Chapitre V

Algorithmes pour les graphes

Nous allons tout d'abord donner une défintion rigoureuse de la notion de graphe, apprendre le vocabulaire qui y est associé et prouver quelques propriétés basiques sur les graphes.
Puis nous nous intéresserons à la représentation des graphes informatiquement puis à l'élaboration d'algorithmes classiques basés sur ces représentations : parcours de graphes, détection de cycles, plus court chemin dans un graphe.

Partie A

Graphes

1. Graphes simples non-orientés

a. Définitions et vocabulaire

Définition 1. *Graphe non-orienté*

On appelle **graphe non-orienté** un couple $G = (V, E)$ où V un ensemble et E un ensemble de singletons et de paires d'éléments de V.
On appelle les élements de V **les sommets du graphe** et ceux de l'ensemble E, **les arêtes du graphe**.

Notation 1.

Soit V un ensemble et $v, s \in V$. On note $v — s$ ou encore vs l'ensemble $\{v, s\}$.

Notation 1.

Soit V un ensemble et $v, s \in V$. On note $v — s$ ou encore vs l'ensemble $\{v, s\}$.

Remarque 1.

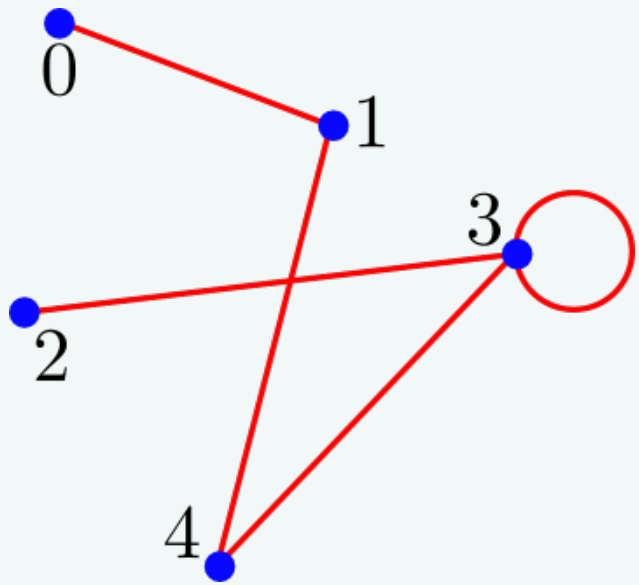
On a $v — s = s — v$.

Exemple 1. *Représentation graphique d'un graphe*

Soit $G = (V, E)$ un graphe non-orienté où

$V = \{0, 1, 2, 3, 4\}$ et $E = \{0 — 1,\ 1 — 4,\ 2 — 3,\ 3 — 3,\ 3 — 4\}$

On peut représenter graphiquement ce graphe G de la façon suivante :



Définition 2. *Vocabulaire et notations relatifs aux graphes*

Soit $G = (V, E)$ un graphe.

- On appelle **ordre du graphe** G, le cardinal de V i.e. le nombre de sommets du graphe.
- On dit que deux sommets v, s de V sont **adjacents** si v — s appartient à E. Dans ce cas, on dit que s et v **sont réliés par une arête** ou qu'**une arête relie** s et v.
- On dit qu'une arête de E est une **boucle** si c'est un singleton i.e. si elle relie un sommet et lui-même.
- Une arête de E qui relie deux sommets de V est dite **incidente** à chacun de ces deux sommets.
- On appelle **degré** d'un sommet de V le nombre d'arêtes de E incidentes à ce sommet - *une boucle est doublement incidente*.

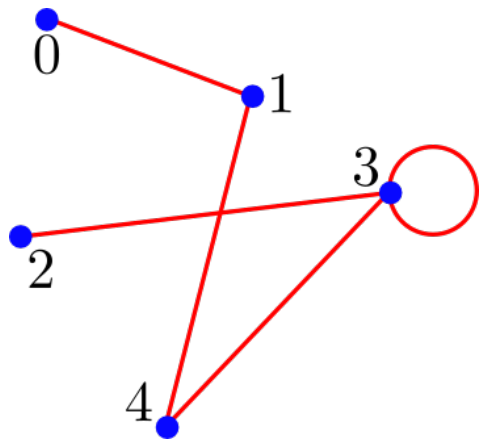
Définition 3. *Graphe non-orienté simple*

On dit qu'un graphe non-orienté est **simple** si son ensemble d'arête ne comporte pas de boucle.

Définition 3. *Graphe non-orienté simple*

On dit qu'un graphe non-orienté est **simple** si son ensemble d'arête ne comporte pas de boucle.

Exercice 1.



Le graphe représenté ci-dessus est-il simple ? Quel est son ordre ? Quel sont les sommets adjacents à 3 ? Quel est le degré de 1 ? de 3 ?

b. Graphe complet

Définition 4.

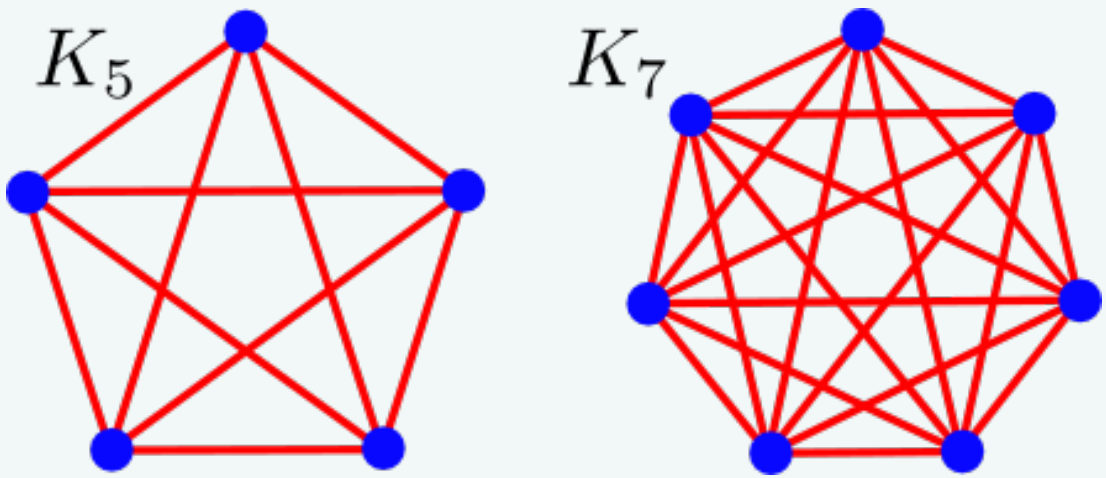
Soit $n \in \mathbb{N}^*$. On appelle **graphe complet** et on note K_n un graphe non-orienté simple d'ordre n dont tous les sommets sont adjacents.

b. Graphe complet

Définition 4.

Soit $n \in \mathbb{N}^*$. On appelle **graphe complet** et on note K_n un graphe non-orienté simple d'ordre n dont tous les sommets sont adjacents.

Exemple 2. *Représentations de graphes complets*



Exercice 2.

Soit $n \in \mathbb{N}^*$.

1. Quel est le degré de chaque sommet dans K_n ? Combien y a-t-il d'arêtes dans K_n ?
2. En déduire une majoration du nombre d'arêtes d'un graphe non-orienté simple d'ordre n.

Exercice 3.

Écrire une fonction `K(n)` qui trace une représentation graphique du graphe K_n dont les sommets seront régulièrement espacés sur un cercle.

c. Propriétés élémentaires

Théorème 1.

Soit $G = (V, E)$ un graphe non-orienté simple. La somme des degrés des sommets de V est égale au double du nombre d'arêtes de E.

c. Propriétés élémentaires

Théorème 1.

Soit $G = (V, E)$ un graphe non-orienté simple. La somme des degrés des sommets de V est égale au double du nombre d'arêtes de E.

Corollaire 1.

Soit $G = (V, E)$ un graphe non-orienté simple. Le nombre de sommets de degré impair est pair.

2. Chaîne de sommets

a. Vocabulaire

Définition 5. *Chaîne*

Soit $G = (V, E)$ un graphe non-orienté, $n \in \mathbb{N}$ et $v_0, ...v_n \in V$. On dit que le $n+1$-uplet $c = (v_0, ..., v_n)$ est une **chaîne** ou encore un **chemin**, si, pour tout $i \in [\![0, n-1]\!]$, si v_i — $v_{i+1} \in E$.
De plus, on dit que :

- la chaîne c **relie** le sommet v_0 au sommet v_n.
- les sommets v_0 et v_n sont les **extrémités** de la chaîne c.
- la chaîne c est de **longueur** n et on note $\ell(c) = n$.

Définition 6. *Chaînes particulières*

Soit $G = (V, E)$ un graphe non-orienté, $n \in \mathbb{N}^*$. On dit qu'une chaîne $c = (v_0, ...v_n)$ est :

- **élémentaire** si $v_i \neq v_j$, pour tous $i \neq j$ avec $i, j \in [\![0, n]\!]$.
- **simple** si les arêtes correspondantes à deux sommets successifs sont toutes différentes.
- **fermée** si $v_n = v_0$;
- un **cycle** si c'est une chaîne simple et fermée.

b. Distance dans un graphe

Définition 7.

Soit $G = (V, E)$ un graphe non-orienté et $v, v' \in V$. On dit que le sommet v' est **accessible** depuis le sommet v s'il existe une chaîne qui relie x et y.

b. Distance dans un graphe

Définition 7.

Soit $G = (V, E)$ un graphe non-orienté et $v, v' \in V$. On dit que le sommet v' est **accessible** depuis le sommet v s'il existe une chaîne qui relie x et y.

Lemme 1.

Si v' est accessible depuis v, alors l'ensemble

$$\{\ell(c) \mid c \text{ chaîne reliant } v \text{ à } v'\}$$

possède un plus petit élément.

b. Distance dans un graphe

Définition 7.

Soit $G = (V, E)$ un graphe non-orienté et $v, v' \in V$. On dit que le sommet v' est **accessible** depuis le sommet v s'il existe une chaîne qui relie x et y.

Lemme 1.

Si v' est accessible depuis v, alors l'ensemble

$$\{\ell(c) \mid c \text{ chaîne reliant } v \text{ à } v'\}$$

possède un plus petit élément.

Définition 8. *Distance entre deux sommets*

Soit $G = (V, E)$ un graphe non-orienté et $v, v' \in V$. On appelle **distance** entre v et v' et on note $d(v, v')$ la quantité :

$$d(v, v') = \begin{cases} \min\{\ell(c) \mid c \text{ chaîne reliant } v \text{ à } v'\} & \text{si } v' \text{ est accessible depuis } v \\ +\infty & \text{sinon.} \end{cases}$$

Proposition 1.

Soit $G = (V, E)$ un graphe non-orienté. Pour tous $v, v', v'' \in V$, on a :

- $d(v, v) = 0$
- $d(v, v') = d(v', v)$
- $d(v, v'') \leq d(v, v') + d(v', v'')$

c. Graphes connexes

Proposition 2.

Soit $G = (V, E)$ un graphe non-orienté. La relation d'accessibilité est une relation d'équivalence sur V.

c. Graphes connexes

Proposition 2.

Soit $G = (V, E)$ un graphe non-orienté. La relation d'accessibilité est une relation d'équivalence sur V.

Définition 9.

Soit $G = (V, E)$ un graphe non-orienté. Une classe d'équivalence pour la relation d'accessibilité est appelée **composante connexe** du graphe G.

Définition 10. *Graphe connexe*

Soit $G = (V, E)$ un graphe non-orienté. On dit que G est **connexe** s'il ne possède qu'une seule composante connexe.

Autrement dit, G est connexe si, pour tous $v, v' \in V$, il existe une chaîne qui relie v et v'.

d. Chaînes eulériennes

Définition 11. *Chaîne eulérienne*

Soit $G = (V, E)$ un graphe non-orienté et c une chaîne.
On dit que c est une **chaîne eulérienne** si c est une chaîne simple telle que toutes les arêtes du graphe y sont représentées.
Si de plus, c est fermée, on dit que c est un **cycle eulérien**.

Théorème 2.

Soit $G = (V, E)$ un graphe non-orienté connexe. Le graphe G admet une chaîne eulérienne si, et seulement si, le nombre de sommets de degré impair est égal à 0 ou 2.

Théorème 2.

Soit $G = (V, E)$ un graphe non-orienté connexe. Le graphe G admet une chaîne eulérienne si, et seulement si, le nombre de sommets de degré impair est égal à 0 ou 2.

Exercice 4.

Soit $G = (V, E)$ un graphe non-orienté connexe.

1. Donner une condition nécéssaire et suffisante pour que G admette un cycle eulérien.
2. On suppose que G possède deux sommets de degré impair. Quelles peuvent être les extrémités d'une chaîne eulérienne de G ?

3. Graphes orientés

Définition 12. *Graphe orienté*

On appelle **graphe orienté** un couple $G = (V, E)$ où V un ensemble (les sommets de G) et E un ensemble (les arêtes orientés de G) de *couples* d'éléments de V.
Si $e = (v, v') \in E$ est une arête,

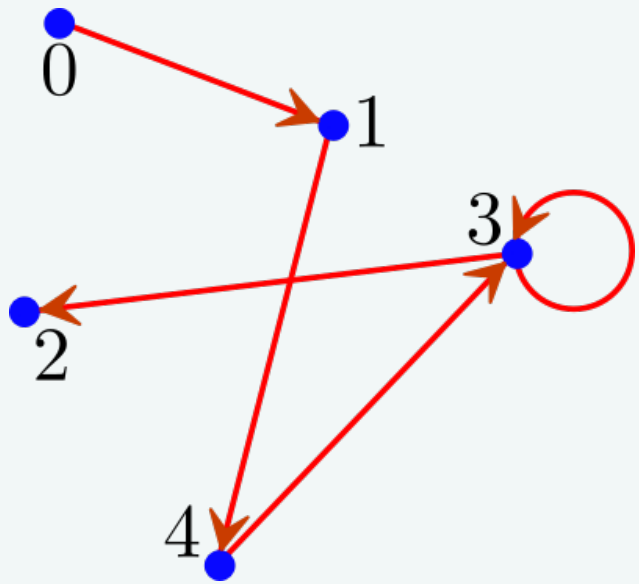
- on dit que e relie v **à** v' et on note $e = v \to v'$;
- on dit que e est une arête sortante de v - ce qui permet de définir le *degré sortant* d'un sommet ;
- on dit que e est une arête entrante de v' - ce qui permet de définir le *degré entrant* d'un sommet.

Exemple 3. *Représentation graphique d'un graphe orienté*

Soit $G = (V, E)$ un graphe orienté où :

$V = \{0, 1, 2, 3, 4\}$ et $E = \{0 \to 1,\ 1 \to 4,\ 3 \to 2,\ 3 \to 3,\ 4 \to 3\}$

On peut représenter graphiquement ce graphe G de la façon suivante :

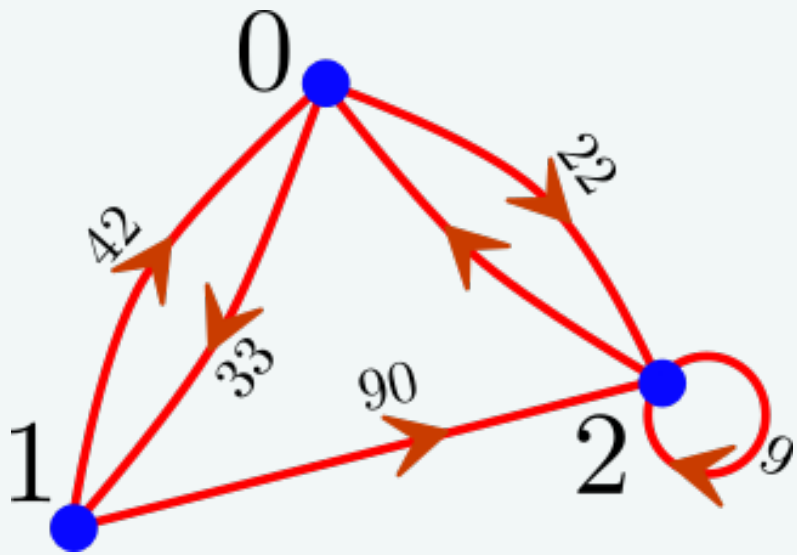


a. Graphe pondéré

Définition 13. *Graphe pondéré*

On appelle **graphe pondéré** un triplet $G_p = (V, E, \rho)$ tel que $G = (V, E)$ est un graphe orienté et $\rho : E \to \mathbb{R}$ est une fonction, appelée **fonction de pondération des arêtes**.
Si $e \in E$ est une arête, la quantité $\rho(e)$ est appelé **poids de l'arête** e.

Exemple 4. *Exemple de graphe pondéré*



L'arête $0 \to 2$ a pour pondération 22 i.e. $\rho(0 \to 2) = 22$.

Partie B

Représentations d'un graphe

1. Représentation matricielle

a. Matrice d'adjacence

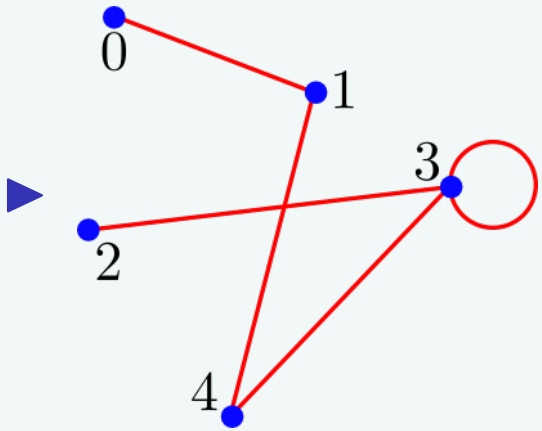
Définition 14.

Soit $G = (V, E)$ un graphe d'ordre n dont on numérote les sommets de 0 à $n-1$. On appelle **matrice d'adjacence** de G la matrice $M = (m_{i,j})_{0 \leq i,j \leq n-1}$ carrée d'ordre n telle que pour tout $i, j \in [\![0, n-1]\!]$,

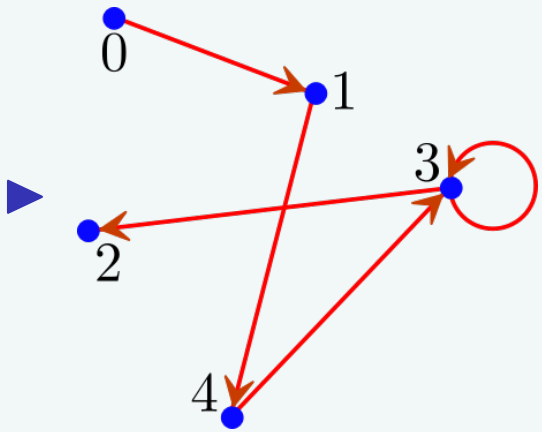
$$m_{ij} = \begin{cases} 0 \text{ s'il n'y a pas d'arête reliant } i \text{ et } j \\ 1 \text{ s'il y a une arête reliant } i \textbf{ à } j \end{cases}$$

Exemple 5.

Voici deux exemples de matrices d'adjacence de graphes :



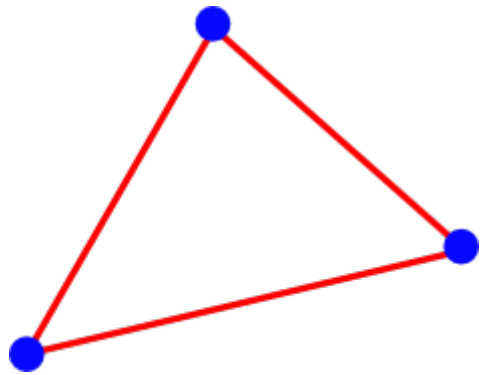
$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$



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

Exercice 5.

1. Que peut-on dire sur les coefficients diagonaux d'une matrice d'adjacence d'un graphe simple ? Que peut-on dire des coefficients de part et d'autre de la diagonale d'une matrice d'adjacence d'un graphe non-orienté ?
2. Déterminer la matrice d'adjacence du graphe suivant :



3. Soit $M = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$. Déterminer un graphe qui possède M pour matrice d'adjacence.

Théorème 3.

Soit G un graphe d'ordre n, $M = (m_{ij})$ sa matrice d'adjacence et $p \in \mathbb{N}^*$.
Pour tout $i, j \in [\![1, n]\!]$, le nombre de chaînes de longueur p reliant i à j est égale au coefficient de la i-ième ligne et j-ième colonne de M^p.

Exercice 6.

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

Écrire un programme qui trace un graphe (on placera les 7 sommets régulièrement espacés sur un cercle) dont la matrice d'adjacence est M puis déterminer le nombre de chaînes de longueur **au plus** 10 reliant 0 à 6.

b. Matrice de transition d'un graphe pondéré

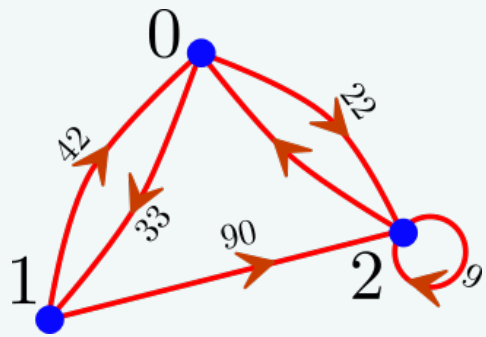
Définition 15.

Soit $G_p = (V, E, \rho)$ un graphe pondéré d'ordre n dont on numérote les sommets de 0 à $n-1$. On appelle **matrice de transition** de G_p la matrice carrée $M = (m_{ij})$ d'ordre n telle que pour tout $i, j \in [\![0, n-1]\!]$,

$$m_{ij} = \begin{cases} 0 & \text{s'il n'y a pas d'arête reliant } i \text{ et } j \\ \underbrace{\rho(i,j)}_{\text{poids de l'arête } (i,j)} & \text{s'il y a une arête reliant } i \textbf{ à } j \end{cases}$$

Exemple 6. *Exemple de matrice de transition*

Voici un exemple de graphe pondéré et de sa matrice de transition.



$$M = \begin{pmatrix} 0 & 33 & 22 \\ 42 & 0 & 90 \\ -8 & 0 & 9 \end{pmatrix}$$

2. Représentation par liste d'adjacence

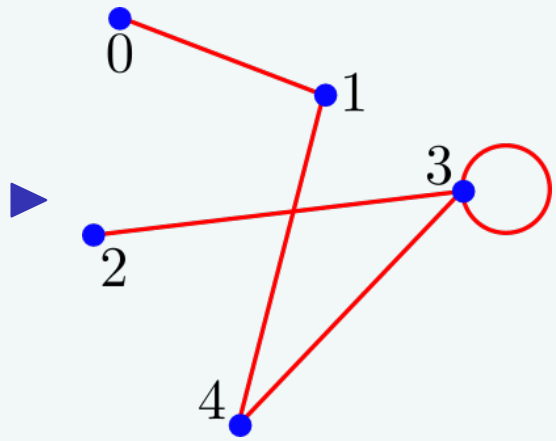
Définition 16. *Liste d'adjacence d'un graphe non pondéré*

Soit $G = (V, E)$ un graphe. On appelle **liste d'adjacence** de G, l'application $L_G : V \to \mathcal{P}(V)$ telle que, pour $v \in V$:

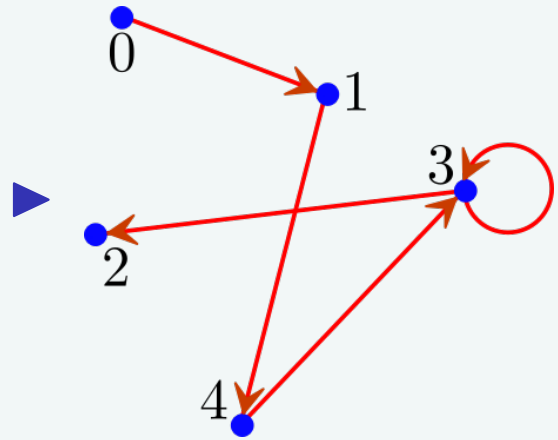
$$L_G(v) = \{v' \in V \mid \text{ il existe un arête reliant } v \text{ à } v'\}$$

Exemple 7.

Voici deux exemples de listes d'adjacence de graphes :



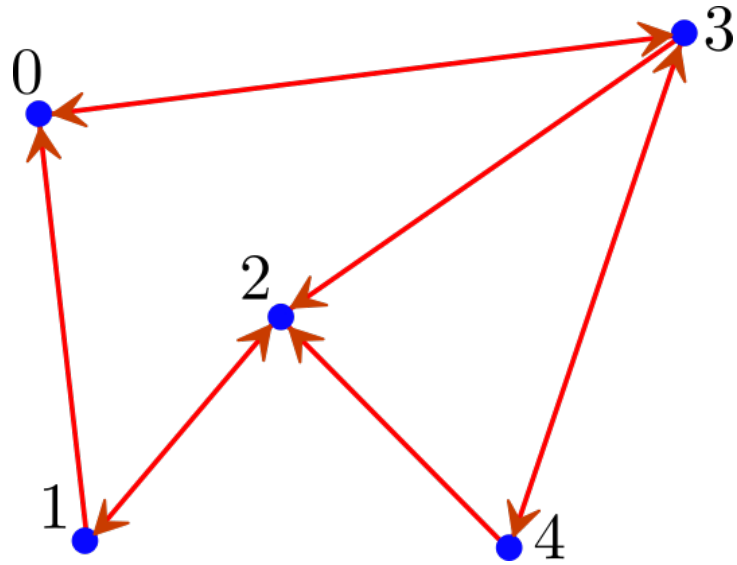
$$L : \begin{cases} 0 & \mapsto \{1\} \\ 1 & \mapsto \{0, 4\} \\ 2 & \mapsto \{3\} \\ 3 & \mapsto \{2, 3, 4\} \\ 4 & \mapsto \{1, 3\} \end{cases}$$



$$L : \begin{cases} 0 & \mapsto \{1\} \\ 1 & \mapsto \{4\} \\ 2 & \mapsto \emptyset \\ 3 & \mapsto \{2, 3\} \\ 4 & \mapsto \{3\} \end{cases}$$

Exercice 7.

1. Déterminer la matrice d'adjacence du graphe suivant :



2. Représenter graphiquement un graphe dont la liste d'adjacence est la suivante :

$$L : \begin{cases} 0 & \mapsto \{2, 4\} \\ 1 & \mapsto \{1, 4\} \\ 2 & \mapsto \{1, 2\} \\ 3 & \mapsto \{0, 1, 2, 4\} \\ 4 & \mapsto \{0, 3\} \end{cases}$$

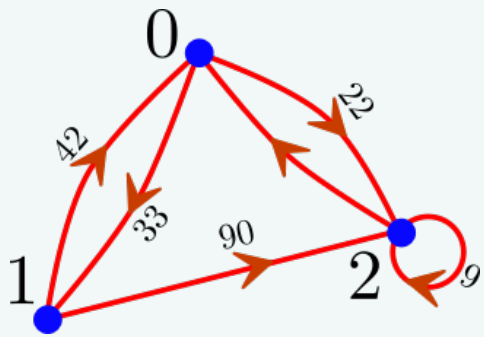
Définition 17. *Liste d'adjacence d'un graphe pondéré*

Soit $G_p = (V, E, \rho)$ un graphe pondéré. On appelle **liste d'adjacence** de G_p, l'application $L_G : V \to \mathcal{P}(\mathbb{R} \times V)$ telle que, pour $v \in V$:

$$L_G(v) = \{\big(\rho(v \to v'), v'\big) \in V \mid \text{ il existe un arête reliant } v \text{ à } v'\}$$

Exemple 8. *Exemple de liste d'adjacence d'un graphe pondéré*

Voici un exemple de graphe pondéré et de sa liste d'adjacence.



$$L : \begin{cases} 0 & \mapsto \{(33,1),(22,2)\} \\ 1 & \mapsto \{(42,0),(90,2)\} \\ 2 & \mapsto \{(-8,0),(9,2)\} \end{cases}$$

Partie C

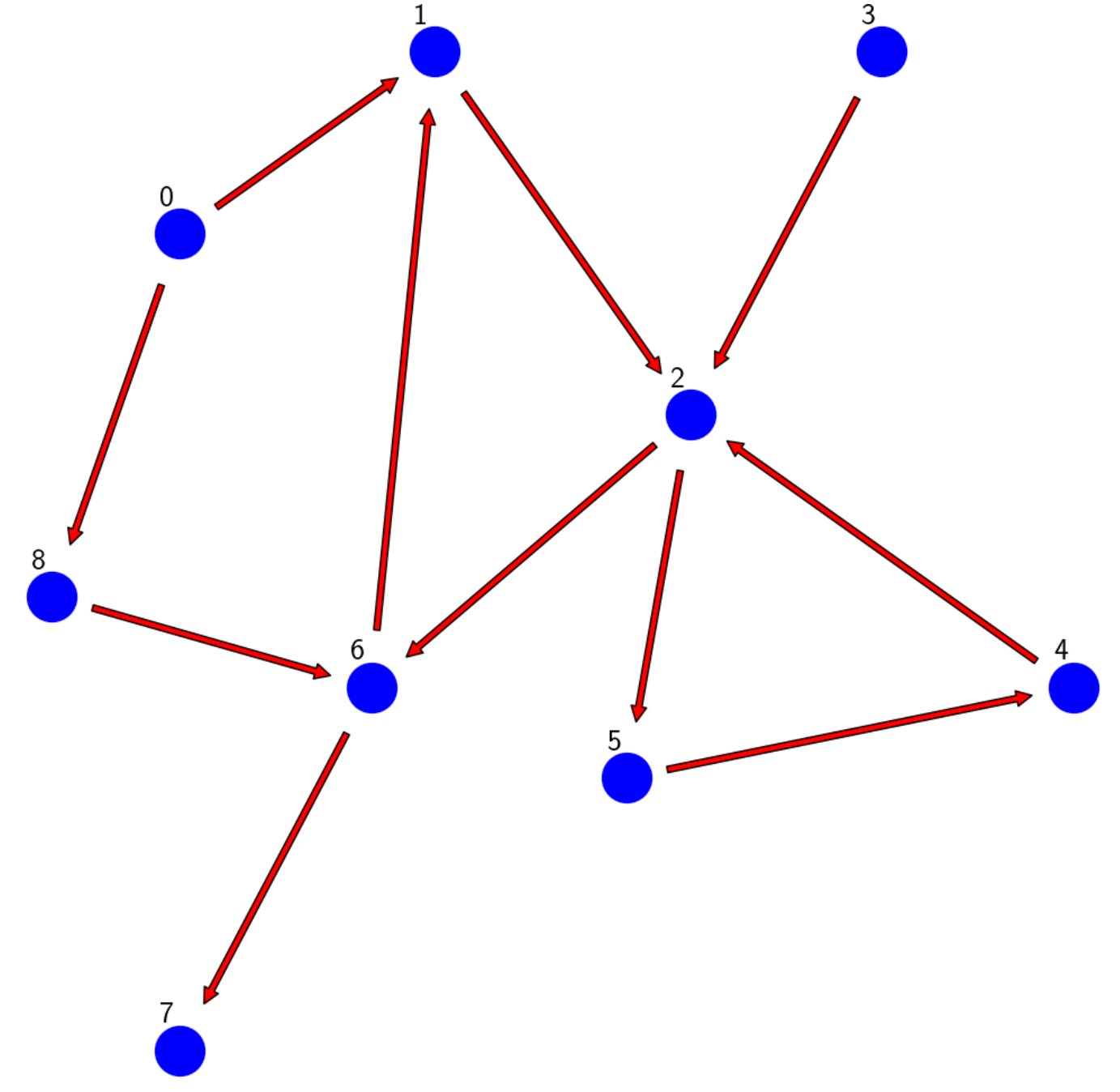
Algorithmes pour les graphes

Dans cette partie, nous allons mettre en place différents algorithmes de parcours des sommets d'un graphe et diverses applications de ces algorithmes.

1. Parcours d'un graphe

En partant d'un sommet donné d'un graphe qu'on appellera la *source*, on veut parcourir les sommets accessibles depuis la source. On munit chaque sommet d'un état qui peut prendre les valeurs "vu" ou "pas vu". On dispose également d'un "sac" dans lequel on pourra stocker et prendre des sommets.

```
mettre la source dans le sac
Tant que le sac n'est pas vide Faire
    prendre x dans le sac
    Si x n'a pas été vu Faire
        marquer x comme vu
        Pour toute arrête x--y
            mettre y dans le sac
        Fin Pour
    Fin Si
Fin Tant que
```



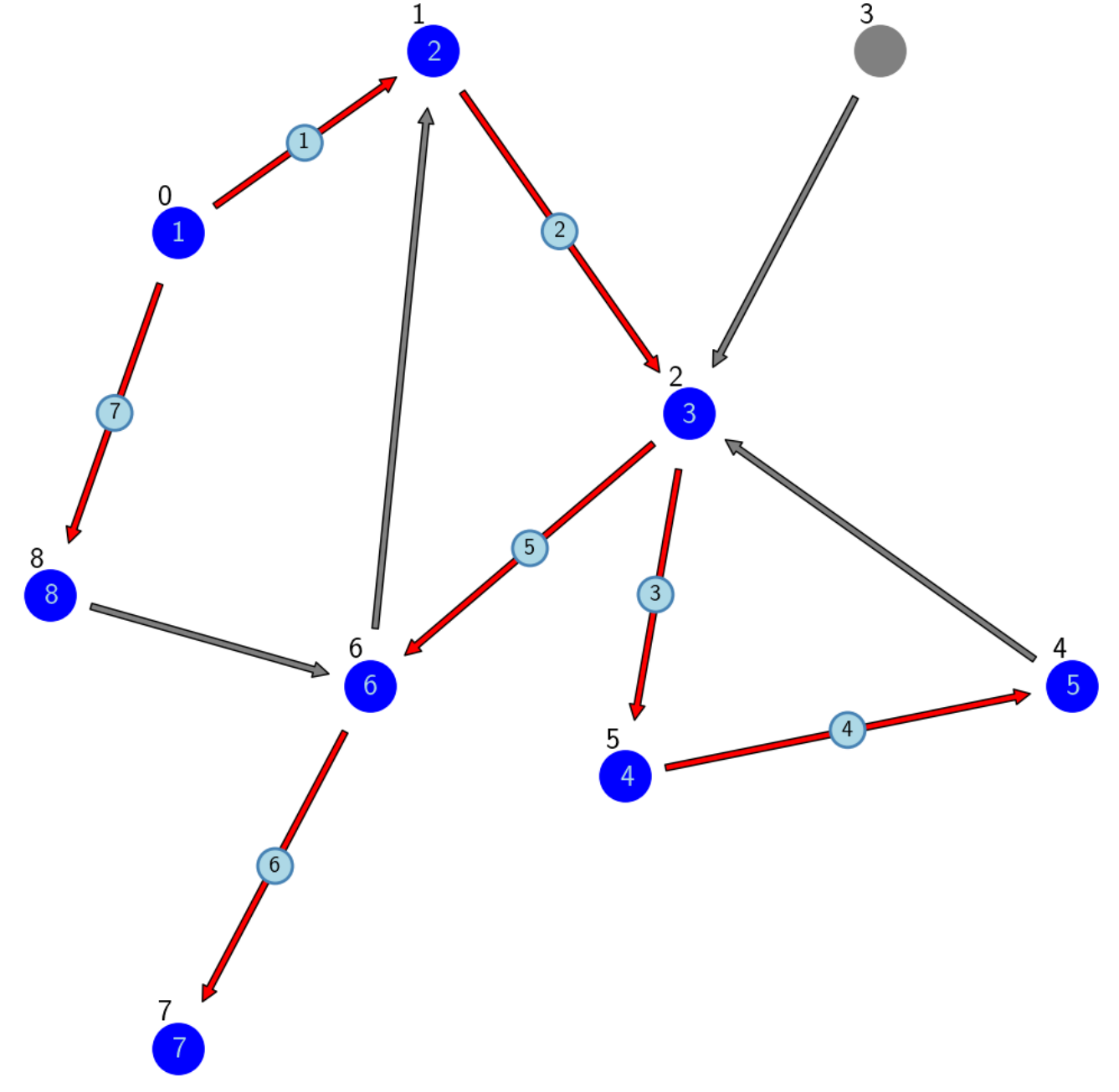
Dans la suite, on appliquera nos différentes implémentations du parcours au graphe ci-dessus.

2. Parcours en profondeur

a. Implémentations

Version itérative

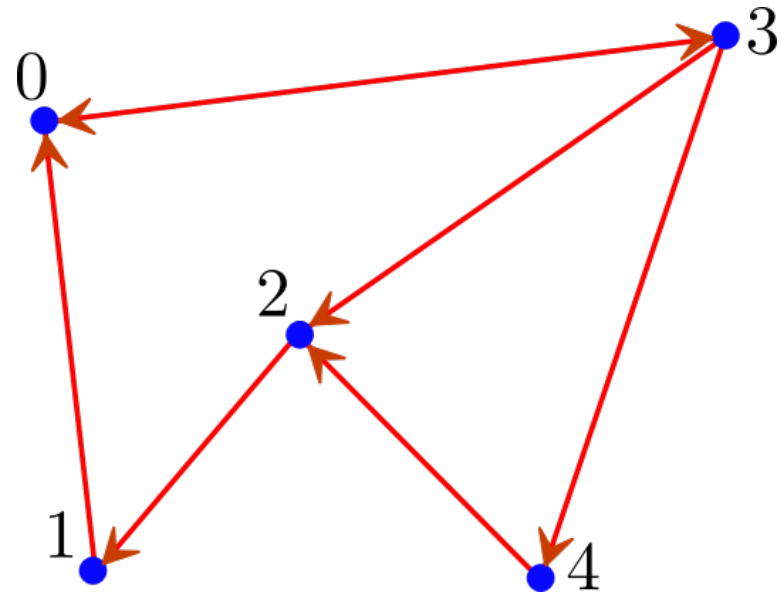
```
1 def parcours_profondeur(G, s):
2     sac_pile = [s]
3     vu = [False for v in range(len(G))]
4     while len(sac_pile) > 0:
5         v = sac_pile.pop() #on prend le sommet en
              haut de la pile.
6         if not vu[v]:
7             vu[v] = True
8             for w in G[v]:
9                 sac_pile.append(w)
```



Exemple du cas récursif du parcours en profondeur : dans les sommets et sur les arêtes sont indiqués les ordres de passage de l'algorithme.

Exercice 8.

On considère le graphe G suivant :



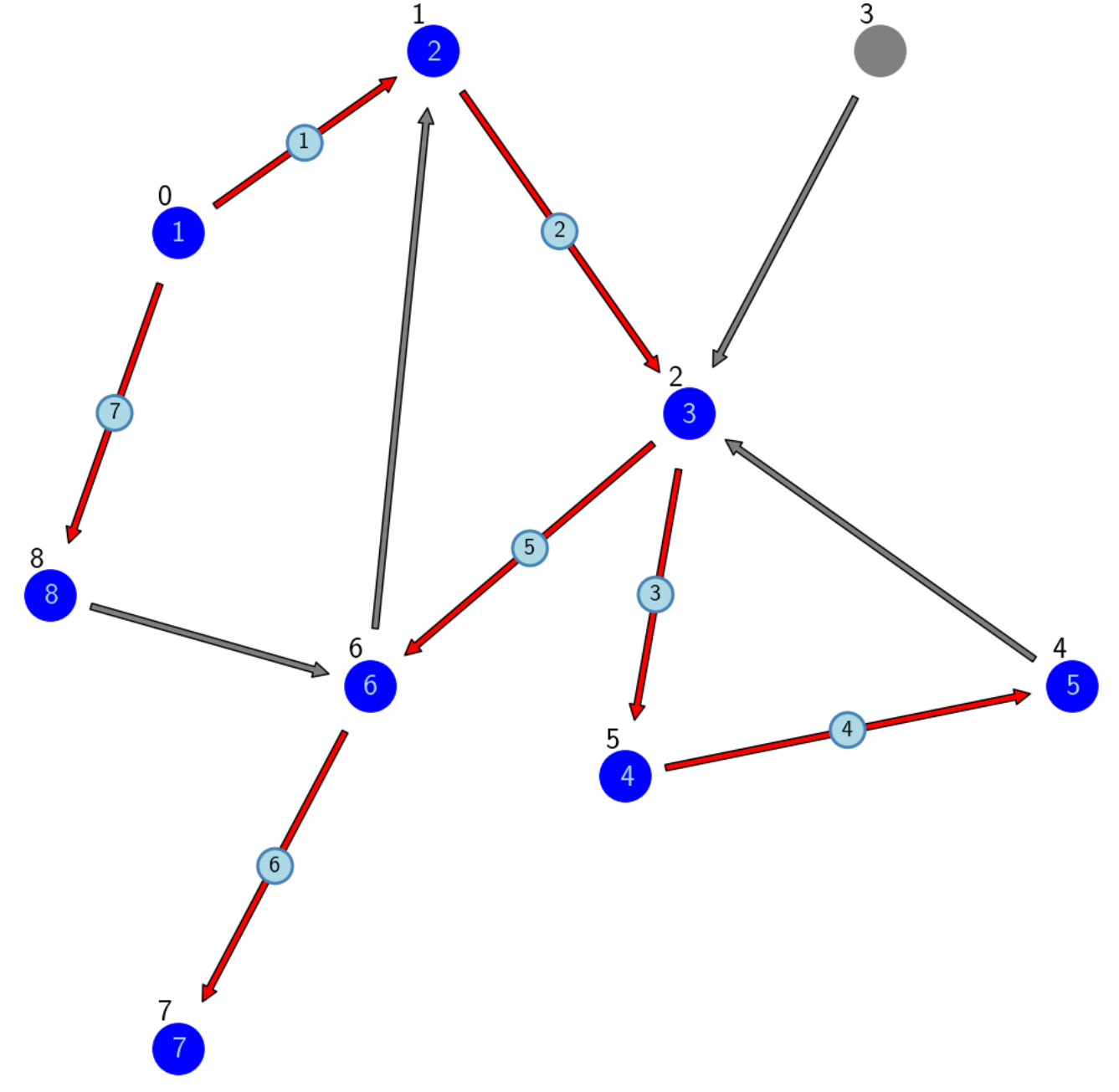
1. Déterminer la liste d'adjacence de ce graphe puis l'implémenter dans une liste `G`.
2. Implémenter le parcours en profondeur en ajoutant l'instruction `print(v)` au moment du marquage "visité" d'un sommet `v` dans le parcours. Puis appliquer le parcours en profondeur au graphe G.

Exercice 9.

Implémenter le parcours en profondeur lorsque le graphe est représenté par une matrice d'adjacence.

Version récursive

```
1 def prof_rec(G, vu, v):
2     if not visite[v]:
3         vu[v] = True
4         for w in G[v]:
5             prof_rec(G, visite, w)
6
7 def parcours_profondeur_rec(G, s):
8     vu = [False for v in range(len(G))]
9     prof_rec(G, vu, s)
```



Exemple du cas récursif du parcours en profondeur : dans les sommets et sur les arêtes sont indiqués les ordres de passage de l'algorithme.

Exercice 10.

Déterminer la complexité de l'algorithme du parcours en profondeur en fonction de l'ordre n du graphe et de son nombre a d'arêtes.

b. Applications

Exercice 11. *Connexité*

1. Écrire une fonction `acces(G,v1,v2)` qui renvoie `True` si `v2` est accessible depuis `v1` et `False` sinon.
2. Écrire une fonction `connexe(G)` qui renvoie `True` si G est connexe et `False` sinon.
3. Écrire une fonction `composantes_connexes(G)` qui renvoie la liste des composantes connexes de G (chacune représentée par la liste des sommets qui la composent).

Exercice 12. *Détection de cycle*

1. Expliquer pourquoi, en l'état, le parcours en profondeur ne permet pas de détecter si un graphe orienté possède un cycle ou non.
2. En rajoutant un troisième état possible aux sommets lors du parcours en profondeur, écrire une fonction récursive `cycle(G)` qui renvoie `True` si G possède un cycle, `False` sinon.

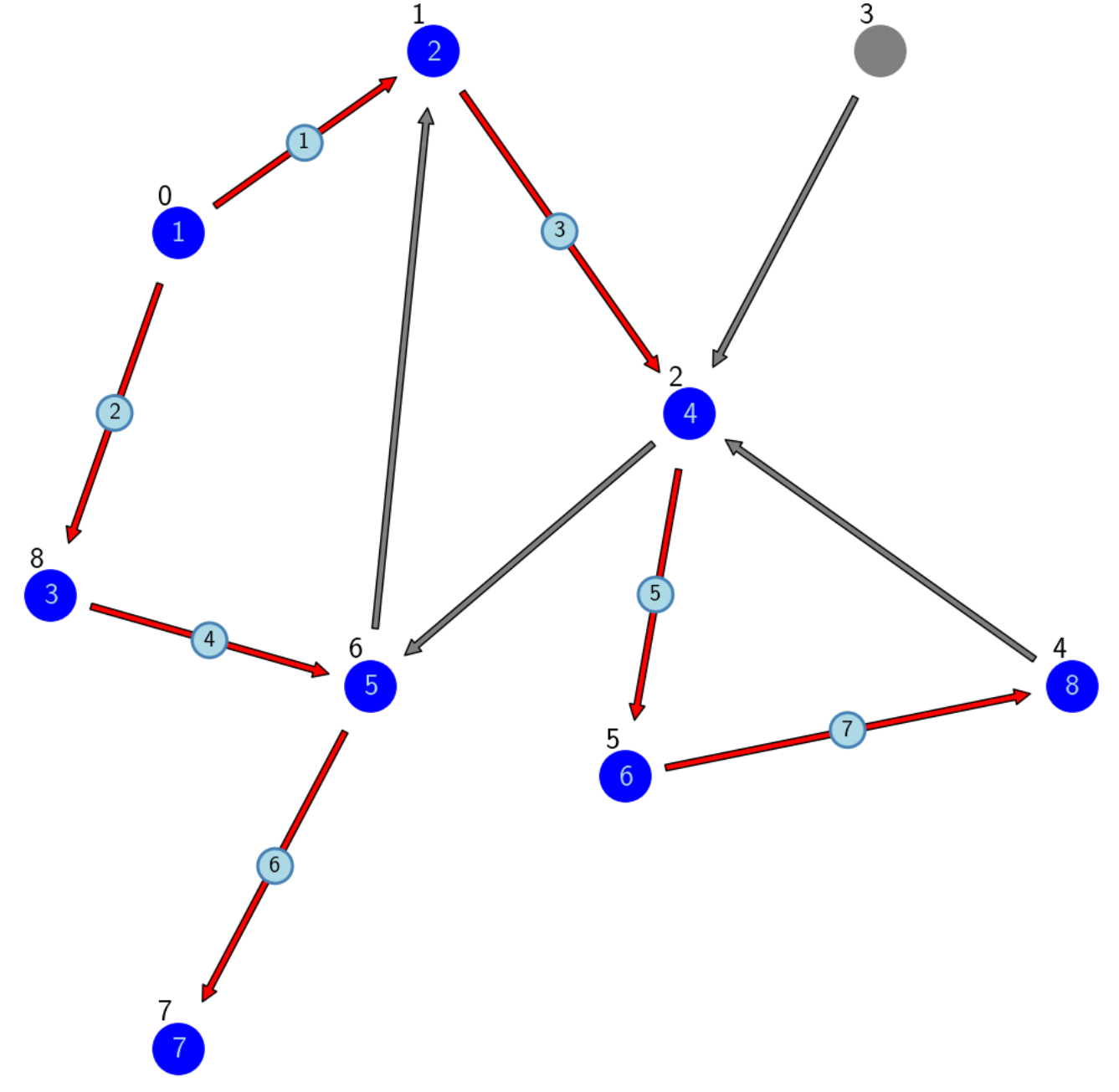
3. Parcours en largeur

Parcours en largeur à marquage tardif

```
1 import collections as c #module permettant l'
      utilisation de file
2
3 def parcours_largeur(G, s):
4     vu = [False for v in range(len(G))]
5     sac_file = c.deque()
6     sac_file.append(s)
7     while len(sac_file) > 0:
8         v = sac_file.popleft()
9         if not vu[v]:
10            vu[v] = True
11            for w in G[v]:
12                sac_file.append(w)
```

Parcours en largeur à marquage précoce

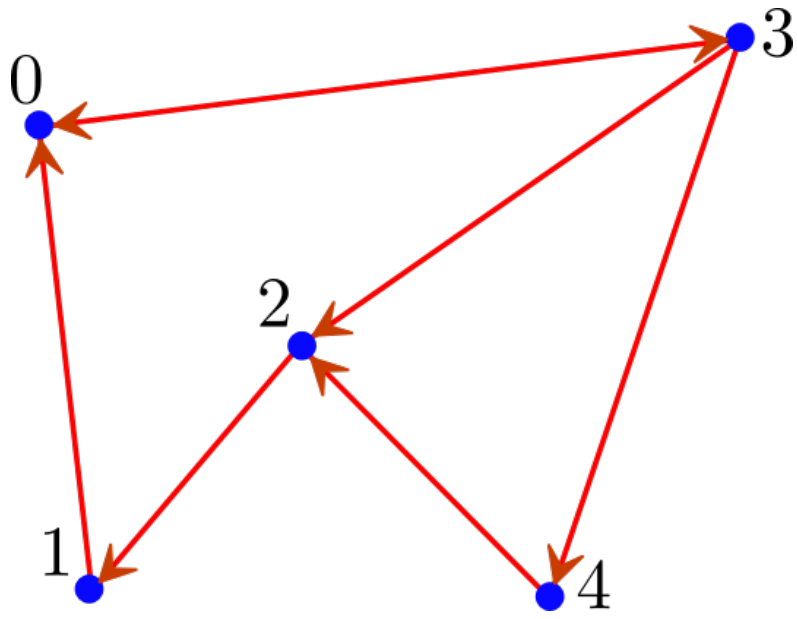
```
1  import collections as c #module permettant l'
       utilisation de file
2
3  def parcours_largeur(G, s):
4      vu = [False for v in range(len(G))]
5      sac_file = c.deque()
6      sac_file.append(s)
7      vu[s] = True
8      while len(sac_file) > 0:
9          v = sac_file.popleft()
10         for w in G[v]:
11             if not vu[w]:
12                 vu[w] = True
13                 sac_file.append(w)
```



Exemple de parcours en largeur : dans les sommets et sur les arêtes sont indiqués les ordres de passage de l'algorithme.

Exercice 13.

On considère le graphe G suivant :



Implémenter le parcours en largeur en ajoutant l'instruction `print(v)` au moment du marquage "vu" d'un sommet `v` dans le parcours. Puis appliquer le parcours en largeur au graphe G.

Exercice 14.

Écrire une fonction `dist(s)` qui renvoie la liste (indicée par les numéros des ommets) des distances de chaque sommet du graphe au sommet `s`. Si un sommet n'est pas accessible depuis `s`, on pourra associer la valeur `None`.

4. Plus court chemin

a. Algorithme de Dijkstra

```
1 def append_priorite(L,couple_poids_element):
2     """ L est une liste de couples (poids,element
          ) triée par ordre décroissant de poids
          """
3     L.append(couple_poids_element)
4     n=len(L)
5     i=n-1
6     while i>0 and L[i][0]>L[i-1][0]:
7         L[i],L[i-1]=L[i-1],L[i]
8         i-=1
```

Algorithme de Dijkstra

```
1  def dijkstra(G, s):
2      n = len(G)
3      vu = [False for v in range(n)]
4      distance = [None for v in range(n)]
5      file_priorite = []
6      append_priorite(file_priorite,(0.0,s))
7      while len(file_priorite) > 0:
8          d,v = file_priorite.pop()
9          if not vu[v]:
10             distance[v] = d
11             vu[v] = True
12         for poids,w in G[v]:
13             append_priorite(file_priorite, (d+
                   poids,w))
14     return distance
```