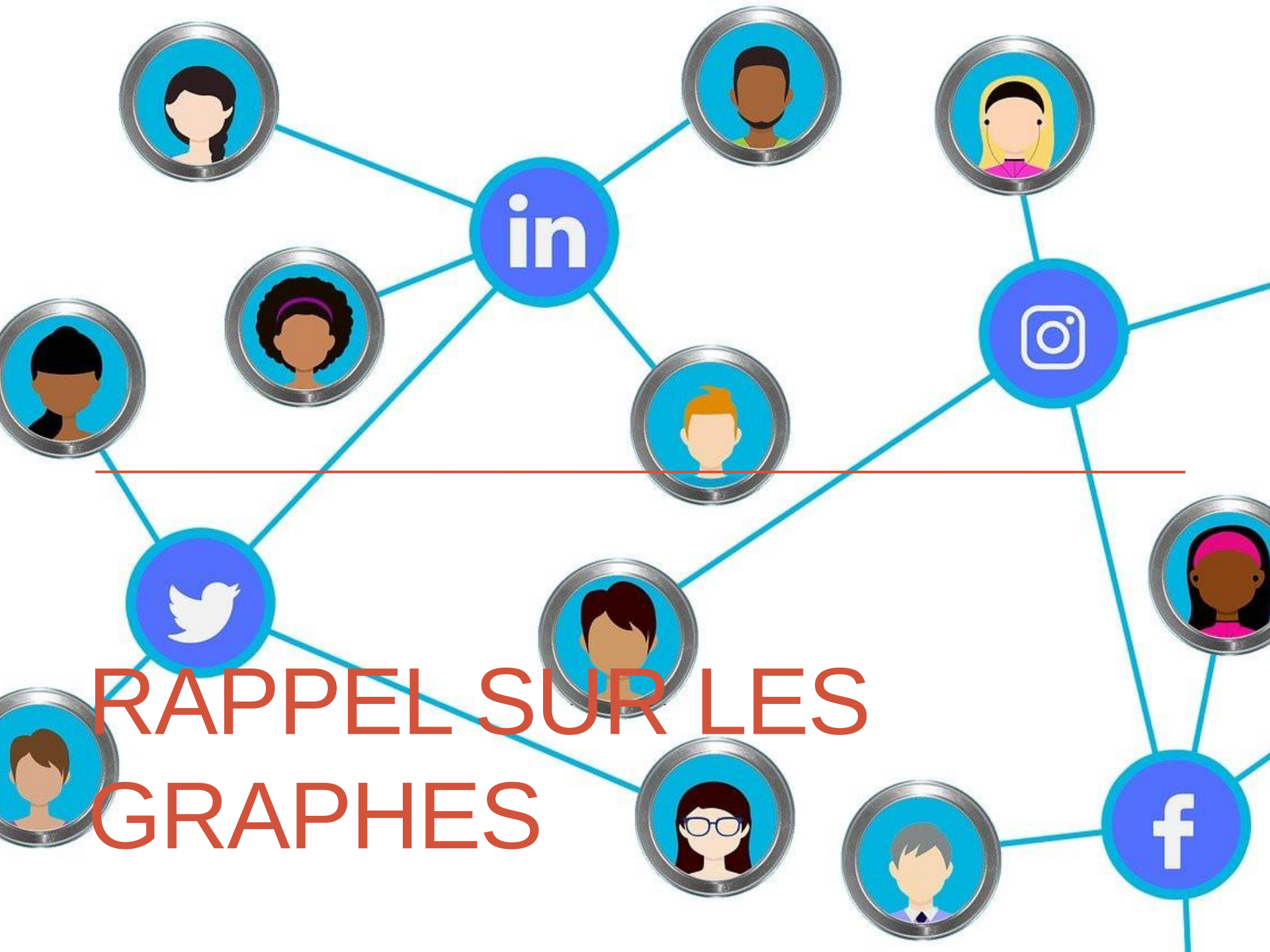
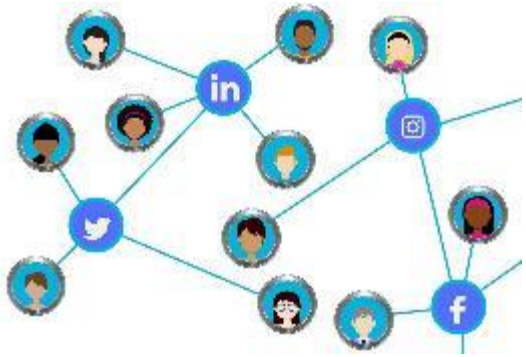


RAPPEL SUR LES GRAPHES

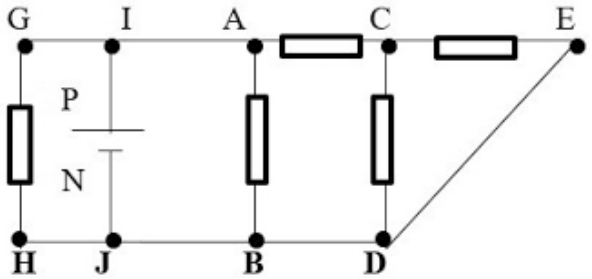
Un **graphe** est un objet mathématique utilisé afin de modéliser des éléments et leurs relations.

Un réseau social:



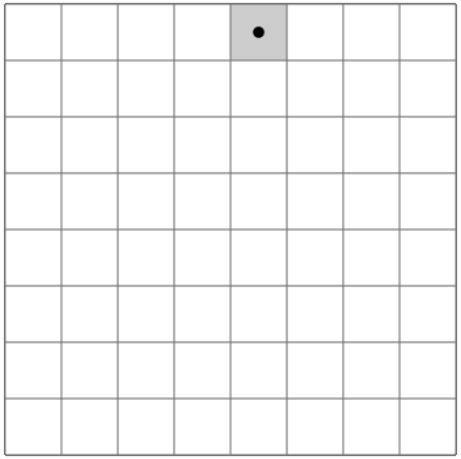
Le réseau du métro:

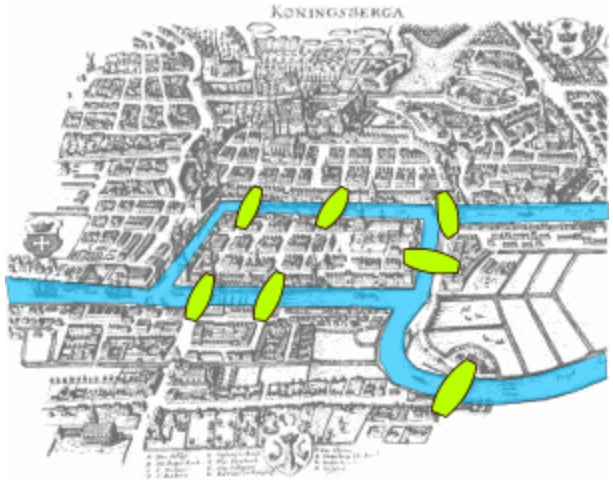
Un circuit électrique:



Les problèmes à résoudre sur un graphe:

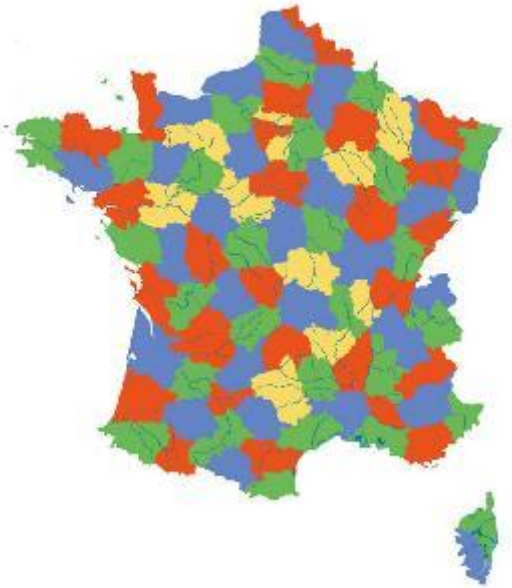
Le problème du cavalier: comment parcourir toutes les cases d'un échiquier une seule fois avec un cavalier (al-Adli ar-Rumi, 840)





Les sept ponts de Königsberg: comment passer par tous les ponts de Königsberg une seule fois et revenir à son point de départ (Euler, 1735)

Problème des quatre couleurs: peut-on colorier une carte avec quatre couleurs de façon à ce que deux territoires adjacents n'ait jamais la même couleur (Appel et Haken, 1976)?



Problème du plus court chemin: comment trouver le plus court chemin permettant de relier deux villes entre elles (Dijkstra, 1959)?

Comment construire un réseau autoroutier de manière à limiter les bouchons le plus possible?

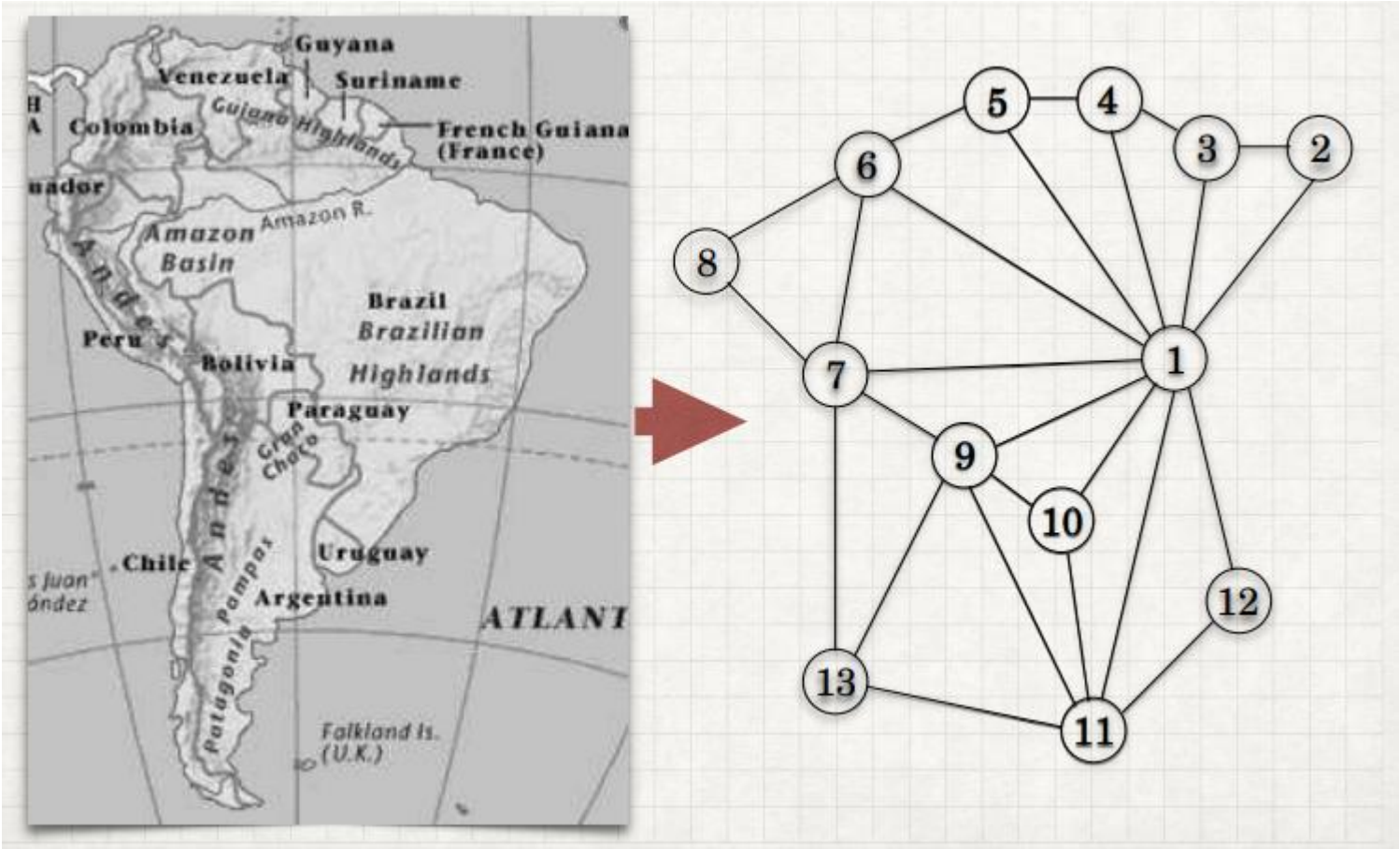
Comment construire les emplois du temps de tout le lycée sans problèmes de salle?

EMPLOI DU TEMPS PSI

	LUNDI	MARDI	MERCREDI	JEUDI	VENDREDI
8h10					
8h45	INFO	ANGLAIS	MATHS Gr. B / PHYS 5.75/TP Gr. A	MATHS	
10h05	PHYSIQUE	MATHS	PHYS 5.75/TP Gr. B / MATHS Gr. A	MATHS Gr. A / SI Gr. B 5.50 / SI Gr. A 5.50 / MATHS Gr. B	MATHS
12h10					
13h15					
14h15	SI	PHYSIQUE	FRANCAIS	PHYSIQUE	TP INFO
15h10	PHYS Gr. B / SI 5.50 Gr. A	TIPE 3.5D/TP		EPS	
17h15	PHYS Gr. A / SI 5.50 Gr. B				
18h15					

Pourquoi les graphes?

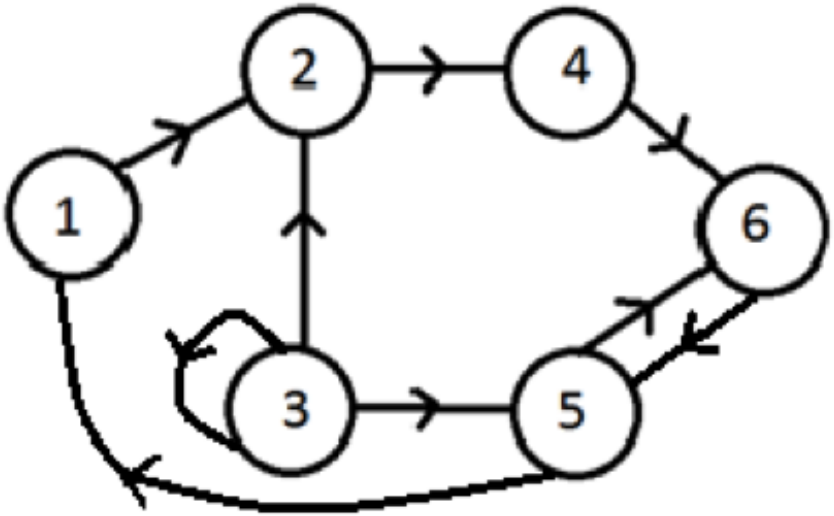
Ils permettent d'enlever les informations superflues



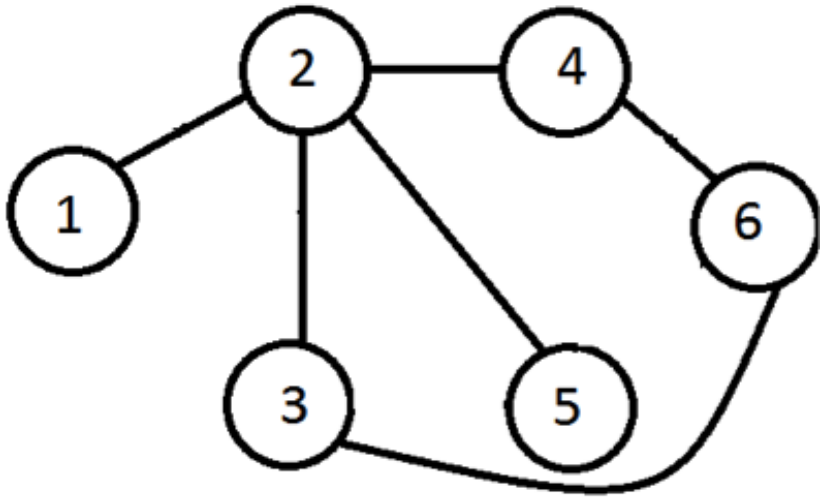
Les types de graphes

Un **graphe orienté** G = (S,A) est un couple défini par deux ensembles:

1. Un ensemble de points nommés **sommets** S = {s_1, s_2,…,s_n}: une place
2. Un ensemble de flèches nommées **arcs** A = {a_1,a_2,…,a_p}: des routes



Exemple: systèmes routiers avec sens uniques, flot de contrôle d'un programme



Un graphe **non orienté** G = (S,A) est un couple défini par deux ensembles:

1. Un ensemble de points nommés **sommets** S = $\{s_1, s_2,\ldots,s_n\}$.

2. Un ensemble de lignes nommées **arêtes** A = $\{a_1,a_2,\ldots,a_p\}$.

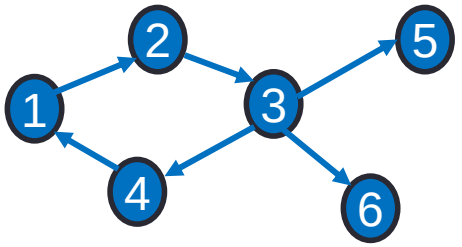
Remarque: Les arêtes d'un graphe non orienté sont notées entre accolades. Les arcs d'un graphe orienté sont notés entre parenthèse.

- Le graphe des utilisateurs de Facebook a un sommet par utilisateur et une arête entre deux sommets lorsque deux utilisateurs sont *amis*. Noter que $S = 10^9$ et que $A = 10^{11}$.

- Le graphe des utilisateurs de Facebook a un sommet par utilisateur et une arête entre deux sommets lorsque deux utilisateurs sont *amis*. Noter que $S = 10^9$ et que $A = 10^{11}$.

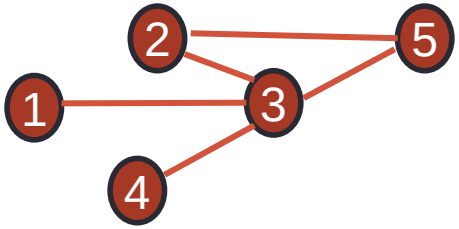
- Dans le graphe du métro parisien, les stations sont les sommets et les arêtes, les liaisons entre ces stations.

Vocabulaire



1) Graphes orientés

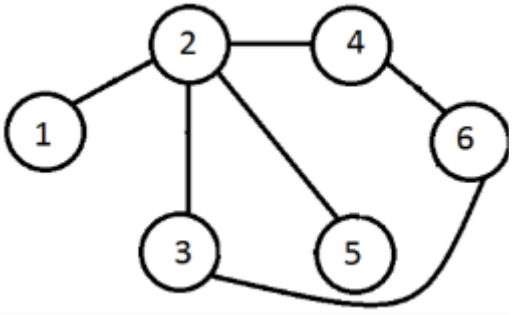
- Soit *a* = (*s*,*s'*) un arc d'un graphe orienté G = (S,A).
  - *s'* est appelé le successeur de *s*. Ex:
  - *s* est appelé le prédécesseur de *s'*. Ex:
  - le nombre d'arcs sortant de *s* est appelé degré sortant de *s*. Ex:
  - le nombre d'arcs entrant dans *s* est appelé degré entrant de *s*. Ex:
- On appelle chemin une suite d'arcs qui se suivent. Ex:
- Un circuit est un chemin qui forme une boucle. Ex:



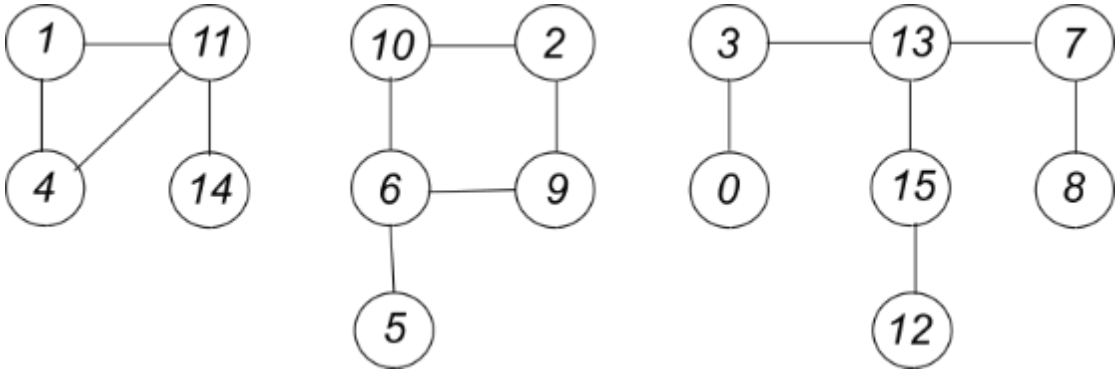
2) Graphes non orientés

- Le degré d'un sommet est le nombre d'arêtes liées à *s*.
- Une chaîne est une suite d'arêtes qui se suivent.
- Un cycle est une chaîne qui forme une boucle.

Un graphe non orienté est dit connexe si pour toute paire de sommets distincts {*s*,*s'*}, il existe au moins une chaîne reliant *s* à *s'*.



Un graphe connexe

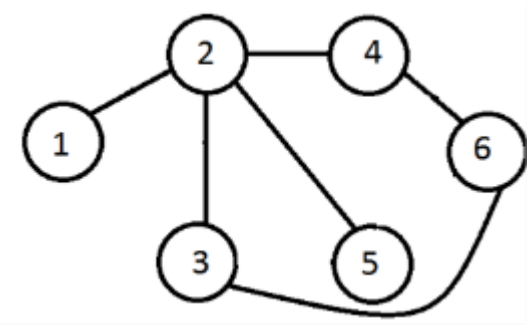


Un graphe non connexe avec trois composantes connexes

- Matrice d'adjacence: la matrice d'adjacence associée à un graphe non orienté G est la matrice (A_{ij}) telle que:

$A_{ij} = 1$ si $\{s_i, s_j\}$ est une arête
$A_{ij} = 0$ sinon

- Pour:



- Liste d'adjacence: c'est une liste à n lignes (une ligne par somme) telle que la $i^{ème}$ ligne est la liste des sommets ayant une arête commune avec le $i^{ème}$ sommet s_i.

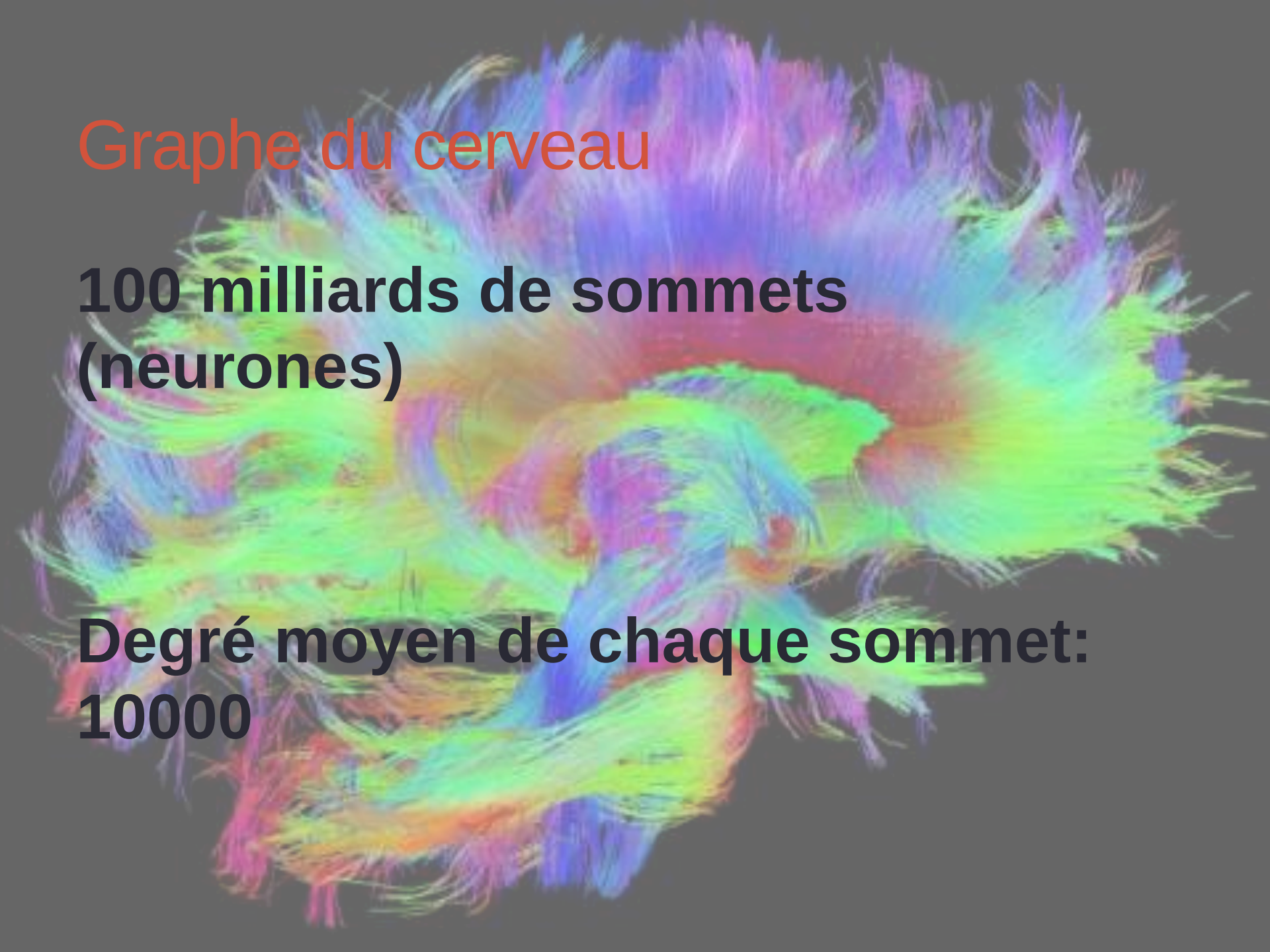
Graphe de Facebook

2 milliards de sommets (utilisateurs)

Degré moyen d’un sommet: 338 (338 amis en moyenne).


Graphe de Wikipédia
10,5 millions d’articles
5 milliards de liens

Graphe du cerveau



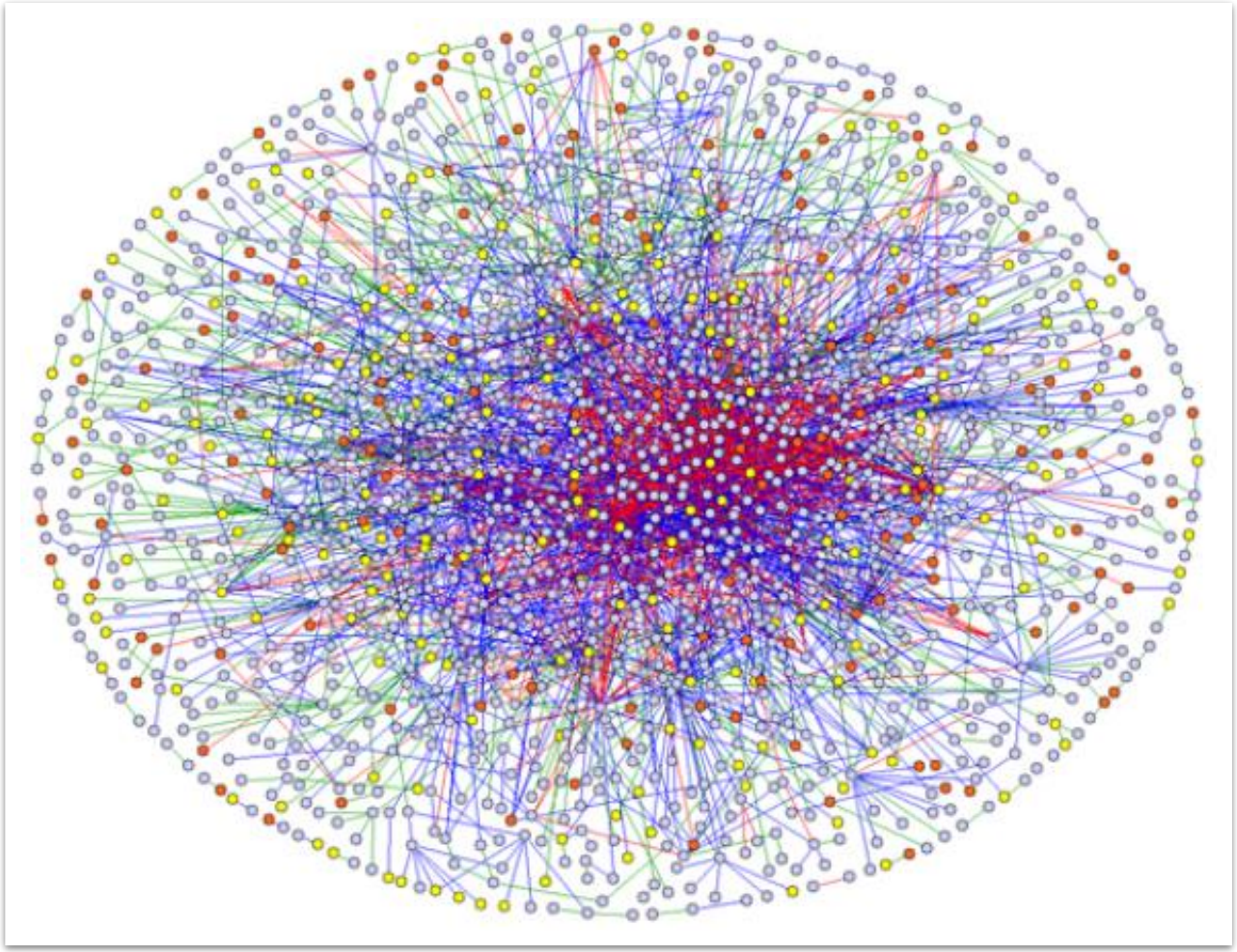
100 milliards de sommets (neurones)

Degré moyen de chaque sommet: 10000

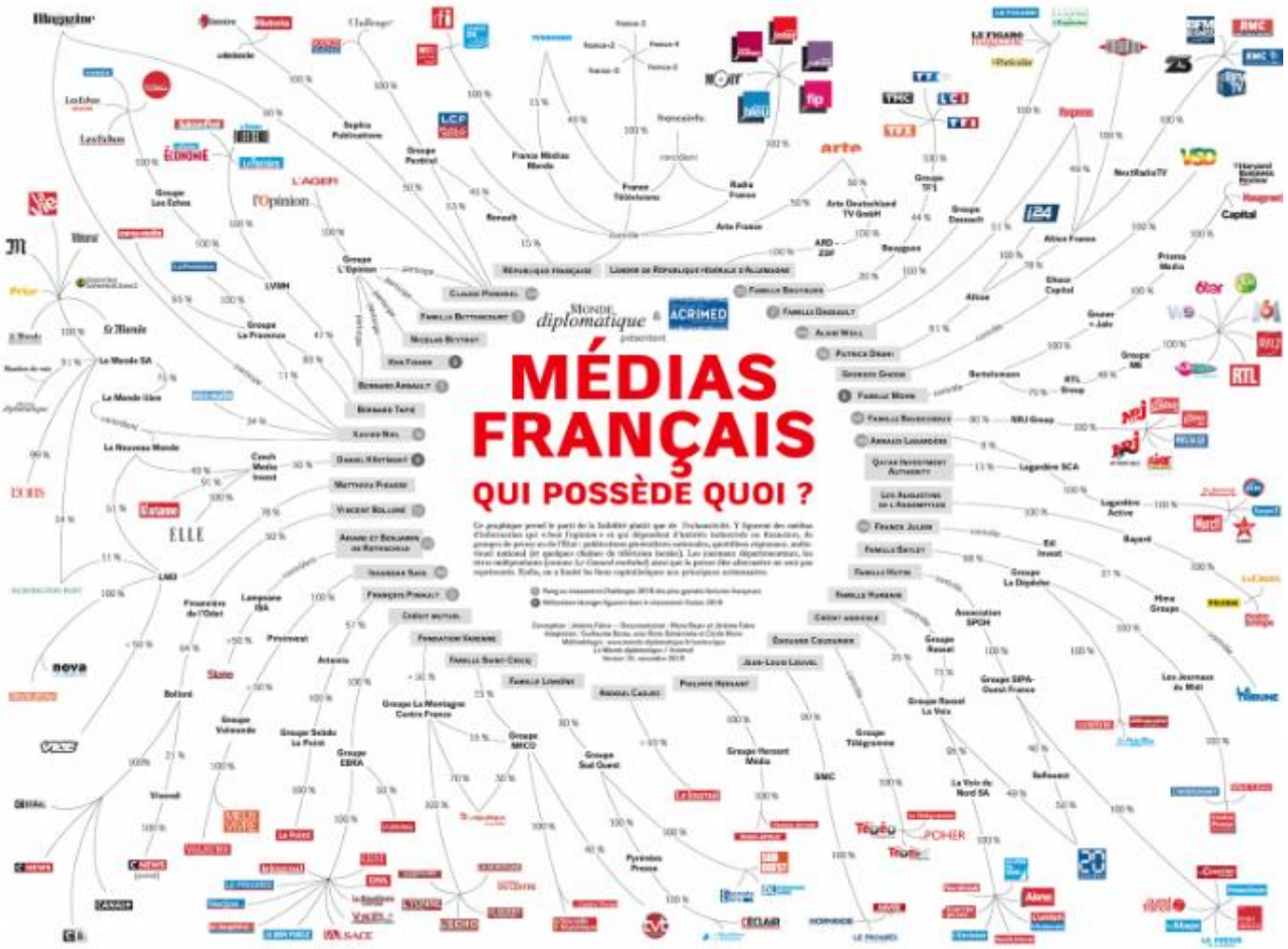
Graphe des protéines du corps humain

1700 protéines humaines

3200 interactions entre elles.



Les médias français



Parcours d'un graphe

- Parcourir un graphe consiste à explorer tous les sommets d'un graphe. On peut en déduire les relations entre les sommets, entre les arcs, détecter les cycles…

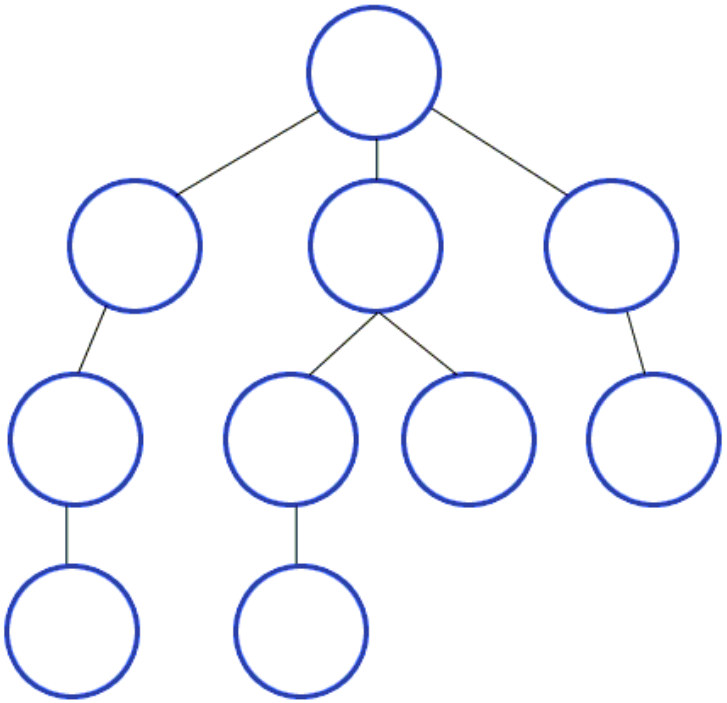
- Lors du parcours d'un graphe, les sommets passent par deux états successifs:
  - Etat **non marqué**: le sommet n'a pas encore été visité
  - Etat **marqué**: le sommet a été visité.

- Il existe deux grands types de parcours d'un graphe:
  - Le parcours en **largeur**.
  - Le parcours en **profondeur**.

- Lorsque le graphe est parcouru, on peut facilement trouver le chemin le plus court entre deux sommets, savoir s'il possède des cycles.

Parcours en largeur

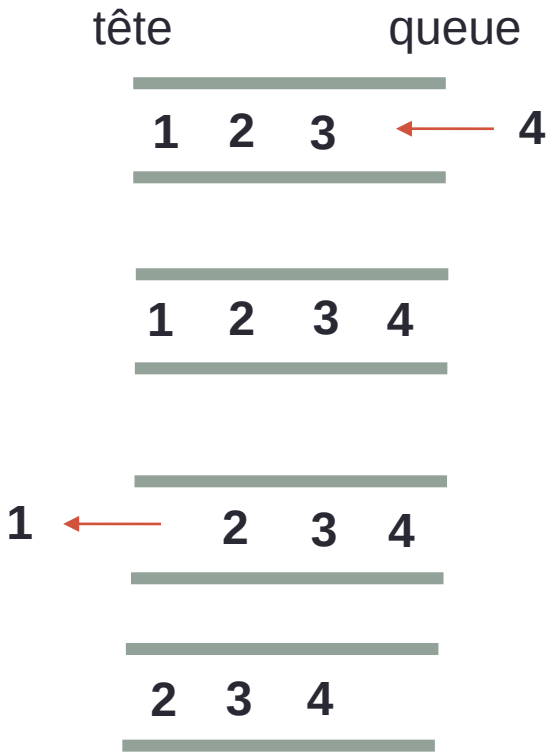
- On part d'un sommet choisi au hasard (appelé *racine*) qu'on marque. Puis on visite tous les successeurs de la racine, puis tous les successeurs non encore visités des successeurs etc…



Remarque: le cas d'un graphe non orienté s'en déduit trivialement.

Pour parcourir les différents sommets, on utilise une file (FIFO: *first in*, *first out)*.

Une file fonctionne comme une queue à la Poste: les sommets rentrent dans la file par la queue et en sortent par la tête lorsqu'ils sont marqués.

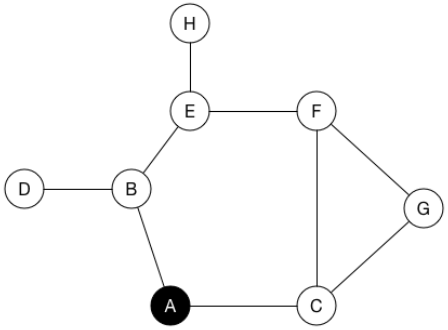


Principe de l'algorithme

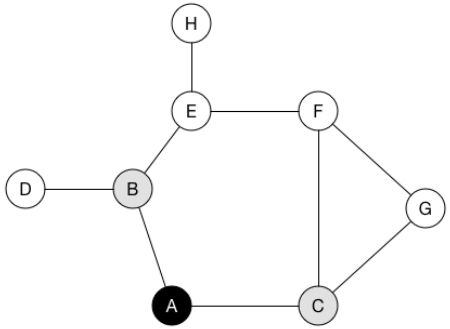
1. On choisit au hasard un sommet non encore visité. On l'enfile dans une file.

2. Tant que la file n'est pas vide:
   a) La tête de la file est marquée comme visité.
   b) On enfile tous les successeurs non visités du sommet de la file.
   c) On désenfile la tête et on avance tous les autres éléments de la file.
   d) On recommence.

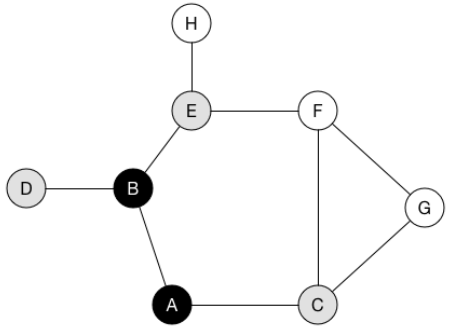
3. Si la file est vide et qu'il reste des sommets non marqués, on recommence à l'étape 1.



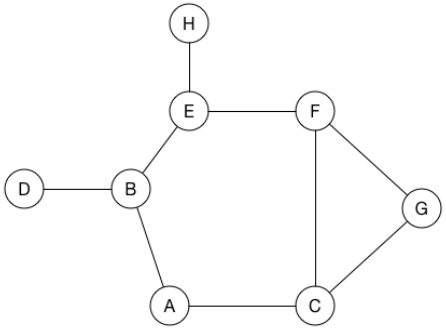
A



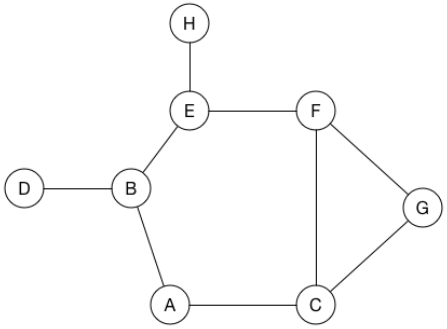
A ← B C

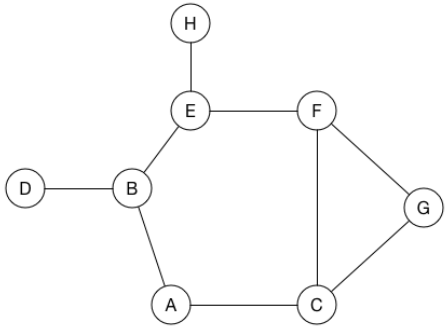


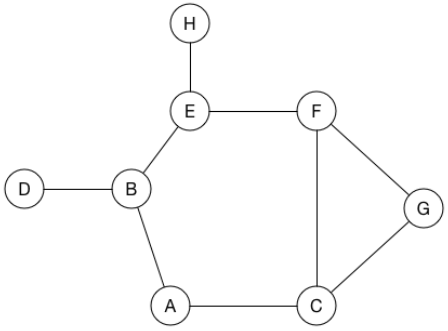
A ← B C ← D E

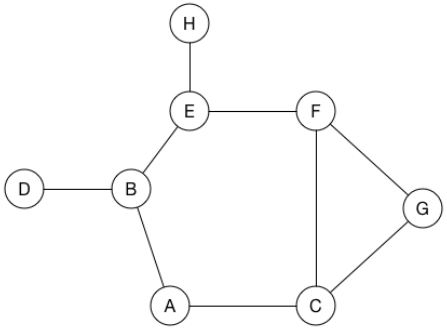


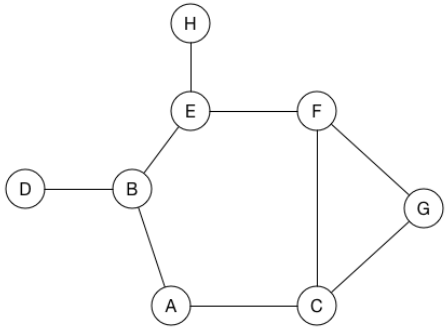
B ← C D E ←

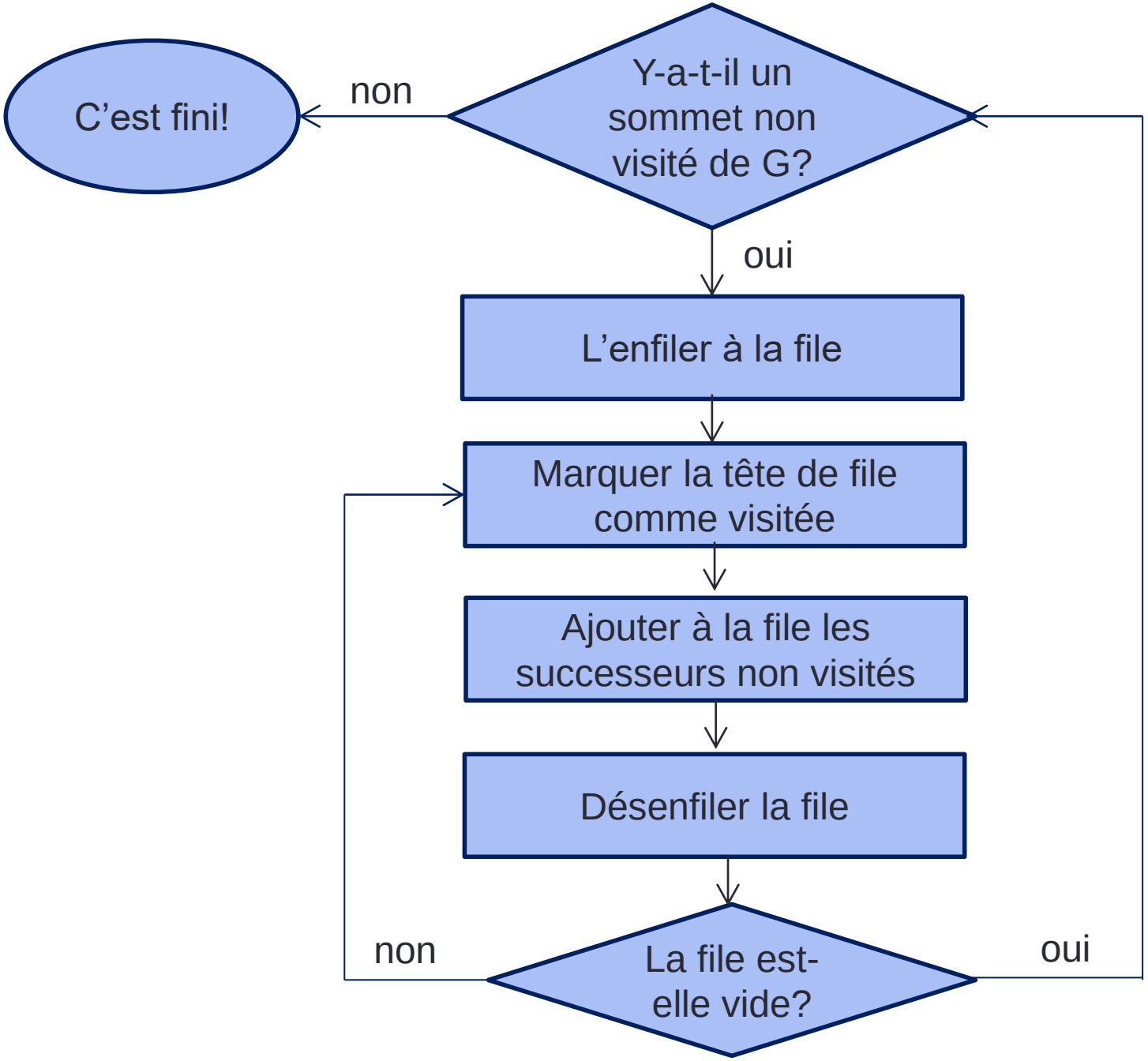











Y-a-t-il un sommet non visité de G?
non
C’est fini!
oui
L’enfiler à la file
Marquer la tête de file comme visitée
Ajouter à la file les successeurs non visités
Désenfiler la file
La file est-elle vide?
non
oui

Codage en Python

- Le graphe est représenté par un **dictionnaire** contenant sa liste d'adjacence:

Graphe = {'A': ['B','C'],'B' = ['A','D','E'],'C' = ['A','F','G'], 'D' = ['B'], 'E' = ['B','F','H'], 'F' = ['C','E','G'], 'G' = ['C','F'], 'H' = ['E']}

- La file sera représentée à l'aide d'une **liste** F:
  - Pour enfiler un sommet s en queue de la liste F, on utilise:
  - Pour supprimer le sommet en tête de liste, on utilise:

- L'état des sommets sera codé à l'aide d'un **dictionnaire**:
  - Les clés sont les noms des sommets,
  - Les valeurs sont l'état: True (sommet visité) ou False (sommet non visité)

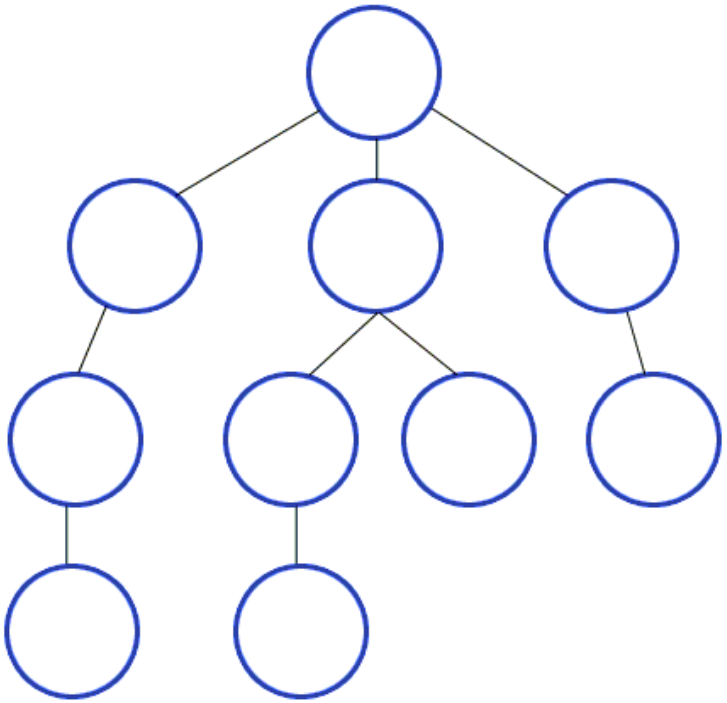
Etats = {'A'=True, 'B' = True, 'C' = True, 'D' = False, 'E' = False, 'F' = False, 'G' = False, 'H' = False}

- Le code renvoie un dictionnaire contenant le prédécesseur de chaque sommet. Les racines n'ont pas de prédécesseurs ('None').

```python
def parcours_largeur(G):
    """Parcours en largeur d'un graphe. La fonction renvoie le dictionnaire
    contenant un prédecesseur de chaque sommet"""
    etats = dict()
    for s in G:
        etats[s] = False #
    pred = dict()
    for r in G:
        if etats[r] == False:
            pred[r] = 'None' # r est une ...................
            file = [r] #.................................
                while ...................: # tant que la file n'est pas vide
                    r = file[0]
                    ................... # marquer r comme visité
                    for s in G[r]: # pour tous les ...................
                        if etats[s] == False:
                            ................. #rajouter s à la file
                            pred[s] = r
                    ............. #virer r de la file.
    return pred
```

Parcours en profondeur

- On part d'un sommet choisi au hasard (appelé *racine*) qu'on marque. Puis on visite un successeur de la racine, puis un successeur du successeur etc… Lorsqu'un sommet n'a pas de successeur (cul-de-sac), on revient au sommet précédemment visité et on recommence.



Pour parcourir les différents sommets, on utilise une pile (LIFO: *last in*, *first out)*.

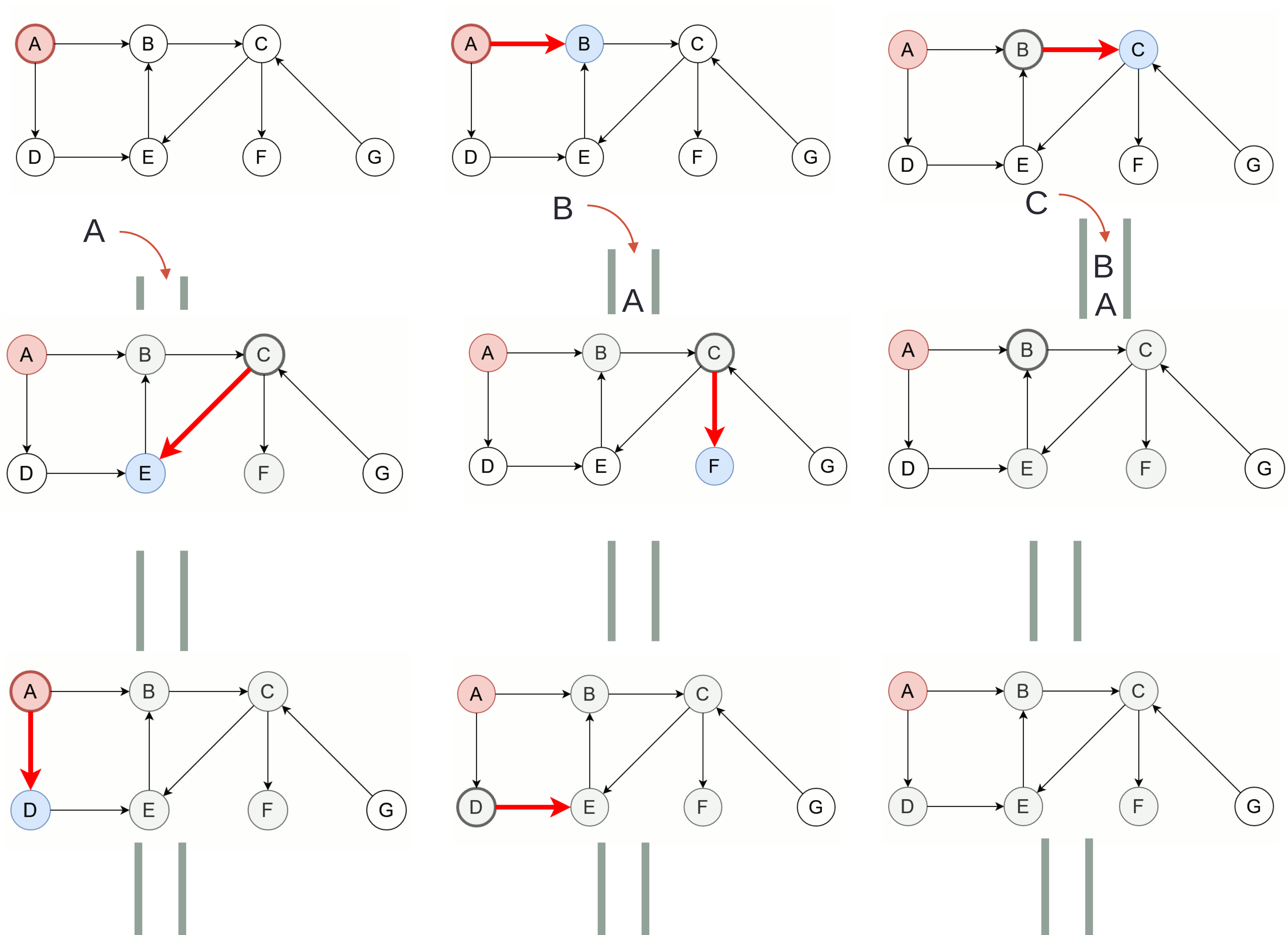
Une pile fonctionne comme une pile d'assiette: les sommets rentrent par le sommet de la pile et en ressortent de la même manière:

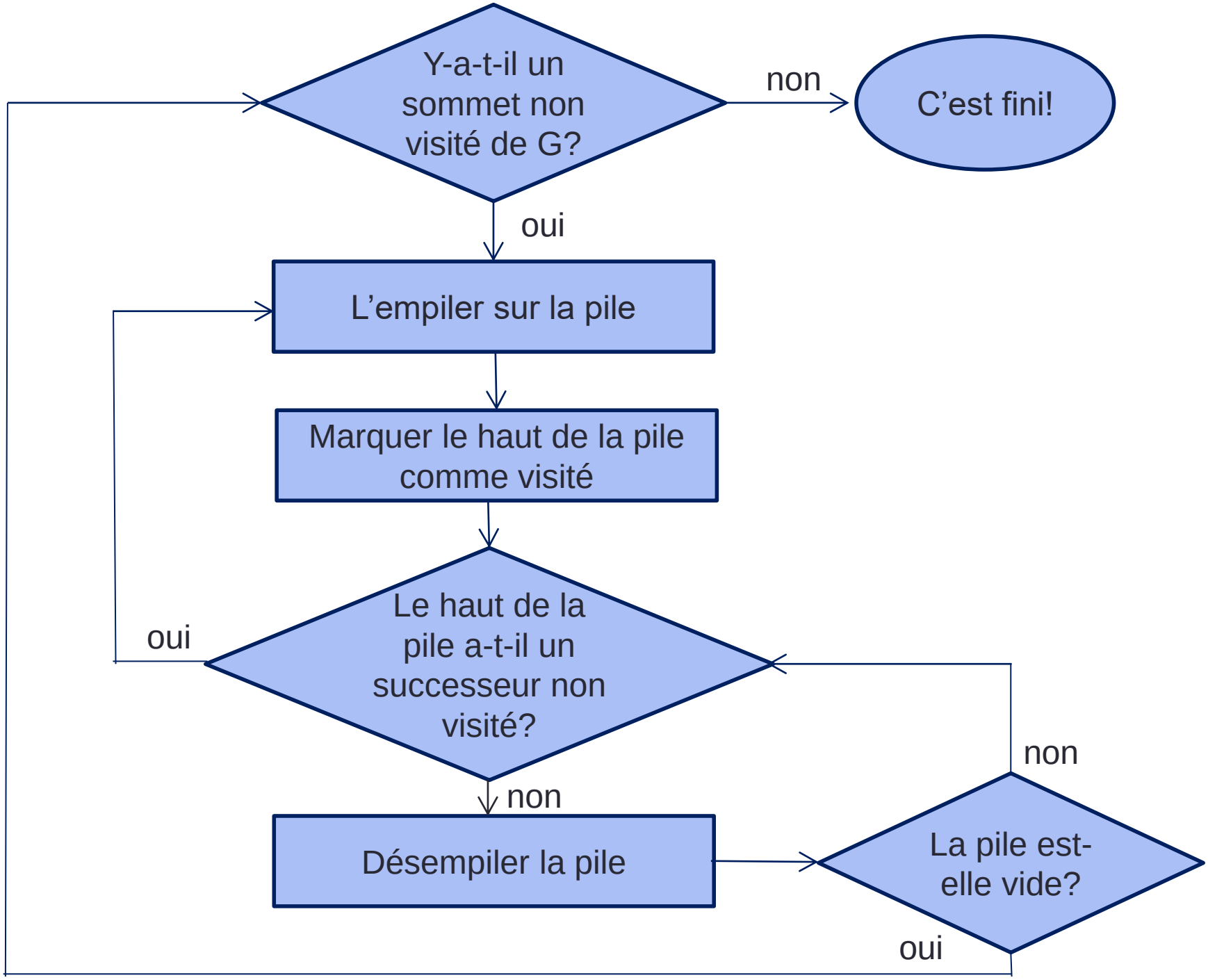
Principe de l'algorithme

1. On choisit au hasard un sommet non encore visité. On l'empile sur une pile.

2. Tant que la pile n'est pas vide:
   a) Le haut de la pile est marqué comme visité.
   b) Si le haut de la pile a des successeurs non visités, on en choisit un au hasard et on l'empile.
   c) S'il n'en a pas, on le désempile de la pile.

3. Si la pile est vide et qu'il reste des sommets non marqués, on recommence à l'étape 1.


A
B
C
D
E
F
G
A
B
A
C
B
A


Y-a-t-il un sommet non visité de G?
non
C'est fini!
oui
L'empiler sur la pile
Marquer le haut de la pile comme visité
Le haut de la pile a-t-il un successeur non visité?
oui
non
Désempiler la pile
La pile est-elle vide?
non
oui

Codage en Python

- Le graphe est représenté par un dictionnaire comme pour le parcours en largeur.

- La file sera représentée à l'aide d'une **liste** `P`:
  - Pour enfiler un sommet s en haut de la pile `P`, on utilise:
  - Pour supprimer le sommet en haut de la pile, on utilise:

- L'état des sommets est représenté par un dictionnaire comme pour le parcours en largeur.

- Le code renvoie un dictionnaire contenant le prédécesseur de chaque sommet. Les racines n'ont pas de prédécesseurs ('`None`').

```python
def parcours_profondeur(G):
    """Parcours en profondeur d'un graphe"""
    etats = dict()
    for s in G:
        etats[s] = False
    pred = dict()
    for r in G:
        if etats[r] == False:
            etats[r] = True
            pred[r] = 'None'
            pile = [r]
            while not (pile == []):
                r = pile[-1] # r est ..................
                suc_list = []
                for s in G[r]:
                    if etats[s] == False:
                        suc_list.append(s) # sucs contient ............
                if suc_list != []:
                    suc = suc_list[0]
                    etats[suc] = ..............
                    pred[suc] = ..............
                    ..............
                else:
                    ..............
    return pred
```