ; 0), (D ; 1), (E ; 1).

Pile : B, A, D, E

Affichés :

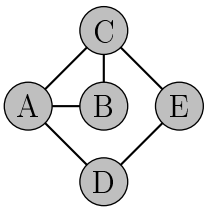


E pointe vers D – dont le booléen vaut 1 et qui ne peut donc être considéré – et C. On ajoute C et change son booléen.

Nœuds : (A ; 1), (B ; 1), (C ; 1), (D ; 1), (E ; 1).

Pile : B, A, D, E, C

Affichés :



On ne peut plus ajouter personne, on commence à dépiler. Et comme à chaque étape du dépilement, on ne peut ajouter de nouveau sommet, on dépile intégralement la pile.

Nœuds : (A ; 1), (B ; 1), (C ; 1), (D ; 1), (E ; 1).

Pile :

Affichés : C, E, D, A, B

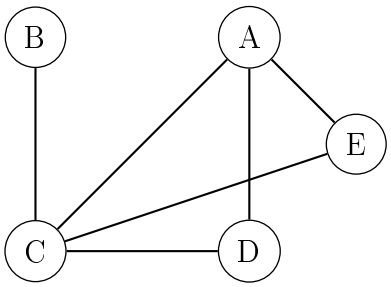
3.3 Cycle

3.3.1 Chaîne et cycle

On appelle **chaîne** toute succession d'arêtes dont l'extrémité de l'une (sauf la dernière) est l'origine de la suivante. Le nombre d'arêtes qui composent une chaîne est appelé longueur de la chaîne.

Une chaîne dont l'origine et l'extrémité coïncident est dite **fermée**. Un **cycle** est une chaîne fermée dont les arêtes sont toutes distinctes. Enfin, un cycle **élémentaire** qui ne passe pas deux fois par un même sommet.

Exemple : Considérons le graphe ci-dessous.



— B - C - D - A - E est une chaîne de longueur 5 ;

— B - C - A - D- C - B est une chaîne fermée mais pas un cycle car elle passe deux fois par l'arête {B ; C}.

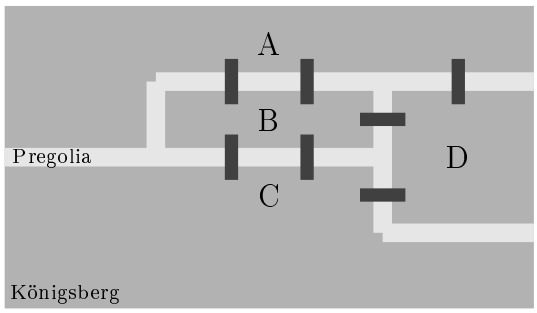
— A - C - D - A et A - E - C - D - A sont des cycles.

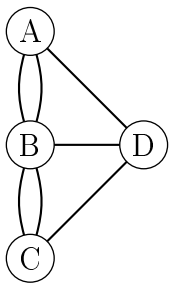
3.3.2 Chaîne et cycle eulériens

Dans un graphe une **chaîne eulérienne** est une chaîne passant une et une seule fois par chaque arête du graphe. On dit alors que le graphe est **eulérien**. Si en plus la chaîne se finit par son sommet de départ, on a alors un cycle : on parle de **cycle eulérien**.

Le nom d'eulérien a été donné en hommage à Leonhard Euler, mathématicien et physicien du XVIII^{e}, en raison de ses importants travaux en théorie des graphes, avec notamment la résolution du problèmes des sept ponts de Königsberg.

Königsberg est au XVIII^{e} une ville Prusse traversée par la rivière Pregolia, laquelle peut être traversée par sept ponts comme indiqués sur le schéma ci-dessous à gauche.





Le problème était de savoir s'il était possible de se promener en traversant chaque pont de la ville une et une seule fois. La ville est découpée en quatre zones par la rivière : A, B, C et

D. Ces zones peuvent être modélisées par les sommets d'un graphe et les ponts par ses arêtes. On obtient alors le graphe ci-dessus à droite.

Le problème était donc de savoir si le graphe ci-dessus était eulérien : passer par tous les ponts une et une seule fois revient chercher un cycle passant une et une seule par toutes les arêtes. L. Euler établit le théorème suivant qui permet de résoudre le problème des sept ponts de Königsberg.

Théorème 3.1.

1. *Un graphe connexe admet une chaîne eulérienne si et seulement si ses sommets sont tous de degré pair sauf au plus deux.*
2. *Un graphe connexe admet un cycle eulérien si et seulement si tous ses sommets sont de degré pair.*

Comme les sommets A et C sont de degrés impairs, on en déduit qu'il n'existe pas de cycle eulérien dans le graphe et le problème des sept ponts de Königsberg n'a donc pas de solution.

Remarque : on peut comprendre intuitivement cette idée de n'avoir que des sommets de degrés pairs. En effet, si on veut accéder à un sommet et en sortir sans emprunter la même arête, il faut qu'il y ait autant d'entrées que de sorties ; le degré du sommet doit donc être égal à deux fois le nombre d'entrées, il est pair.

3.3.3 Chaîne et cycle hamiltonien

Dans un graphe une **chaîne hamiltonienne** est une chaîne élémentaire passant une et une seule fois par chaque sommet du graphe. On dit alors que le graphe est **hamiltonien**. Si en plus la chaîne se finit par son sommet de départ, on a alors un cycle : on parle de **cycle hamiltonien**.

Le nom d'hamiltonien a été donné en hommage à W. R. Hamilton, mathématicien, physicien et astronome du XIX^{e} pour ces travaux sur la recherche de cycles hamiltoniens dans le graphe des arêtes du dodécaèdre.

Si on dispose de conditions nécessaires et suffisantes pour déterminer si un graphe est eulérien, ce n'est pas le cas pour les graphes hamiltoniens. Il existe des critères pour déterminer si un graphe est hamiltonien ou non mais ceux-ci ne permettent pas de détecter tous les graphes hamiltoniens. C'est un problème qui reste encore ouvert.

3.4 Recherche d'un chemin dans un graphe

3.4.1 Recherche de chaînes

La recherche de chaînes et de cycles dans des graphes répond à de nombreuses problématiques, notamment de transport (fourniture d'un trajet routier, ferroviaire, d'une tournée de distribution). Dans ces derniers, on souhaitera en plus optimiser la chaîne en fonction de divers paramètres (coût, temps, distance, etc.

Toutefois, avant d'optimiser le parcours d'un graphe, il faut déjà pouvoir établir l'existence d'un chemin entre le départ et l'arrivée. C'est l'objet de l'algorithme ci-dessous.

Algorithme 1 : Recherche de l'existence d'un chemin entre deux sommets

```
1  Fonction existence_chemin(sommet_départ, sommet_arrivée) :
2
3      file_nœuds_possibles ← enfiler(sommet_départ)
4      Marquer sommet_départ comme présent dans la file
5
6      Tant que file_nœuds_possibles non vide :
7
8          nœud ← défiler(file_nœuds_possibles)
9          Si nœud = sommet_arrivée :
10             Renvoyer : Vrai
11
12         Pour voisin dans voisins(nœud) :
13             Si voisin n'est pas marqué visité :
14                 Ajouter voisin à nœuds_possibles Marquer voisin comme présent dans
                   la file
15     Renvoyer : Faux
```

Remarques :

— Pour rechercher une chaîne fermée passant deux sommets, il suffit alors d'appeler deux fois notre fonction ci-dessus en inversant le rôle du sommet de départ et d'arrivée lors du deuxième appel.

— Cet algorithme est en réalité un parcours en largeur du graphe dans lequel on teste à chaque fois si le nœud qui l'on vient de faire défiler est le nœud recherché.

3.4.2 Recherche du plus court chemin ; l'algorithme de Dijkstra

Dans un graphe pondéré, le plus court chemin entre deux sommets du graphe est celui dont la somme des coûts de toutes les arêtes empruntées est minimale. Il déterminé à l'aide de l'algorithme de Djikstra, du nom de son inventeur, l'informaticien E. W. Djikstra. Il permet notamment de trouver le plus court chemin entre deux routeurs dans le protocole OSPF, le plus trajet le plus rapide entre un départ et une destination dans un réseau routier, ferroviaire, etc.

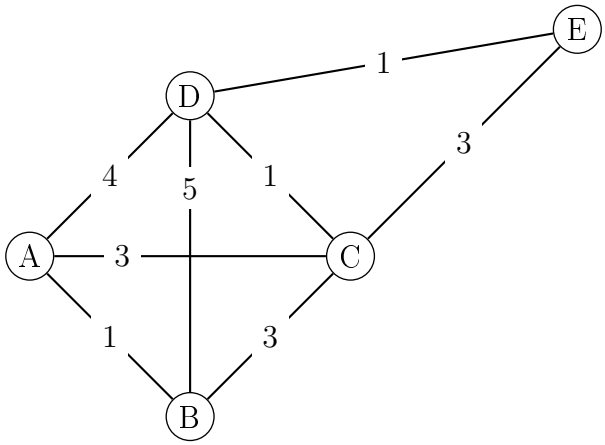
Remarques :

— L'algorithme de Dijkstra est fondé sur celui du parcours en largeur.

— L'algorithme de Dijkstra ne permet pas de déterminer le plus court chemin dans un graphe pondéré comportant des poids négatifs. Ce sont les algorithmes de Bellman-Ford et Floyd-Warshall qui permettent de le faire.

— Pour un graphe à s sommets et a arêtes, l'algorithme de Dijkstra à dans le pire des cas une complexité temporelle en $O(a + s\ln(s))$.

Exemple : Considérons le réseau ci-dessous constitué des routeurs A, B, C, D et E. Avec les coût de chaque chemin exprimé sur l'arête le représentant.



Application de l'algorithme de Djikstra afin de déterminer le plus court chemin entre A et E.

A	B	C	D	E	Choix
0	∞	∞	∞	∞	A
	1(A)	3(A)	4(A)	∞	B
		3(A)	4(A)	∞	C
			4(C)	6(C)	D
				5(D)	E

On trouve A-C-D-E avec un coût de 5.

À chaque étape, on note les nœuds accessibles depuis le nœud en cours ou depuis les nœuds précédemment choisis en notant le coût cumulé d'accès au prochain nœud. Si un nœud est accessible depuis plusieurs nœuds déjà visités à une étape, on garde le chemin ayant le coût le plus faible. Enfin, le nœud disponible ayant le coût le plus faible est choisi pour l'étape suivante. Il s'agit d'un algorithme glouton, en effet, à chaque étape, on fait un choix minimisant un coût local et non global. Cependant, ces choix conduisent à une optimisation du coût global.

3.5 Exercices

3.5.1 Démarrage

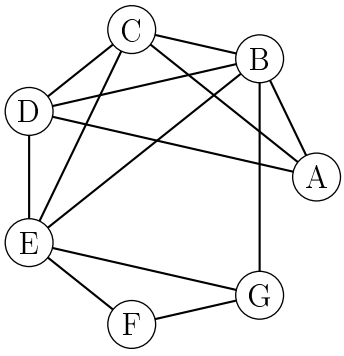
Exercice 3.1. [Graphes complets] Dessiner les graphes complets d'ordre 1 à 5.

Exercice 3.2. Dessiner un graphe dont le diamètre n'est pas égal au double du rayon.

Exercice 3.3.

Déterminer les caractéristiques ci-dessous du graphe ci-contre.

- Ordre du graphe ;
- degré des sommets ;
- écartements des sommets ;
- centre(s) du graphe ;
- rayon et diamètre du graphe.

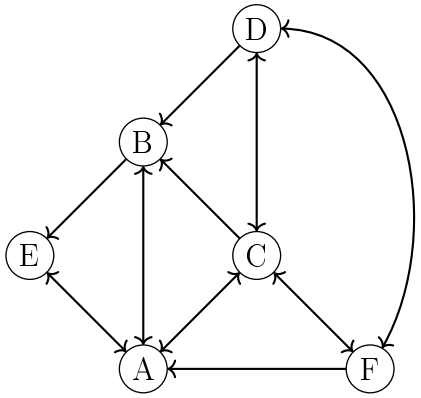


Exercice 3.4. Déterminer la matrice d'adjacence du graphe de l'exercice 3.3.

Exercice 3.5.

Déterminer les caractéristiques ci-dessous du graphe ci-contre.

— Ordre du graphe ;
— degré des sommets ;
— écartements des sommets ;
— centre(s) du graphe ;
— rayon et diamètre du graphe ;
— matrice d'adjacence.



Exercice 3.6. Déterminer dans chaque cas si les graphes correspondant aux matrices d'adjacences ci-dessous sont simples, orientés ou non, puis les construire.

1. $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$

2. $\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$

Exercice 3.7. Donner les parcours en largeur des graphes des exercices 3.3 et 3.5 en partant des nœuds E puis F.

Exercice 3.8. Donner les parcours en profondeur des graphes des exercices 3.3 et 3.5 en partant des nœuds E puis F.

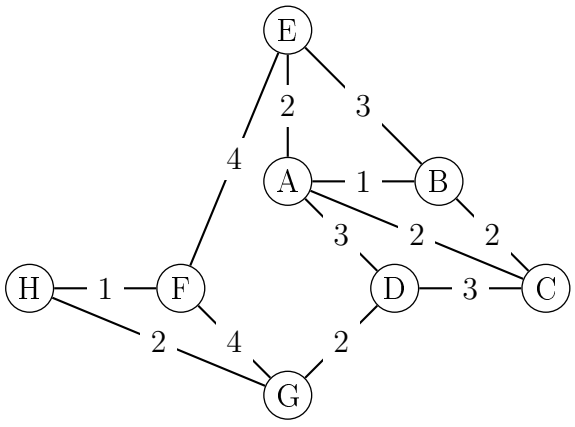
Exercice 3.9.

1. Donner pour le graphe de l'exercice 3.3 un exemple de chaîne, de chaîne fermé (sans être un cycle) et de cycle de longueur 7.
2. Est-il possible de trouver un cycle de longueur 6 dans le graphe de l'exercice 3.5.

Exercice 3.10. Le graphe de l'exercice 3.3 est-il eulérien ? Hamiltonien ?

Exercice 3.11. Exécuter l'algorithme de recherche de l'existence d'un chemin entre les sommets B et F du graphe de l'exercice 3.5.

Exercice 3.12. Déterminer le plus court chemin entre les sommets A et F puis B et H dans le graphe ci-dessous.



3.5.2 Approfondissement

Exercice 3.13. Quel est le nombre d'arêtes d'un graphe non orienté complet d'ordre quelconque n ? En déduire le nombre maximal d'arêtes d'un graphe non orienté d'ordre quelconque n.

Exercice 3.14. [Matrice d'adjacence et relation] Écrire deux fonctions en Python prenants en entrée une matrice d'adjacence et renvoyant en sortie pour chaque sommet :

— la liste des sommets vers lequel il pointe : ses successeurs ;

— la liste des sommets qui pointent vers lui : ses prédécesseurs.

Quel est le cas particulier peut-on avoir ? À quoi ce la correspond-t-il ?

Indication : on pourra écrire la matrice d'adjacence sous la forme d'un tableau ou d'une liste de dictionnaires.

Exercice 3.15. [Matrice d'adjacence et graphe simple] Écrire une fonction en Python prenant en entrée une matrice d'adjacence et déterminant si elle correspond à un graphe simple ou non.

Exercice 3.16. [Matrice d'adjacence et graphe (non) orienté] Écrire une fonction en Python prenant en entrée une matrice d'adjacence et déterminant si elle correspond à un graphe orienté ou non.

Exercice 3.17. [Graphe et POO] Écrire des classes Python `Nœud` et `Graphe` ayant les attributs et méthodes donnés ci-dessous.

Nœud
valeur successeurs prédécesseurs
se présenter

Graphe
nœuds
est vide ajouter / supprimer nœud ajouter / supprimer arête vider graphe rechercher valeur importer matrice d'adjacence exporter matrice d'adjacence est non orienté / est simple se présenter

Les attributs `nœuds`, `successeurs` et `prédécesseurs` seront des listes d'instances de la classe `Nœud`.

La fonction important une matrice d'adjacence importera aussi avec elle une liste des noms des sommets et remplira un graphe vide à partir de ces deux données ; on pourra mettre par défaut la liste des noms à `None` et si c'est le cas créer une liste de valeurs génériques pour les nœuds. On vérifiera aussi au préalable que la matrice importée peut être effectivement une matrice d'adjacence.

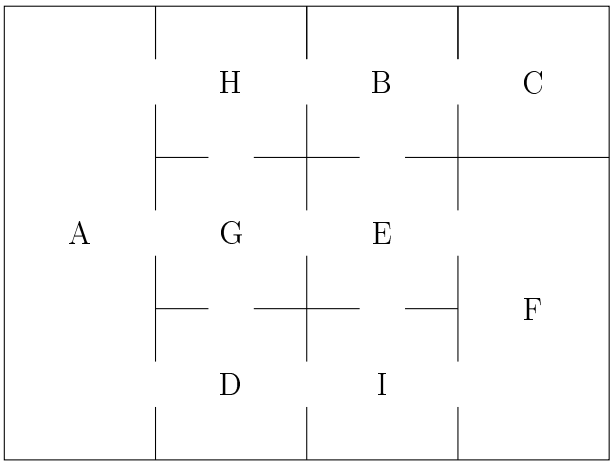
Exercice 3.18. [Pour aller plus loin] Comment pourrait-on modifier les classes `Nœud` et `Graphe` pour prendre en compte en plus des graphes pondérés ?

Exercice 3.19. [Parcours en largeur] Ajouter une méthode de parcours en largeur à la classe `Graphe`. Le booléen permettant de savoir si un nœud a déjà été visité ou non sera ajouté à la classe `Nœud` en tant qu'attribut.

Exercice 3.20. [Parcours en profondeur] Ajouter une méthode de parcours en largeur à la classe `Graphe`.

Exercice 3.21. [Recherche de chaînes] Ajouter à la classe `Graphe` une méthode de recherche de chaînes et de chaînes fermées.

Exercice 3.22. Un musée dispose à l'un de ses étages de salles A, B, C, D, E, F, G, H, I disposées comme dans le dessin ci-dessous. On ne considérera pas le moyen d'accéder à cet étage du musée.



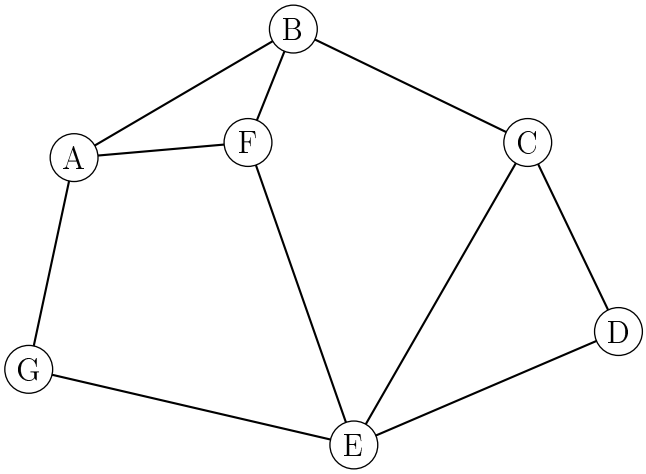
1. Modéliser la situation par un graphe.
2. Est-il possible de trouver un chemin passant une et une seule fois par chaque porte ?
3. Est-il possible de trouver un chemin passant une et une seule fois par chaque salle ?
   (a) Sans nécessairement finir dans la salle par laquelle on a commencé.
   (b) En finissant par la salle on a commencé.

3.5.3 Entraînement

Exercice 3.23.

Déterminer les caractéristiques ci-dessous du graphe ci-contre.

— Ordre du graphe ;
— degré des sommets ;
— écartements des sommets ;
— centre(s) du graphe ;
— rayon et diamètre du graphe.



Exercice 3.24. Déterminer la matrice d'adjacence du graphe de l'exercice 3.23.

Exercice 3.25.

Déterminer les caractéristiques ci-dessous du graphe ci-contre.

— Ordre du graphe ;

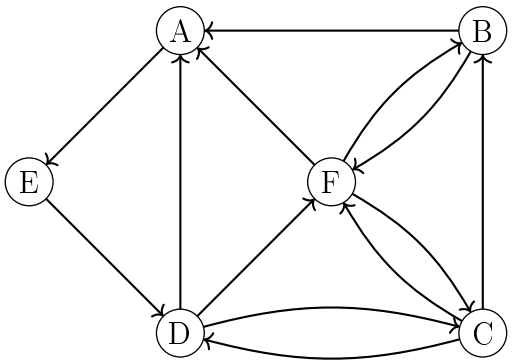
— degré des sommets ;

— écartements des sommets ;

— centre(s) du graphe ;

— rayon et diamètre du graphe ;

— matrice d'adjacence.



Exercice 3.26. Déterminer dans chaque cas si les graphes correspondant aux matrices d'adjacences ci-dessous sont simples, orientés ou non, puis les construire.

1. $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 1 & 1 & 2 & 1 \end{pmatrix}$

2. $\begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$

Exercice 3.27. Donner les parcours en largeur des graphes des exercices 3.23 et 3.25 en partant des nœuds D puis E.

Exercice 3.28. Donner les parcours en profondeur des graphes des exercices 3.23 et 3.25 en partant des nœuds D puis E.

Exercice 3.29.

1. Donner pour le graphe de l'exercice 3.23 un exemple de chaîne, de chaîne fermé (sans être un cycle) et de cycle de longueur 7.
2. Est-il possible de trouver un cycle de longueur 6 dans le graphe de l'exercice 3.25.

Exercice 3.30. Exécuter l'algorithme de recherche de l'existence d'un chemin entre les sommets A et B du graphe de l'exercice 3.25.

Exercice 3.31. Déterminer le plus court chemin entre les sommets C et H puis E et G dans le graphe de l'exercice 3.12.

3.6 Ressources supplémentaires

— Vidéo sur les graphes, généralités 1

— Vidéo sur les graphes, généralités 2

— Vidéo sur les graphes non orientés