INSTITUT INTERNATIONAL POLYTECHNIQUE & COMMERCE 2I

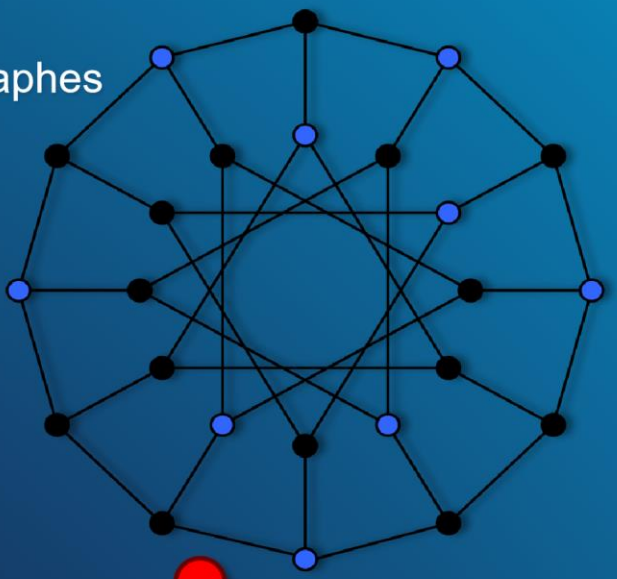
EXPOSÉ

MODULE: Recherche opérationnelle

FILIÈRE: Polytechnique

THÈME: Théorie des graphes

FORMATEUR: MOUKANGALA ALEX



Présenter par : GROUPE III

MBOUMBA-MAPEMBI Monica : GI (DI/GL)

KANISSIDI VIOLETHE Robert Ravisse : GI (DI/GL)

MOUELE TANH : GI (DI/GL)

DIAMONAMESSO CYR : GI (DI/GL)

AOUE DRUCHE Erica : GI (RT)

KAYA-KAYA Romain Dur Van : GI (RT)

DJEMBO TCHIPOUNTOU Grâce Divin : GE (ET)

TCHILOEMBA Tite-Giosias : GE (ET)

BOUYOU Précieux Jérémy : MI

BAZEBIKOUELA GRACIA Francesca Jemima :

MOUKIAMA-BASSEMBA Colombe-Darnella : GPER

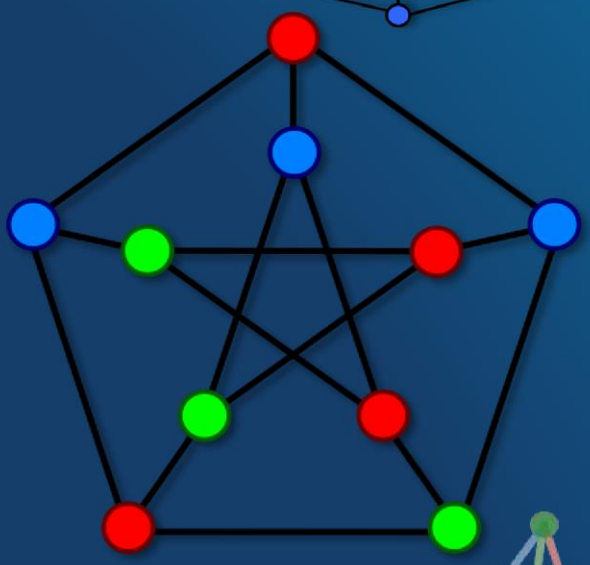
YOLO Princia Juliana : GPIA

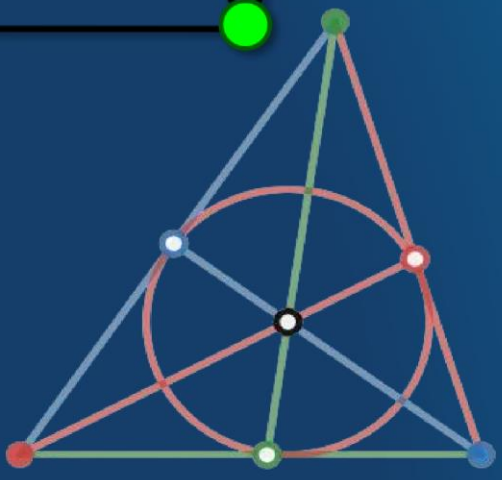
MAVOUNGOU MAGNOUNGOU Berenger : GPIA

MOUAYA Divin Exaucé : GE (AII)

DIANI Ruxelles Schekinah : GE (AII)

OLETA_Johnny Liver : GM (MT)





Année académique: 2025-2026

PLAN DE TRAVAIL

PARTIE I — FONDATIONS THÉORIQUES

- Définition et rôle des graphes en mathématiques et systèmes modernes
- Historique : Euler, évolution, impact en algorithmique
- Terminologie : sommets, arêtes, boucles, multigraphes, graphes simples

PARTIE II — TYPOLOGIE & STRUCTURES

- Graphes orientés vs non orientés
- Graphes pondérés, réguliers, bipartis
- Connexité, planarité, théorème de Kuratowski
- matrice adjacence
- liste d'adjacence

PARTIE III — PROPRIÉTÉS & THÉORÈMES

- Propriétés métriques : degré, distance, diamètre
- Cycles, chemins, composantes • Théorèmes : Euler, Dirac,
- Théorèmes : Kuratowski, ponts
- Conditions de planarité

PARTIE IV — ALGORITHMIQUE

- Parcours BFS et DFS : concepts, pseudo-code, exemples
- Algorithmes d'optimisation : Dijkstra, Bellman-Ford
- Livrables : tableaux comparatifs, démonstrations visuelles
- Algorithmes d'optimisation : Prim, Kruskal,
- Arbres couvrants minimum

PARTIE V — APPLICATIONS & EXERCICES

- Série d'exercices couvrant toute la matière
- Corrections détaillées pour évaluer la compréhension du groupe

Partie I — Fondations théoriques

- **Introduction générale**

La théorie des graphes est une branche des mathématiques qui étudie les graphes, qui sont des structures composées de sommets (ou nœuds) reliés par des arêtes (ou arcs). En recherche opérationnelle, les graphes sont utilisés pour modéliser et résoudre des problèmes complexes.

Concept de graphe : Un graphe est une structure composée d'un ensemble de sommets (ou nœuds) et d'un ensemble d'arêtes (ou arcs) qui relient ces sommets. Exemple : un réseau de transport où les gares sont les sommets et les lignes ferroviaires les arêtes.

- **Place en mathématiques discrètes** : Les graphes appartiennent aux mathématiques discrètes car ils traitent d'objets finis et dénombrables. Ils sont liés à la combinatoire, à la logique et à la théorie des algorithmes.
- **Rôle dans les systèmes modernes** : Les graphes modélisent les réseaux sociaux, Internet, les circuits électroniques, les bases de données relationnelles et les systèmes de transport.

- **Historique & évolution**

- **Euler et le problème des ponts**

  La théorie des graphes commence en 1736 grâce à Leonhard Euler. À Königsberg, sept ponts reliaient quatre parties de la ville. Les habitants voulaient savoir s'il était possible de traverser tous les ponts une seule fois. Euler transforme chaque quartier en point et chaque pont en ligne. Il montre que pour faire un tel trajet, chaque point doit avoir un nombre pair de lignes. À Königsberg, ce n'était pas le cas, donc le parcours est impossible. Cette manière de représenter un problème est à l'origine de la théorie des graphes.

- **Développements essentiels à travers les siècles**

  Au XIXe siècle, Kirchhoff utilise les graphes pour analyser les circuits électriques, et Cayley étudie les arbres utilisés en chimie. Au début du XXe siècle, la théorie se structure et s'enrichit avec l'étude de nouveaux concepts comme les graphes planaires, les graphes bipartis, les cycles, le coloriage et les propriétés de connexité. Ensuite, avec l'arrivée de l'informatique, les graphes prennent une place centrale grâce aux travaux sur les algorithmes, notamment ceux de Dijkstra, Kruskal ou Prim. Ils deviennent essentiels dans les réseaux, les transports, les sciences et l'ingénierie.

- **Influence actuelle en algorithmique et réseaux**

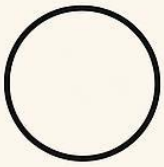
Aujourd'hui, les graphes sont utilisés dans presque tous les domaines. En algorithmique, ils permettent de rechercher les chemins optimaux, d'organiser des tâches, de détecter des cycles et d'optimiser des processus. Dans les réseaux, ils aident à définir les topologies, à faire le routage, à analyser le trafic et à identifier les points critiques d'une infrastructure. Dans la vie quotidienne, ils servent dans les réseaux sociaux, les services de localisation, les systèmes de recommandation, la cybersécurité et même la biologie.

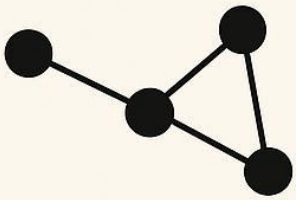
3. Terminologie de base

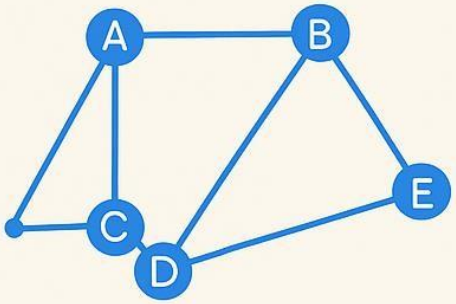
- **Sommet** : élément de base (ville, utilisateur, serveur).
- **Arête** : lien entre deux sommets.
- **Chemin**: suite de sommets reliés par des arêtes.
- **Cycle** : chemin qui revient à son point de départ.
- **Boucle** : arête reliant un sommet à lui-même.
- **Multigraphe** : plusieurs arêtes possibles entre deux sommets.
- **Graphe simple** : pas de boucle ni d'arêtes multiples.

Introduction à la théorie des graphes

Un graphe est une structure composée de sommets (ou nœuds) reliés par des arêtes (ou arcs).

- Sommet
- Arête 
- Chemin 
- Cycle 





Exemple : réseau de transport où les gares sont les sommets et les lignes ferroviaires les arêtes.

Partie II — Typologie & structures

4. Graphes orientés et non orientés

Non orienté :
Un graphe non orienté est un graphe dans lequel les arêtes n'ont pas de direction. Une arête relie simplement deux sommets, et la relation est symétrique.

- **Exemple concret** : une relation d'amitié sur Facebook : si A est ami avec B, alor s B est ami avec A.
- **Utilisation** : utile pour représenter des réseaux où les connexions sont réciproqu es (réseaux électriques, collaborations scientifiques).
- **Importance** : la notion de connexité est centrale : un graphe non orienté est conn exe si chaque sommet est atteignable depuis n'importe quel autre.

- **NB :**

  - Les arêtes n'ont pas de sens.
  - A—B signifie connexion dans les deux directions.
  - Connexe : on peut aller de n'importe quel sommet à n'importe quel autre

Orienté : Un graphe orienté est un type de graphe dans lequel, chaque arête possède une direction.On parle alors d'arc, qui relie un sommet de départ à un sommet d'arrivée.

- **Exemple concret** : un réseau de rues à sens unique, où l'on peut aller de A vers B mais pas forcément de B vers A.
- **Utilisation** : les graphes orientés sont essentiels pour modéliser des flux (circulat ion, transferts d'informations, relations hiérarchiques).
- **Importance** : ils permettent d'étudier des notions comme la forte connexité (poss ibilité de circuler dans les deux sens entre sommets) ou les cycles orientés.
- **NB :**

  - Les arêtes ont un sens (A → B)
  - Faiblement connexe : devient connexe si on ignore les flèches.
  - Fortement connexe : pour chaque A et B, il existe un chemin A → B et un chemin B → A

5. Graphes pondérés, réguliers & bipartis

a. Graphe pondéré : Un graphe pondéré est un graphe dans lequel chaque arête est associée à une valeur numérique appelée poids. Ce poids peut représenter : distance, énergie, coût, durée, charge, etc.

Caractéristiques clés :

- Chaque arête a un nombre associé.
- Les chemins optimaux utilisent les poids.
- La représentation matricielle utilise les valeurs au lieu de 0 et 1.

Exemple : (A) / \ 2 / \ 5 / \ (C)---4---(B) 3 \ (D) Ici :- A–C a un poids de 2- A–B a un poids de 5- C–B a un poids de 4- B–D a un poids de 3

b.Graphe régulier : Un graphe est dit k régulier lorsque tous les sommets ont exactement k arêtes. Le degré de chaque sommet est donc le même.

Caractéristiques clés :

- Structure parfaitement équilibrée.
- Très utilisé en théorie des graphes et en cryptographie.
- Les cycles sont 2 réguliers.

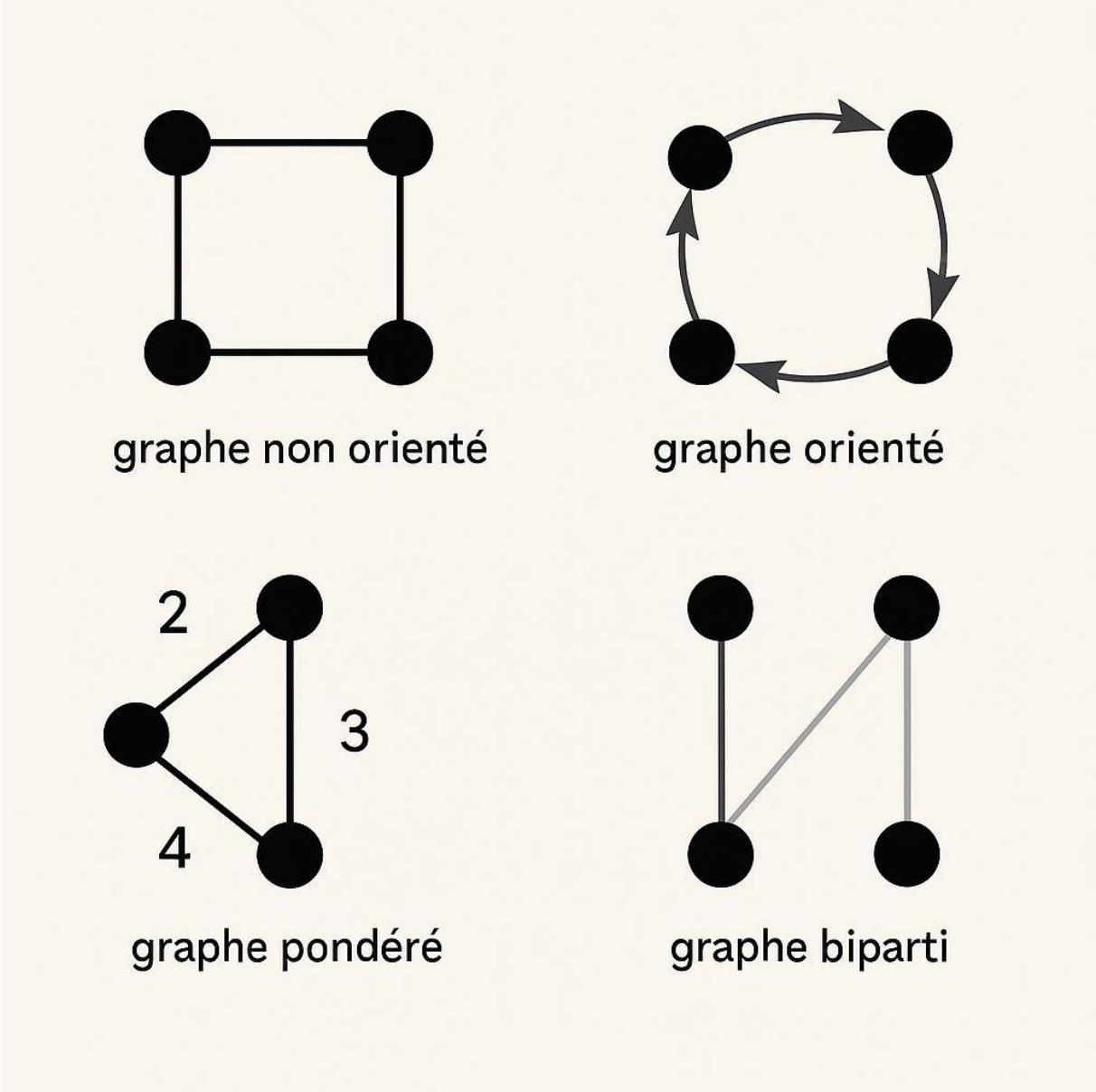
Exemple: Graphe 3 régulier Chaque sommet a exactement 3 voisins. (A) ----- (B) / \ / \ (F) (\) (/) (C) \ \ / // (E)---(D)---/ Tous les sommets ont 3 connexions → c'est bien un graphe 3 régulier.

c. Graphe biparti : Un graphe biparti est un graphe dont les sommets sont séparés en deux ensembles A et B, et où toutes les arêtes relient un sommet de A à un sommet de B (jamais à l'intérieur d'un même ensemble).

Caractéristiques clés :

- Aucun cycle impair n'est présent.
- Utilisé pour les systèmes d'affectation ou de matching - .- Toujours représentable sous forme de « deux colonnes ».

Exemple: Ensemble A Ensemble B {A, B, C} {1, 2, 3} (A) -------- (1) (B) -------- (2) (C) ---------- (3) Aucune arête ne relie deux sommets de A ou deux sommets de B : c'est un graphe biparti parfait.



6. Connexité & planarité

1. Connectivité

La connectivité indique s'il est possible de se déplacer d'un sommet à un autre dans un graphe.

- **Graphe connexe :** un graphe est connexe lorsqu'il existe au moins un chemin entre chaque paire de sommets. Cela signifie que le graphe forme un seul bloc.

- **Graphe fortement connexe :** cette notion concerne uniquement les graphes orientés. Un graphe est fortement connexe si, pour chaque couple de sommets, on peut aller de l'un à l'autre et revenir en respectant le sens des flèches.

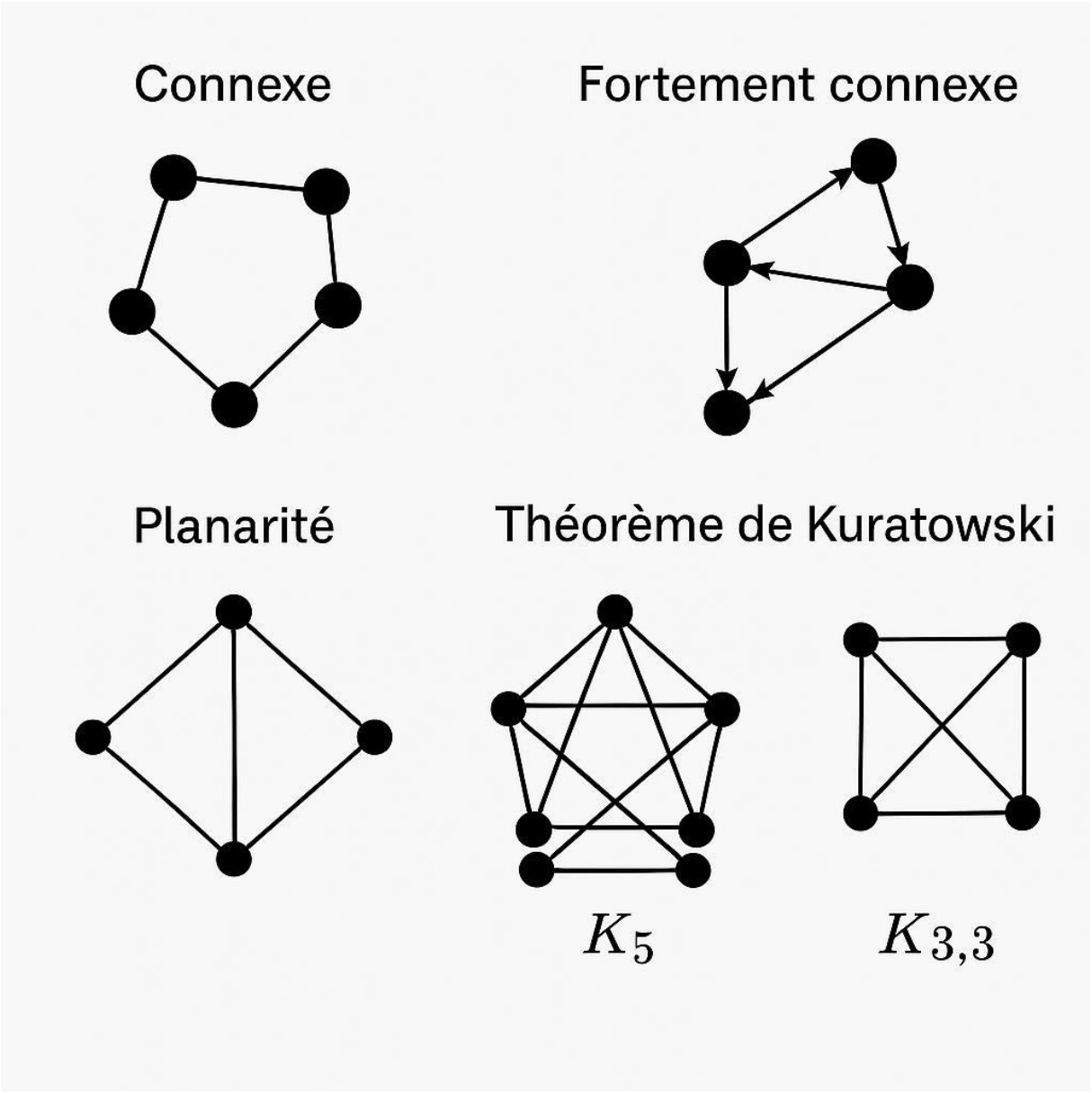
- **Composante connexe :** si un graphe n'est pas connexe, il est divisé en plusieurs parties isolées. Chaque partie connexe est appelée une composante connexe.

2. Planarité et théorème de Kuratowski

La planarité décrit la possibilité de représenter un graphe sur un plan sans croisement d'arêtes.

- **Graphe planaire :** un graphe est planaire s'il peut être dessiné sur une feuille sans que deux arêtes se croisent, même après déplacement des sommets.

- **Théorème de Kuratowski** : un graphe n'est pas planaire s'il contient l'un des deux graphes interdits : K5 (graphe complet à 5 sommets) ou K3,3 (graphe des trois maisons et trois usines)



7. Matrice d'adjacence

une matrice adjacente est souvent nommée A est une matrice carrée de forme n x n (ou n est le nombre de sommets du graphe). Aij ses éléments indiquent la présence ou l'absence d'une arête (ou d'un arc pour un graphe orienté) allant du sommet i au sommet j.

Graphe Non Pondéré (Simple) :

- AIJ =1 s'il existe une arête entre i et le sommet j.
- AIJ=0 sinon

GRAPHE PONDERE : Aij contient le poids (valeur) de l'arête entre i et j.

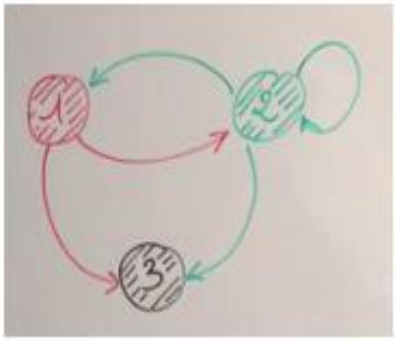
Boucles : si un sommet est relié a lui-même (boucle). L'élément diagonal correspondant Aij est généralement mis a 1 ou un poids spécifique. Bien que dans de nombreux contextes, les graphes simples n'aient pas de boucles, et donc la diagonale est remplie de zéros

Propriétés : Symétrie : pour un graphe non orienté, la matrice est symétrique (Aij =Aji pour tout i,j)car une arête lie a i a j et j a i Non symétrie : pour un graphe orienté, la matrice n'est généralement pas symétrique car un arc peut exister i vers j sans exister de j vers i

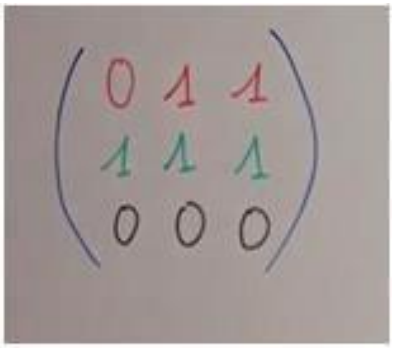
Degré Des Sommet : Dans un graphe non orienté, la somme des coefficients d'une ligne (ou colonne) i donne le degré du sommet i. Dans un graphe orienté, la somme d'une ligne i donne le demi degré extérieur (arcs sortants), et de la somme d'une colonne i donne le demi degré intérieur (arcs entrants)

Application : l'intérêt principal de la matrice adjacente réside dans ses application algorithmique et mathématiques

GRAPHE



MATRICE



8. Liste d'adjacence

Une liste adjacente est une manière de représenter un graphe. Au lieu d'utiliser un grand tableau (comme la matrice d'adjacence), on écrit pour chaque sommet la liste des sommets auxquels il est relié.

a. **Construction** : Une liste d'adjacence associe à chaque sommet la liste des sommets voisins. Elle se construit différemment selon que le graphe est orienté ou non orienté.

- **Graphes non orienté**: Pour une arête (u, v), on ajoute v à la liste de u et u à la liste de v
- **Graphe orienté :** Pour une arête orientée (u → v), on ajoute uniquement v à la liste de u.
- Représentation en mémoire Souvent avec des tableaux, dictionnaires ou listes chaînées selon le langage (C, Java, Python…).

b. **Analyse du stockage :** La liste d'adjacence nécessite O(V + E) unités de stockage, ce qui est très efficace pour les graphes clairsemés.

c. **Comparaison synthétique :**

- **Liste d'adjacence :** faible mémoire (O(V+E)), idéale pour les graphes clairsemés.
- **Matrice d'adjacence :** accès très rapide O(1), mais mémoire O(V^2).
- **Liste d'arêtes :** compacte O(E), mais lente pour accéder au voisinage

Partie III — Propriétés & théorèmes

9. Propriétés métriques

Les propriétés métriques servent à mesurer les distances dans un graphe, étudier sa structure interne et comprendre comment les sommets sont reliés entre eux. Elles sont essentielles pour analyser l'efficacité d'un réseau, la circulation d'informations, ou le comportement d'un système.

1. Le Degré D'un Sommet:

Le degré d'un sommet est le nombre d'arêtes qui lui sont connectées. Si une arête relie un sommet A à B, alors le degré de A augmente de 1.

Dans un graphe orienté :

Degré entrant (in-degree) : nombre d'arcs qui arrivent au sommet.
Degré sortant (out-degree) : arcs qui partent du sommet.
Degré total = entrant + sortant.

Pourquoi c'est important ?

Mesure l'importance ou la centralité d'un sommet.

En réseau social :

un sommet de grand degré = une personne très connectée.

En réseau routier : cela représente des

carrefours importants.

Notations :

deg(v) : degré du sommet v $deg^+(v)$:

degré sortant (graphe orienté) $deg^-(v)$:

degré entrant (graphe orienté)

Exemple : Si un sommet A est

connecté à B, C et D alors : deg(A) = 3

2. Distance Dans Un Graphe :

La distance entre deux sommets u et v (notée d(u, v)) est la longueur du plus court chemin qui les relie. Longueur = nombre d'arêtes parcourues. Si aucun chemin n'existe : d(u, v) = ∞ (infinie)

Types de distances :

- **Graphe non pondéré :** Chaque arête compte comme 1. On calcule avec un BFS (Breadth-First Search).
- **Graphe pondéré :** Chaque arête a un "coût" ou poids.

Distance = somme minimale des poids. (On utilise Dijkstra, Bellman-Ford…) **Utilité :** Comprendre la "proximité" entre sommets. Vérifier l'efficacité d'un réseau. Trouver les trajets optimaux (ex : GPS).

Exemple : Si le chemin le plus court entre A → D est : A – B – C – D Alors : d(A, D) = 3

3. Diamètre D'un Graphe :

Le diamètre est la plus grande distance entre deux sommets quelconques du graphe.

Formule : \text{Diamètre}(G) = \max_{u,v} d(u,v)

Intuition : C'est la "taille" du graphe en termes de distance interne. Plus le diamètre est faible, plus tout est relié efficacement.

Exemple : Distances maximales entre sommets : d(A, D) = 3

d(A, C) = 2 d(B, D) = 2 Donc le diamètre = 3

Cas particuliers :

Graphe complètement connecté = diamètre = 1 Graphe en ligne de n sommets = diamètre = n – 1

Utilité : Mesurer la performance d'un réseau : Réseaux sociaux Internet Réseaux électriques Dans un réseau social, un faible diamètre indique un phénomène "smallworld".

Degré : nombre d'arêtes d'un sommet. Distance : longueur du plus court chemin entre deux sommets.

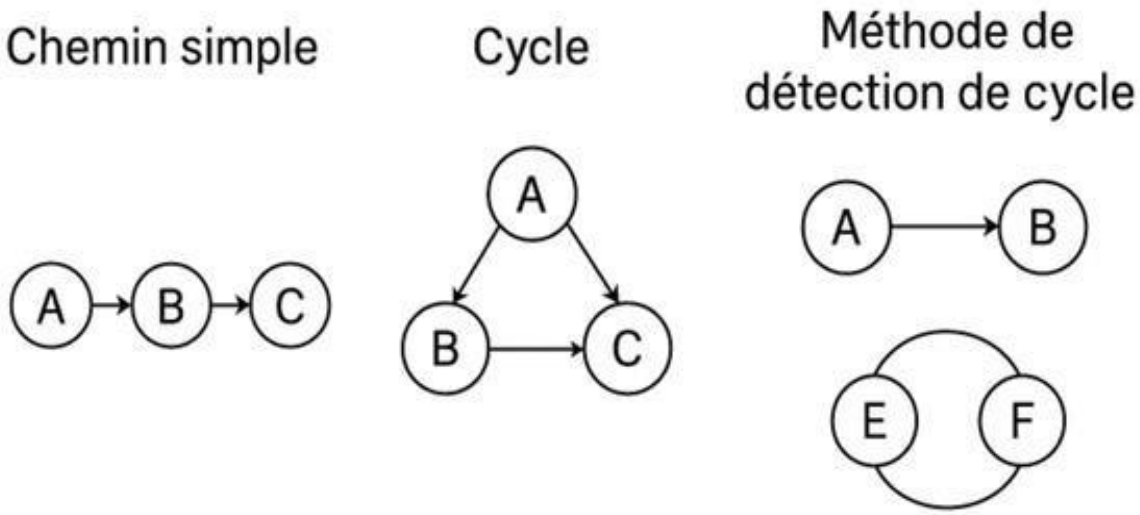
Diamètre : plus grande distance entre deux sommets du graphe.

10. Cycles, chemins & composantes

a. **Chemin simple :** Un chemin simple est une suite de sommets dans un graphe où chaque sommet est visité une seule fois. Exemple dans le schéma : A → B → C → D est un chemin simple.
b. **Cycle (ou circuit) :** Un cycle est un chemin qui commence et se termine au même sommet, sans répéter d'arête ni de sommet (sauf le point de départ/arrivée). Exemple dans le schéma : A → B → C → A est un cycle.

c. **Méthodes de détection de cycle :**

- En parcours en profondeur (DFS) : on détecte un cycle si on revisite un sommet déjà présent dans le chemin en cours.
- En parcours en largeur (BFS) avec suivi des parents. ○ Pour les graphes orientés, on utilise aussi la détection de cycle par couleur (blanc, gris, noir).



11. Théorèmes essentiels (1)

- **Théorème d'Euler** : un graphe connexe possède un cycle eulérien (c'est-à-dire un parcours qui passe une seule fois par chaque arête et revient au point de départ) si et seulement si tous les sommets ont un degré pair.
- **Théorème de Dirac** : un graphe simple ayant au moins n≥3 sommets est hamiltonien (c'est-à-dire qu'il existe un cycle passant une seule fois par chaque sommet) si chaque sommet a un degré supérieur ou égal à n/2.

12. Théorèmes essentiels (2)

Kuratowski

En 1930, le mathématicien polonais Kazimierz Kuratowski a établi une caractérisation complète et élégante des graphes planaires. Son théorème identifie précisément quels graphes sont "responsables" de la non-planarité.

Les graphes interdits : K_5 et $K_{3,3}$

Nous avons déjà vu que K_5 (graphe complet à 5 sommets) et $K_{3,3}$ (graphe biparti complet) ne sont pas planaires. Ces deux graphes jouent un rôle fondamental dans la théorie de la planarité.

Subdivision d'un graphe

Une subdivision d'un graphe est obtenue en ajoutant un ou plusieurs sommets sur une ou plusieurs arêtes. Par exemple, transformer l'arête A—B en A—X—B (avec un nouveau sommet X) est une subdivision.

Énoncé du théorème

Théorème de Kuratowski (1930)

Un graphe fini est planaire si et seulement s'il ne contient pas de sous-graphe qui soit une subdivision de K_5 (le graphe complet à 5 sommets) ou de $K_{3,3}$ (le graphe biparti complet à 3+3 sommets).

En d'autres termes :

- Si un graphe contient K_5 ou $K_{3,3}$ (ou une de leurs subdivisions), il n'est PAS planaire
- Si un graphe ne contient ni K_5 ni $K_{3,3}$ (ni leurs subdivisions), alors il EST planaire

Importance et conséquences

Le théorème de Kuratowski est remarquable car il réduit le problème de la planarité à la recherche de deux graphes spécifiques. K_5 et $K_{3,3}$ sont les "plus petits" graphes non planaires, et tout graphe non planaire contient nécessairement l'un d'eux.

Exercice : Test de planarité avec Kuratowski

Soit G un graphe formé de 6 sommets disposés en hexagone, où chaque sommet est relié à ses deux voisins. G est-il planaire ?

Solution :

Analysons ce graphe :

- s = 6 sommets
- a = 6 arêtes

Vérifions : $a \leq 3s - 6$

$6 \leq 3(6) - 6 = 12$ ✓

Ce graphe est simplement un cycle de longueur 6. Il peut être dessiné sans croisement (c'est l'hexagone lui-même). Il ne contient ni K_5 ni $K_{3,3}$, ni aucune de leurs subdivisions.

Conclusion : Le graphe est planaire.

Exercice 3 : Problème avancé

Montrez qu'un graphe complet K_n est planaire si et seulement si $n \leq 4$.

Solution :

Sens direct (⇒) :

Supposons K_n planaire. Alors K_n ne peut pas contenir K_5 comme sous-graphe (par Kuratowski). Or, si $n \geq 5$, alors K_n contient K_5 (il suffit de prendre 5 sommets quelconques). Donc $n \leq 4$.

Sens réciproque (⇐) :

- K_1, K_2, K_3 sont trivialement planaires (points, segment, triangle)
- K_4 peut être dessiné comme un tétraèdre aplati : un triangle avec un point au centre relié aux trois sommets du triangle

Partie IV — Algorithmique

13. BFS (Breadth-First Search)

- **Idée** : exploration du graphe par couches successives, en visitant d'abord tous les voisins proches avant de passer aux suivants.

- **Cas d'usage** : calcul des distances minimales dans un graphe non pondéré, identification des composantes connexes.

14. DFS (Depth-First Search)

- **Idée** : exploration en profondeur, on suit un chemin jusqu'au bout avant de revenir en arrière pour explorer les autres.

- **Cas d'usage** : détection de cycles, ordonnancement topologique dans les graphes orientés acycliques.

15. Dijkstra & Bellman-Ford

• Dijkstra

Principe général L'algorithme de Dijkstra est une approche gloutonne (greedy). Il construit progressivement l'ensemble des sommets dont la distance depuis la source est définitivement minimale. À chaque étape :

a. On sélectionne le sommet non traité ayant la plus petite distance connue (à l'aide d'une file de priorité).
b. On met à jour ("relaxe") les distances de ses voisins.
c. On répète jusqu'à ce que tous les sommets soient traités.

NB: Condition importante : Dijkstra ne fonctionne correctement que si tous les poids sont positifs ou nuls.

Étapes simplifiées Initialiser toutes les distances à +∞ sauf la source (0).

- Placer tous les sommets dans une file à priorité.
- Répéter : extraire le sommet le plus proche, mettre à jour ses voisins.
- **Une fois sorti de la file, la distance d'un sommet est définitive. Complexité**

Avec une file de priorité (tas binaire) :

O(|E| \log |V|)

Dijkstra est donc très rapide et adapté aux grands graphes, notamment pour les systèmes GPS ou les réseaux routiers.

Exemple illustratif

Graphe : A (source), B, C, D, E

Poids : A→B = 4, A→C = 2, B→C = 1, B→D = 5, C→D = 8, C→E = 10, D→E = 2

Étape Sommet extrait dist[] après mise à jour

Init	-	A=0, B=∞, C=∞, D=∞, E=∞
1	A	B=4, C=2
2	C	D=10, E=12
3	B	D=9
4	D	E=11
5	E	distances finales

Distances finales : A → B = 4, A → C = 2, A → D = 9, A → E = 11

- **Bellman-Ford** :

Principe général Bellman–Ford repose sur une technique appelée relaxation répétée. Au lieu d'explorer le graphe de façon gloutonne, il passe sur toutes les arêtes et tente d'améliorer les distances. Il répète ce processus |V| - 1 fois, car la plus longue chaîne simple possible dans un graphe contient (|V|-1) arêtes. Après ces répétitions, les distances sont garanties correctes à condition qu'il n'y ait pas de cycle négatif.

Grand avantage : poids négatifs

Contrairement à Dijkstra, Bellman–Ford peut gérer :

- Arêtes avec poids négatifs

- Détection des cycles de poids négatif

C'est l'algorithme utilisé dans les protocoles de routage RIP.

Détection des cycles négatifs

Après les |V|-1 itérations, Bellman–Ford réalise une passe supplémentaire : Si une distance peut encore être améliorée → le graphe contient un cycle négatif. C'est une fonctionnalité importante absente de Dijkstra.

Complexité

O(|V| \cdot |E|)

Il est plus lent que Dijkstra, mais beaucoup plus général.

Exemple illustratif (avec poids négatif mais sans cycle négatif)

Graphes : S (source), A, B, C

Arêtes :

S→A = 4

S→B = 5

A→B = -3

B→C = 6

A→C = 5

Itération 1 :

S→A = 4

S→B = 5

A→B = 1

B→C = 7

A→C = 9 (inutile car 7 est meilleur) Itérations 2 et 3 : Aucun changement → distances stables

Bellman–Ford trouve donc : dist[S]=0, A=4, B=1, C=7

Aucun cycle négatif détecté. 15.4.

Tableau comparatif : Dijkstra vs Bellman–Ford

Algorithmes d'optimisation (1) – Dijkstra & Bellman–Ford

Critère	Dijkstra	Bellman–Ford
Poids autorisés	uniquement ≥ 0	négatifs acceptés
Complexité	$O(\|E\| \log \|V\|)$	$O(\|V\| \cdot \|E\|)$
Détection cycle négatif	Non	Oui (itération supplémentaire)
Méthode	Greedy	Relaxation répétée
Usage typique	Réseaux routiers, graphes sans poids négatifs	Graphes avec poids négatifs, détection de cycles négatifs

16. Prim & Kruskal

- **Prim** : construit un arbre couvrant minimum en ajoutant progressivement les arêtes les moins coûteuses reliées à l'arbre en cours.
- **Kruskal** : trie toutes les arêtes par poids et les ajoute une à une si elles ne créent pas de cycle, jusqu'à obtenir un arbre couvrant minimum.
- **Exemple concret** : relier un ensemble de villes avec un coût minimal de construction de routes.

Partie V — Applications & exercices

17. Exercices

Exercice 1 — Matrice d'adjacence

Énoncé : Construis la matrice d'adjacence du graphe suivant : Sommets = {A, B, C, D}, Arêtes = {A–B, A–C, B–C, C–D}.

Correction :

```
  A B C D
A [0 1 1 0]
B [1 0 1 0]
```

C [1 1 0 1]

D [0 0 1 0]

Chaque ligne/colonne indique les voisins directs.

Exercice 2 — Liste d'adjacence

Énoncé : Donne la liste d'adjacence du graphe précédent.

Correction :

- A : [B, C]
- B : [A, C]
- C : [A, B, D]
- D : [C]

Exercice 3 — Graphe eulérien

Énoncé : Vérifie si le graphe précédent est eulérien.

Correction :
Degrés : A=2, B=2, C=3, D=1.
Comme deux sommets (C et D) ont un degré impair, le graphe n'est pas eulérien.

Exercice 4 — Graphe hamiltonien (Dirac)

Énoncé : Considère un graphe complet (K_4). Est-il hamiltonien selon le théorème de Dirac ?

Correction :
(n=4). Chaque sommet a un degré de 3.
Condition de Dirac : degré ≥ (n/2 = 2).
Ici, 3 ≥ 2 → condition satisfaite → (K_4) est hamiltonien.

Exercice 5 — Connexité

Énoncé : Le graphe avec sommets {A, B, C, D} et arêtes {A–B, B–C} est-il connexe ?

Correction :

Le sommet D est isolé → il n'existe pas de chemin vers D. Le graphe est non connexe.

Exercice 6 — Planarité

Énoncé : Le graphe complet (K_5) est-il planaire ?

Correction :
Par le théorème de Kuratowski, (K_5) contient une subdivision de lui-même → il est non planaire.

Exercice 7 — BFS

Énoncé : Applique BFS sur le graphe avec sommets {A, B, C, D, E} et arêtes {A–B, A–C, B–D, C–E}, en partant de A.

Correction :

- Niveau 0 : A
- Niveau 1 : B, C
- Niveau 2 : D, E
  Distances : d(A)=0, d(B)=1, d(C)=1, d(D)=2, d(E)=2.

Exercice 8 — DFS

Énoncé : Applique DFS sur le même graphe en partant de A.

Correction :
Parcours possible : A → B → D → retour → C → E. Ordre de visite : A, B, D, C, E.

Exercice 9 — Dijkstra

Énoncé : Graphe pondéré : sommets {A, B, C}, arêtes : A–B (poids 2), A–C (poids 5), B–C (poids 1). Trouve le plus court chemin de A à C.

Correction :

- A→C direct = 5. • A→B→C = 2 + 1 = 3.
  Le plus court chemin est A–B–C avec distance 3.

Exercice 10 — Prim vs Kruskal

Énoncé : Graphe pondéré : sommets {A, B, C, D}, arêtes : A–B (1), B–C (2), C–D (3), A–D (4). Compare Prim et Kruskal.

Correction : • Prim (départ A) : A–B (1), B–C (2), C–D (3). • Kruskal : trie les arêtes → (1, 2, 3, 4). Ajoute A–B, B–C, C–D. Résultat identique : arbre couvrant minimum de poids 6.

Exercice 11 — Matrice d'adjacence (4 sommets)

Énoncé : Construis la matrice d'adjacence du graphe simple suivant : Sommets = {A, B, C, D}, Arêtes = {A–B, A–C, B–D}.

Correction :

Code

```
  A B C D
A [0 1 1 0]
B [1 0 0 1]
C [1 0 0 0]
D [0 1 0 0]
```

Exercice 12 — Graphe eulérien

Énoncé : Vérifie si le graphe ci-dessus est eulérien.

Correction : Degrés : A=2, B=2, C=1, D=1. Deux sommets (C et D) ont un degré impair → le graphe n'est pas eulérien.

Exercice 13 — BFS

Énoncé : Applique BFS sur le graphe avec sommets {A, B, C, D, E} et arêtes {A–B, A–C, B–D, C–E}, en partant de A.

Correction :

- Niveau 0 : A
- Niveau 1 : B, C
- Niveau 2 : D, E Distances : d(A)=0, d(B)=1, d(C)=1, d(D)=2, d(E)=2.

Exercice 14 — Prim vs Kruskal

Énoncé : Graphe pondéré : sommets {A, B, C, D}, arêtes : A–B (1), B–C (2), C–D (3), A–D (4). Compare Prim et Kruskal.

Correction :

- Prim (départ A) : A–B (1), B–C (2), C–D (3). • Kruskal : trie les arêtes → (1, 2, 3, 4). Ajoute A–B, B–C, C–D. Résultat identique :
  arbre couvrant minimum de poids 6.