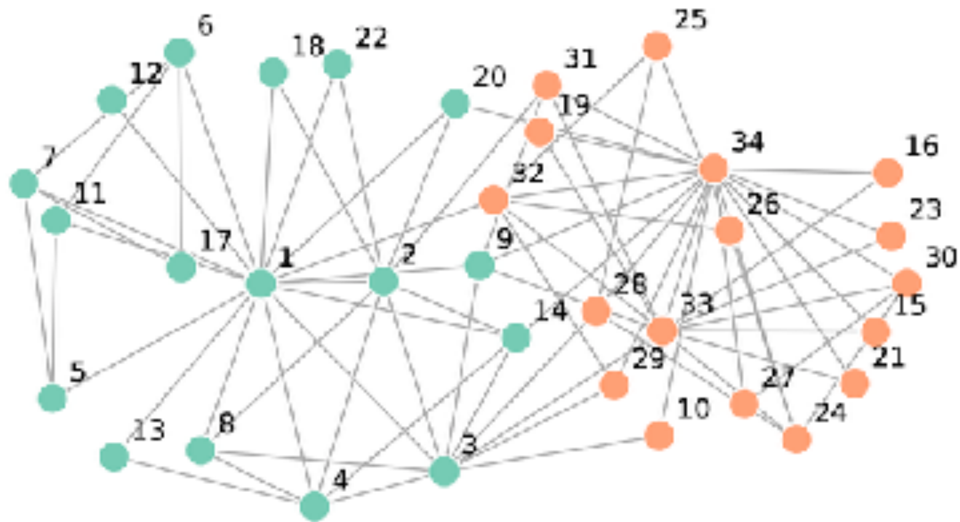
EPFL



Information, Cacul et Communication

Partie Programmation

Cours 13 : Graphes

15.12.2023

Patrick Wang

Au sommaire cette semaine

1. Graphes
2. Files et piles
3. Algorithmes BFS et DFS

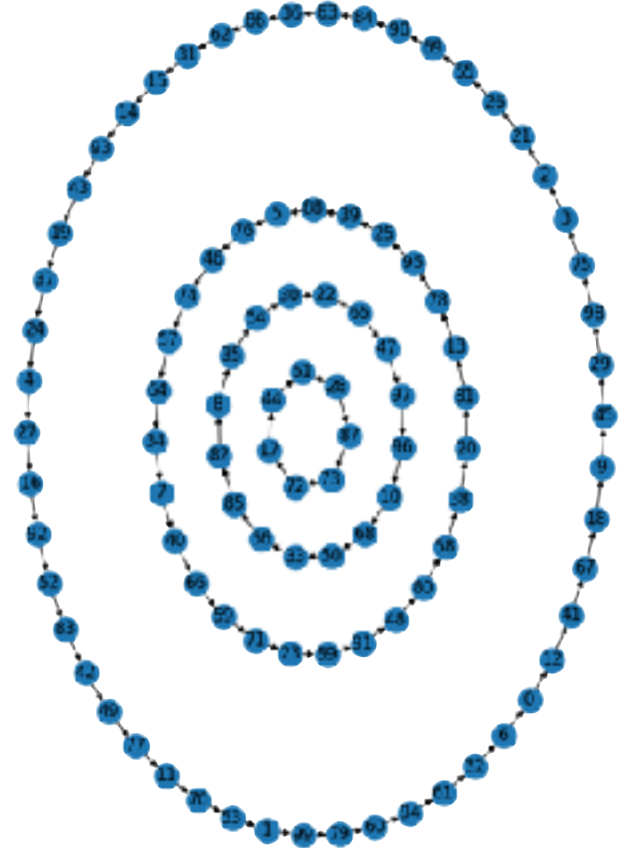
Au sommaire cette semaine

1. Graphes
2. Files et piles
3. Algorithmes BFS et DFS

1. Graphes

Exemples de graphes vus dans le semestre

- Cours de théorie :
  - Problème du voyageur de commerce (*traveling salesman problem*)
  - Internet
- Cours de programmation :
  - Problème des 100 prisonniers

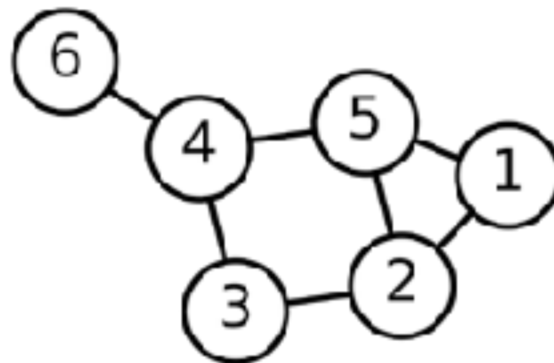


1. Graphes

Définition d’un graphe

- Un graphe *G* est défini par :
  - Un ensemble *V* de sommets ou nœuds («vertices» en anglais)
  - Un ensemble *E* d’arêtes reliant ces sommets («edges» en anglais)

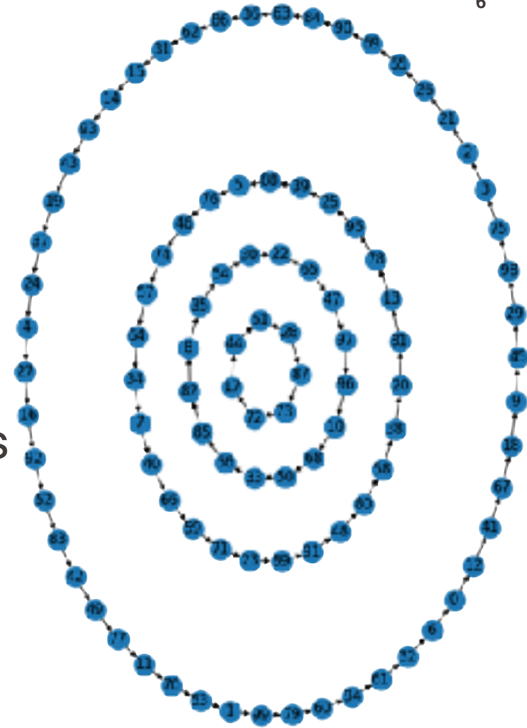
- Par exemple, le graphe représenté ci-contre possède 6 sommets et 7 arêtes.

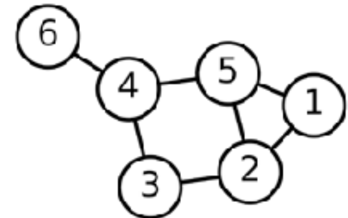


1. Graphes

Propriétés d'un graphe : orientation

- Un graphe peut être orienté ou non-orienté (on dit parfois aussi dirigé)
- Si un graphe est orienté, alors ses arêtes ont un sens
- Ces deux types de graphes permettent de représenter des interactions différentes :
  - Réseau social Twitter : graphe orienté (on peut «follow» sans être suivi en retour)
  - Réseau social Facebook : graphe non-orienté (on est forcément mutuellement amis)





1. Graphes

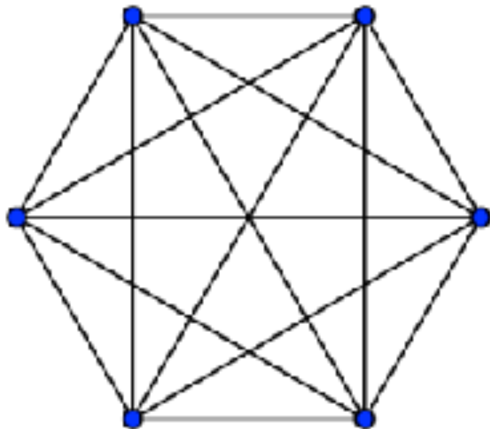
Propriétés d'un graphe : pondération

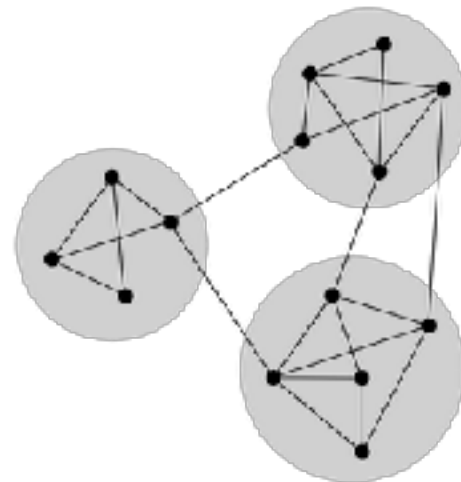
- Un graphe peut être pondéré ou non
- Les arêtes peuvent avoir un poids afin de représenter une grandeur :
  - Distance entre deux points
  - Coût entre deux points
  - Probabilité pour aller d'un point à un autre
  - etc.

1. Graphes

Graphe complet, clique

- Un graphe est dit complet lorsque chaque sommet est relié à tous les autres
- Dans un graphe quelconque, une clique est un sous-ensemble de sommets qui forme un sous-graphe complet

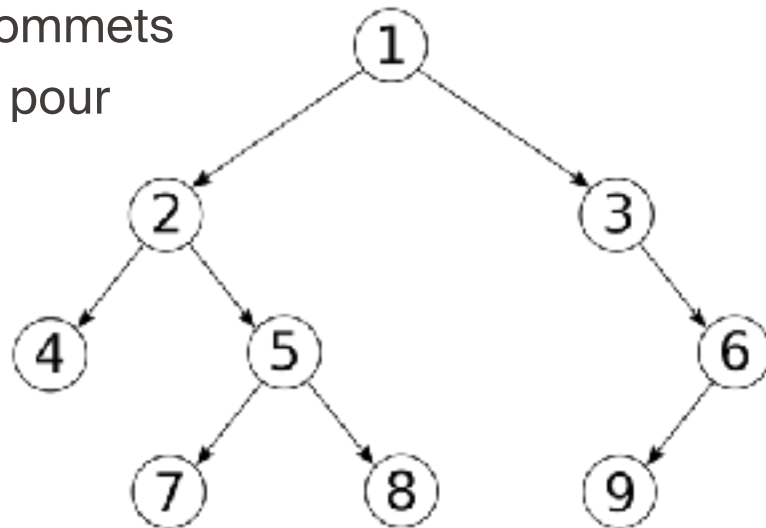




1. Graphes

Un type de graphe particulier : les arbres

- Un arbre est graphe dans lequel tout couple de sommet n'est relié que par un unique chemin
- Encore plus particulier : les arbres binaires dans lesquels chaque sommet possède au plus deux sous-sommets
- On utilise les termes racines et feuilles pour parler des extrémités de l'arbre



1. Graphes

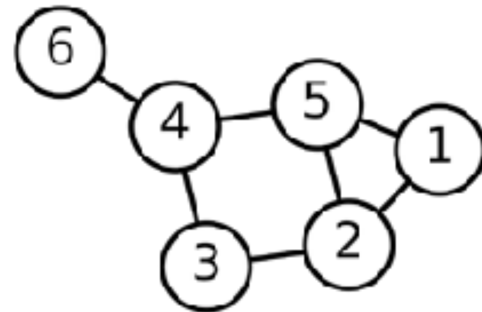
Représentation d'un graphe : matrice d'adjacence

- On peut représenter un graphe grâce à sa matrice d'adjacence M où

$$m_{ij} = \begin{cases} 1 & \text{si et seulement si les sommets } i \text{ et } j \text{ sont reliés} \\ 0 & \text{sinon} \end{cases}$$

- Avec le graphe de l'illustration :

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



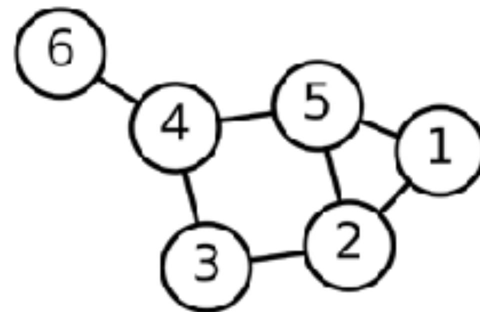
1. Graphes

Représentation d'un graphe : matrice d'adjacence

- On peut représenter un graphe grâce à sa matrice d'adjacence M où

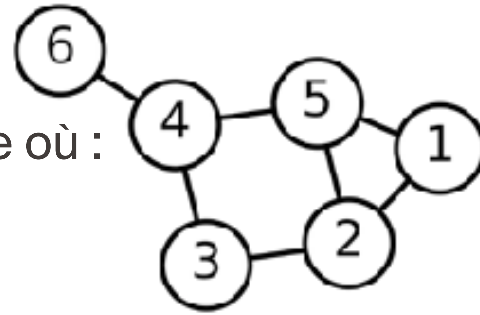
$$m_{ij} = \begin{cases} 1 & \text{si et seulement si les sommets } i \text{ et } j \text{ sont reliés} \\ 0 & \text{sinon} \end{cases}$$

- Avec un graphe pondéré, les m_{ij} peuvent prendre comme valeur le poids des arêtes
- Certaines arêtes peuvent former des boucles, et cela peut se retrouver sur la diagonale de la matrice d'adjacence



1. Graphes

Représentation d'un graphe : listes d'adjacence et dictionnaire



- Autre façon de représenter les choses : un dictionnaire où :
  - Une clé est un sommet ;
  - La valeur sont les sommets adjacents
- Cela peut se représenter de la façon suivante :

```
G: Dict[int, List[int]] = {1: [2, 5],
                           2: [1, 3, 5],
                           3: [2, 4],
                           4: [3, 5, 6],
                           5: [1, 2, 4],
                           6: [4]}
```

EPFL

Au sommaire cette semaine

1. Graphes
2. Files et piles
3. Algorithmes BFS et DFS

2. Files et piles

Encore des nouvelles structures de données

- Les algorithmes BFS (*breadth-first search*) et DFS (*depth-first search*) reposent sur des structures de données que nous ne connaissons pas encore : les files et les piles

2. Files et piles

First In First Out : fonctionnement d'une file

- Structure de données basée sur les listes :
  - Fonctionne comme une file d'attente
  - «Premier arrivé, premier servi»
  - On ajoute des éléments en fin de file
  - On retire les éléments en début de file
- Avec Python :

```
a = [randint(0, 30) for _ in range(20)]
a.append(100)
a.pop(0) # Retire le premier élément ET le retourne
```

EPFL

2. Files et piles

Last In First Out : fonctionnement d'une pile

- Structure de données basée sur les piles :
  - Fonctionne comme un «tas sur lequel on empile des choses»
  - «Dernier arrivé, premier servi»
  - On ajoute des éléments «en haut de la pile»
  - On retire les éléments «en haut de la pile»
- Avec Python :

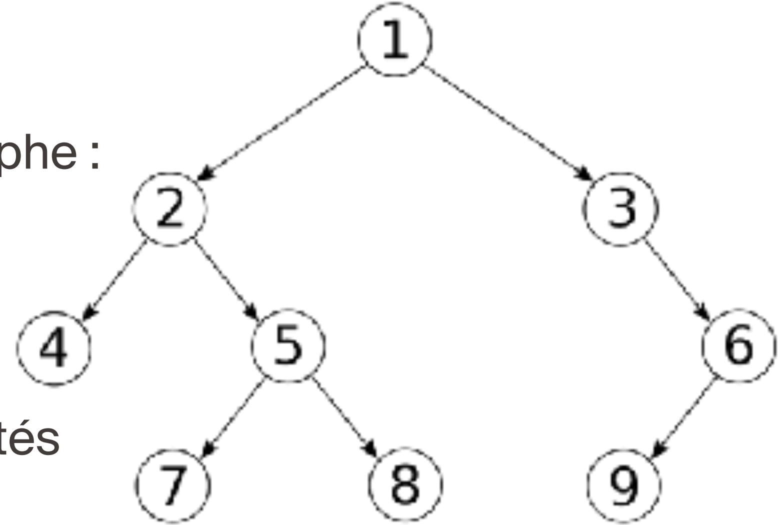
```
a = [randint(0, 30) for _ in range(20)]
a.append(100)
a.pop(-1) # Retire le dernier élément de la pile ET le retourne
```

Au sommaire cette semaine

1. Graphes
2. Files et piles
3. Algorithmes BFS et DFS

3. Algorithmes BFS et DFS

Parcours d'un graphe en largeur ou en profondeur



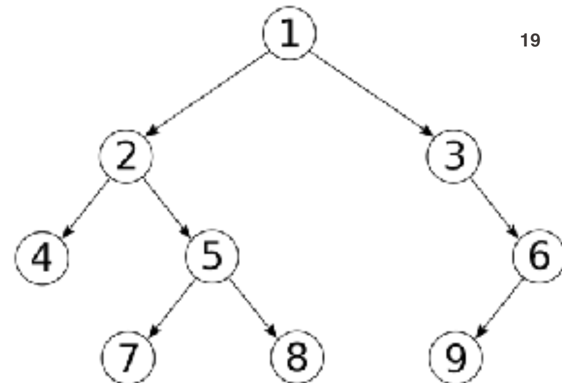
- Deux algorithmes pour parcourir un graphe :
  - en largeur (BFS)
  - en profondeur (DFS)

- Pour l'algorithme BFS, les sommets visités sont (dans l'ordre) : 1, 2, 3, 4, 5, 6, 7, 8, 9

- Pour l'algorithme DFS, les sommets visités sont (dans l'ordre) : 1, 3, 6, 9, 2, 5, 8, 7, 4

3. Algorithmes BFS et DFS

Présentation de l'algorithme BFS



- Idée : Avoir une file des futurs sommets à explorer
- À chaque sommet, ajouter les sommets connectés dans la file (s'il le faut)

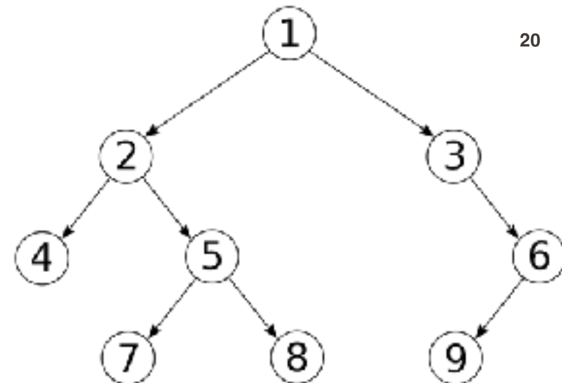
```
def bfs(g, start):
    visited: List[int] = []
    deque: List[int] = [start]
    while len(deque) > 0:
        sommet = deque.pop(0)
        for voisin in g[sommet]:
            if voisin not in visited:
                deque.append(voisin)
        if sommet not in visited:
            visited.append(sommet)
    print(visited)
```

Bloc qui ajoute les voisins du sommet en train d'être parcouru à la file des futurs sommets à explorer

EPFL

3. Algorithmes BFS et DFS

Présentation de l'algorithme BFS



- Idée : Avoir une file des futurs sommets à explorer
- À chaque sommet, ajouter les sommets connectés dans la file (s'il le faut)

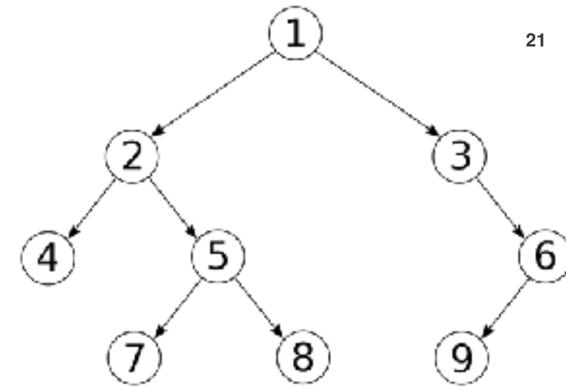
```
def bfs(g, start):
    visited: List[int] = []
    deque: List[int] = [start]
    while len(deque) > 0:
        sommet = deque.pop(0)
        for voisin in g[sommet]:
            if voisin not in visited:
                deque.append(voisin)
        if sommet not in visited:
            visited.append(sommet)
    print(visited)
```

Bloc qui ajoute les sommets parcourus à la liste des sommets visités

EPFL

3. Algorithmes BFS et DFS

Présentation de l'algorithme BFS



- Idée : Avoir une file des futurs sommets à explorer
- À chaque sommet, ajouter les sommets connectés dans la file (s'il le faut)

```
def bfs(g, start):
    visited: List[int] = []
    deque: List[int] = [start]
    while len(deque) > 0:
        sommet = deque.pop(0)
        for voisin in g[sommet]:
            if voisin not in visited:
                deque.append(voisin)
        if sommet not in visited:
            visited.append(sommet)
    print(visited)
```

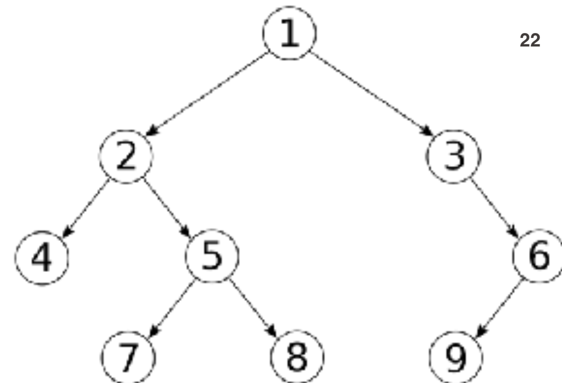
Tant qu'il y a des sommets à visiter :

- Bloc qui ajoute les voisins du sommet en train d'être parcouru à la file des futurs sommets à explorer

- Bloc qui ajoute les sommets parcourus à la liste des sommets visités

3. Algorithmes BFS et DFS

Présentation de l'algorithme BFS



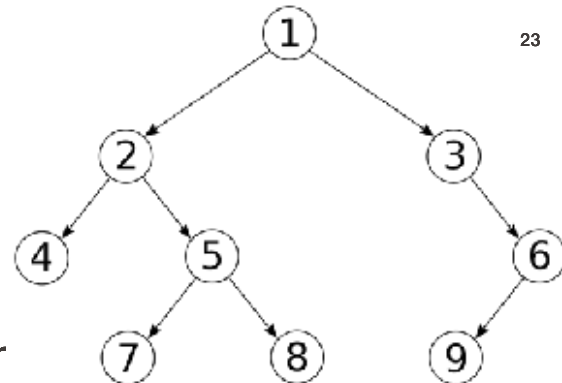
- Idée : Avoir une file des futurs sommets à explorer
- À chaque sommet, ajouter les sommets connectés dans la file (s'il le faut)

```
def bfs(g, start):
    visited: List[int] = []
    deque: List[int] = [start]
    while len(deque) > 0:
        sommet = deque.pop(0)
        for voisin in g[sommet]:
            if voisin not in visited:
                deque.append(voisin)
        if sommet not in visited:
            visited.append(sommet)
    print(visited)
```

EPFL

3. Algorithmes BFS et DFS

Présentation de l'algorithme DFS



- Idée : Avoir une pile des futurs sommets à explorer
- À chaque sommet, ajouter les sommets connectés dans la file (s'il le faut)

```
def dfs(g, start):
    visited: List[int] = []
    deque: List[int] = [start]
    while len(deque) > 0:
        sommet = deque.pop(-1)
        for voisin in g[sommet]:
            if voisin not in visited:
                deque.append(voisin)
        if sommet not in visited:
            visited.append(sommet)
    print(visited)
```

EPFL

Bilan du semestre

Les concepts vus en cours

- Variables (types, opérateurs)
- Structures de contrôle : structures conditionnelles et itératives
- Fonctions («simples», anonymes, d'ordre supérieur)
- Structures de données : `List`, `Tuple`, `Dict`, `Set`, `@dataclass`
- Fichiers en lecture et en écriture

EPFL

Bilan du semestre

Compétences recherchées

- Lire et anticiper le résultat d'un programme
- Concevoir un algorithme (en *langue naturelle*) avant de l'implémenter :
  - Penser à utiliser les mots-clés importants : SI... , RÉPÉTER..., TANT QUE...
  - *Stepwise refinement* : préciser la conception au fur et à mesure
  - Repérer des «structures» courantes (parcours de listes, recherche d'éléments particuliers, etc.)
- Connaître la syntaxe des instructions en Python pour implémenter ces algorithmes